z/OS
Version 2.Release 5

Cryptographic Services
Integrated Cryptographic Service Facility
Application Programmer's Guide

SC14-7508-10

Note

Before using this information and the product it supports, read the information in “Notices” on page 1729.

This edition applies to ICSF FMID HCR77D2 and Version 2 Release 5 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2023-06-26

Contents

Figures ..... **xiii**

Tables ..... **xv**

About this information ..... **xliii**
Who should use this information xliii
How to use this information xliii
Where to find more information xlv
Related Publications xlv

How to send your comments to IBM ..... **xlvii**
If you have a technical problem xlvii

Summary of changes ..... **xlix**
Changes made in Cryptographic Support for z/OS V2R5 (FMID HCR77D2) xlix
Changes made in Cryptographic Support for z/OS V2R2 - z/OS V2R4 (FMID HCR77D1) lv
Changes made in Cryptographic Support for z/OS V2R2 - z/OS V2R3 (FMID HCR77D0) lx
Changes made in Cryptographic Support for z/OS V2R1 - z/OS V2R3 (FMID HCR77C1) lxiv

Part 1. IBM programming **1**

Chapter 1. Introducing programming for ICSF 3
ICSF callable services naming conventions 3
Callable service syntax 3
Callable services with ALET parameters 4
Rules for defining parameters and attributes 5
Parameter definitions 6
Invocation requirements 9
Security considerations 9
Performance considerations 9
Deprecated callable services 10
Special secure mode 10
Compliance mode 11
Using the callable services 11
When the call succeeds 11
When the call does not succeed 11
Linking a program with the ICSF callable services 12

Chapter 2. Introducing CCA symmetric key cryptography 15
Functions of symmetric cryptographic keys 15
Key separation 16
CCA DES control vectors 16
Key forms 16
Key token 17
CCA key wrapping 20
Payload format 21
Types of keys 21
X9.143 (TR-31) key blocks 28
Key strength and wrapping of key 33

Key strength and key wrapping access control points..... 34
DES master key..... 35
DK PIN methods support..... 35
Diversify Directed Key (CSNBDDK and CSNEDDK)..... 36
DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG)..... 36
DK Migrate PIN (CSNBDMP and CSNEDMP)..... 36
DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)..... 36
DK PAN Translate (CSNBDPT and CSNEDPT)..... 36
DK PIN Change (CSNBDPC and CSNEDPC)..... 36
DK PIN Verify (CSNBDPV and CSNEDPV)..... 36
DK PRW Card Number Update (CSNBDPNU and CSNEDPNU) and DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2)..... 36
DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG)..... 36
DK Random PIN Generate (CSNBDRPG and CSNEDRPG) and DK Random PIN Generate2 (CSNBDRG2 and CSNEDRG2)..... 37
DK Regenerate PRW (CSNBDRP and CSNEDRP)..... 37
Australian Payment Network support..... 37
Generating and managing symmetric keys..... 37
Key Generator Utility Program..... 37
Common Cryptographic Architecture DES Key Management Services..... 37
Common Cryptographic Architecture AES Key Management Services..... 41
Common Cryptographic Architecture HMAC Key Management Services..... 42
ECC Diffie-Hellman key agreement models..... 43
Improved remote key distribution..... 44
Diversifying keys..... 56
Callable services for managing the CKDS..... 57
Callable Services that support Secure Sockets Layer (SSL)..... 59
Enciphering and deciphering data..... 59
Encoding and Decoding Data (CSNBECO, CSNEECO, CSNBDCO, and CSNEDCO)..... 60
Translating Ciphertext (CSNBCTT2 or CSNBCTT3 and CSNECTT2 or CSNECTT3)..... 60
Managing data integrity and message authentication..... 60
Message authentication code processing..... 61
Hashing functions..... 63
Managing personal authentication..... 64
Verifying credit card data..... 64
Format preserving encryption..... 66
EMV simplification services..... 67
Derive ICC Master Key callable service (CSNBDCM and CSNEDCM)..... 67
Derive Session Key callable service (CSNBDSK and CSNEDSK)..... 68
EMV Scripting callable service (CSNBESC and CSNEESC)..... 68
EMV Transaction (ARQC/ARPC) callable service (CSNBEAC and CSNEEAC)..... 68
EMV Verification callable service (CSNBEVF and CSNEEVF)..... 69
Generate Issuer Master Key callable service (CSNBGIM and CSNEGIM)..... 69
ANSI X9.143 (TR-31) key block support..... 70
TR-31 Create Callable Service (CSNBT31C and CSNET31C)..... 70
TR-31 Import Callable Service (CSNBT31I and CSNET31I)..... 70
TR-31 Parse Callable Service (CSNBT31P and CSNET31P) 70
TR-31 Optional Data Read Callable Service (CSNBT31R and CSNET31R)..... 70
TR-31 Optional Data Build Callable Service (CSNBT31O and CSNET31O)..... 71
TR-31 Translate Callable Service (CSNBT31X and CSNET31X)..... 71
Secure messaging..... 71
Trusted Key Entry (TKE) support..... 71
Utilities..... 72
Character/Nibble Conversion Callable Services (CSNBXBC and CSNBXCB)..... 72
Code Conversion Callable Services (CSNBXEA and CSNBXAE)..... 72
Cryptographic Usage Statistic (CSFSTAT and CSFSTAT6)..... 72
ICSF Query Algorithm Callable Service (CSFIQA)..... 72
ICSF Query Facility Callable Service (CSFIQF)..... 72

ICSF Query Facility2 Callable Service (CSFIQF2)....72
X9.9 Data Editing Callable Service (CSNB9ED)....72
Typical sequences of ICSF callable services....72
Key forms and types used in the Key Generate callable service....73
Generating an operational key....73
Generating an importable key....74
Generating an exportable key....74
Examples of single-length keys in one form only....74
Examples of OPIM single-length, double-length, and triple-length keys in two forms....74
Examples of OPEX single-length, double-length, and triple-length keys in two forms....75
Examples of IMEX single-length and double-length keys in two forms....75
Examples of EXEX single-length and double-length keys in two forms....76
Using the Cipher Text Translate2 callable service....76
Summary of callable services....76

Chapter 3. Introducing CCA PKA cryptography and using PKA callable services....93
PKA key algorithms....93
PKA keys....94
Master keys....94
Operational private keys....94
Key strength and wrapping of key....94
Key strength and key wrapping access control points....95
RSA private key tokens....95
PKA callable services....96
Callable services supporting digital signatures....96
Callable services for PKA key management....96
Callable services to manage the Public Key Data Set (PKDS)....97
Callable services for working with retained private keys....99
Callable services for the TR-34....99
Callable services for Secure Electronic Transaction (SET)....100
PKA key tokens....100
X.509 certificates....101
PKA key management....102
Security and integrity of the token....102
Key identifier for PKA key token....103
Key label....103
Key token....103
Summary of the PKA callable services....104

Chapter 4. Introducing PKCS #11 and using PKCS #11 callable services....109
PKCS #11 services....109
Attribute list....111
Handles....111

Part 2. CCA callable services....113

Chapter 5. Managing symmetric cryptographic keys....115
Clear Key Import (CSNBCKI and CSNECKI)....116
Control Vector Generate (CSNBCVG and CSNECVG)....118
Control Vector Translate (CSNBCVT and CSNECVT)....124
Cryptographic Variable Encipher (CSNBCVE and CSNECVE)....128
Data Key Export (CSNBDKX and CSNEDKX)....130
Data Key Import (CSNBDKM and CSNEDKM)....133
Derive ICC MK (CSNBDCM and CSNEDCM)....136
Derive Session Key (CSNBDSK and CSNEDSK)....143
Diversified Key Generate (CSNBDKG and CSNEDKG)....151
Diversified Key Generate2 (CSNBDKG2 and CSNEDKG2)....162

Diversify Directed Key (CSNBDDK and CSNEDDK)....170
ECC Diffie-Hellman (CSNDEDH and CSNFEDH).... 182
Generate Issuer MK (CSNBGIM and CSNEGIM).... 197
Key Encryption Translate (CSNBKET and CSNEKET).... 204
Key Export (CSNBKEX and CSNEKEX).... 207
Key Generate (CSNBKGN and CSNEKGN).... 211
Key Generate2 (CSNBKGN2 and CSNEKGN2)....224
Key Import (CSNBKIM and CSNEKIM).... 237
Key Part Import (CSNBKPI and CSNEKPI).... 242
Key Part Import2 (CSNBKPI2 and CSNEKPI2).... 247
Key Test (CSNBKYT and CSNEKYT).... 252
Key Test2 (CSNBKYT2 and CSNEKYT2)....256
Key Test Extended (CSNBKYTX and CSNEKYTX).... 267
Key Token Build (CSNBKTB and CSNEKTB)....271
Key Token Build2 (CSNBKTB2 and CSNEKTB2).... 297
Key Translate (CSNBKTR and CSNEKTR)....343
Key Translate2 (CSNBKTR2 and CSNEKTR2).... 346
Multiple Clear Key Import (CSNBCKM and CSNECKM).... 355
Multiple Secure Key Import (CSNBSKM and CSNESKM).... 358
PKA Decrypt (CSNDPKD and CSNFPKD)....365
PKA Encrypt (CSNDPKE and CSNFPKE).... 374
Prohibit Export (CSNBPEX and CSNEPEX).... 384
Prohibit Export Extended (CSNBPEXX and CSNEPEXX).... 386
Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL).... 388
Remote Key Export (CSNDRKX and CSNFRKX).... 393
Restrict Key Attribute (CSNBRKA and CSNERKA).... 402
Secure Key Import (CSNBSKI and CSNESKI)....408
Secure Key Import2 (CSNBSKI2 and CSNESKI2).... 412
Symmetric Key Export (CSNDSYX and CSNFSYX)....417
Symmetric Key Export with Data (CSNDSXD and CSNFSXD)....426
Symmetric Key Generate (CSNDSYG and CSNFSYG).... 430
Symmetric Key Import (CSNDSYI and CSNFSYI)....439
Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2).... 444
Trusted Block Create (CSNDTBC and CSNFTBC).... 450
Unique Key Derive (CSNBUKD and CSNEUKD)....454

Chapter 6. Protecting data.... 469
Modes of operation.... 469
Electronic Code Book (ECB) Mode.... 470
Cipher Block Chaining (CBC) Mode.... 470
Cipher Feedback (CFB) Mode....470
Output Feedback (OFB) Mode....470
Galois/Counter Mode (GCM)....470
Triple DES Encryption.... 471
Cipher Text Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3)....471
Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)....484
Decode (CSNBDCO and CSNEDCO).... 490
Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1).... 492
Encode (CSNBECO and CSNEECO).... 499
Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)....501
Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1).... 509
Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)....521
Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1).... 531

Chapter 7. Verifying data integrity and authenticating messages.... 543
How MACs are used.... 543
How hashing functions are used....544
How MDCs are used....544

HMAC Generate (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)..... 545
HMAC Verify (CSNBHMV or CSNBHMV1 and CSNEHMV or CSNEHMV1)..... 550
MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)..... 555
MAC Generate2 (CSNBMGN2, CSNBMGN3, CSNEMGN2, and CSNEMGN3)..... 562
MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)..... 567
MAC Verify2 (CSNBMVR2, CSNBMVR3, CSNEMVR2, and CSNEMVR3)..... 574
MDC Generate (CSNBMDG or CSNBMDG1 and CSNEMDG or CSNEMDG1)..... 579
One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1)..... 584
Symmetric MAC Generate (CSNBSMG or CSNBSMG1 and CSNESMG or CSNESMG1)..... 589
Symmetric MAC Verify (CSNBSMV or CSNBSMV1 and CSNESMV or CSNESMV1)..... 594

Chapter 8. Financial services..... 601
How Personal Identification Numbers (PINs) are used..... 602
How VISA card verification values are used..... 602
Translating data and PINs in networks..... 602
Working with Europay–MasterCard–Visa smart cards..... 602
PIN callable services..... 603
Generating a PIN..... 603
Encrypting a PIN..... 603
Generating a PIN Validation Value (PVV) from an encrypted PIN block..... 603
Verifying a PIN..... 603
Translating a PIN..... 604
Algorithms for generating and verifying a PIN..... 604
Using PINs on different systems..... 605
PIN-encrypting keys..... 605
ANSI X9.8 PIN restrictions..... 606
ANSI X9.8 PIN - Enforce PIN block restrictions..... 606
ANSI X9.8 PIN - Allow modification of PAN..... 607
ANSI X9.8 PIN - Allow only ANSI PIN blocks..... 607
ANSI X9.8 PIN – Use stored decimalization tables only..... 607
Enhanced PIN Security..... 608
Enhanced PIN security mode..... 608
Enhanced PIN checking for CSNBPTR and CSNBPTR2..... 609
PIN block error processing mode..... 609
The PIN profile..... 609
PIN block format..... 610
Format control..... 612
Pad digit..... 612
Current key serial number..... 613
Decimalization tables..... 614
Format preserving encryption..... 614
Authentication Parameter Generate (CSNBAPG and CSNEAPG)..... 620
Clear PIN Encrypt (CSNBCPE and CSNECPE)..... 624
Clear PIN Generate (CSNBPGN and CSNEPGN)..... 629
Clear PIN Generate Alternate (CSNBCPA and CSNECPA)..... 634
CVV Key Combine (CSNBCKC and CSNECKC)..... 640
EMV Scripting Service (CSNBESC and CSNEESC)..... 646
EMV Transaction (ARQC/ARPC) Service (CSNBEAC and CSNEEAC)..... 657
EMV Verification Functions (CSNBEVF and CSNEEVF)..... 665
Encrypted PIN Generate (CSNBEPG and CSNEEPG)..... 671
Encrypted PIN Translate (CSNBPTR and CSNEPTR)..... 678
Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)..... 685
Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)..... 701
Encrypted PIN Verify (CSNBPVR and CSNEPVR)..... 713
Encrypted PIN Verify2 (CSNBPVR2 and CSNEPVR2)..... 720
Field Level Decipher (CSNBFLD and CSNEFLD)..... 730
Field Level Encipher (CSNBFLE and CSNEFLE)..... 739
Format Preserving Algorithms Decipher (CSNBFFXD and CSNEFFXD)..... 749

Format Preserving Algorithms Encipher (CSNBFFXE and CSNEFFXE)....754
Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT)....760
FPE Decipher (CSNBFPED and CSNEFPED)....767
FPE Encipher (CSNBFPEE and CSNEFPEE)....776
FPE Translate (CSNBFPET and CSNEFPET)....786
PIN Change/Unblock (CSNBPCU and CSNEPCU)....795
Recover PIN from Offset (CSNBPFO and CSNEPFO)....806
Secure Messaging for Keys (CSNBSKY and CSNESKY)....811
Secure Messaging for PINs (CSNBSPN and CSNESPN)....816
SET Block Compose (CSNDSBC and CSNFSBC)....823
SET Block Decompose (CSNDSBD and CSNFSBD)....828
Transaction Validation (CSNBTRV and CSNETRV)....834
VISA CVV Service Generate (CSNBCSG and CSNECSG)....838
VISA CVV Service Verify (CSNBCSV and CSNECSV)....843

Chapter 9. Financial services for DK PIN methods....849
Weak PIN table....849
DK PIN methods....849
DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG)....850
DK Migrate PIN (CSNBDMP and CSNEDMP)....857
DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)....863
DK PAN Translate (CSNBDPT and CSNEDPT)....871
DK PIN Change (CSNBDPC and CSNEDPC)....878
DK PIN Verify (CSNBDPV and CSNEDPV)....892
DK PRW Card Number Update (CSNBDPNU and CSNEDPNU)....897
DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2)....904
DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG)....912
DK Random PIN Generate (CSNBDRPG and CSNEDRPG)....916
DK Random PIN Generate2 (CSNBDRG2 and CSNEDRG2)....922
DK Regenerate PRW (CSNBDRP and CSNEDRP)....930

Chapter 10. X9.143 (TR-31) symmetric-key management....937
TR-31 Create (CSNBT31C and CSNET31C)....938
TR-31 Import (CSNBT31I and CSNET31I)....959
TR-31 Optional Data Build (CSNBT31O and CSNET31O)....998
TR-31 Optional Data Read (CSNBT31R and CSNET31R)....1001
TR-31 Parse (CSNBT31P and CSNET31P)....1004
TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)....1007

Chapter 11. TR-34 symmetric key management....1059
TR-34 protocol....1059
User flows....1059
Setup....1060
BIND....1062
UNBIND....1064
REBIND....1065
2-pass key transport....1066
1-pass key transport....1069
TR-34 Bind-Begin (CSNDT34B and CSNFT34B)....1072
TR-34 Bind-Complete (CSNDT34C and CSNFT34C)....1079
TR-34 Key Distribution (CSNDT34D and CSNFT34D)....1086
TR-34 Key Receive (CSNDT34R and CSNFT34R)....1098

Chapter 12. Using digital signatures....1109
Signature algorithms and formatting methods....1109
Digital Signature Generate (CSNDDSG and CSNFDSG)....1110
Digital Signature Verify (CSNDDSV and CSNFDSV)....1119

Chapter 13. Managing PKA cryptographic keys....1131
PKA Key Generate (CSNDPKG and CSNFPKG)....1131
PKA Key Import (CSNDPKI and CSNFPKI)....1140
PKA Key Token Build (CSNDPKB and CSNFPKB)....1147
PKA Key Token Change (CSNDKTC and CSNFKTC)....1165
PKA Key Translate (CSNDPKT and CSNFPKT)....1169
PKA Public Key Extract (CSNDPKX and CSNFPKX)....1181
Public Infrastructure Certificate (CSNDPIC and CSNFPIC)....1184
Retained Key Delete (CSNDRKD and CSNFRKD)....1191
Retained Key List (CSNDRKL and CSNFRKL)....1194

Chapter 14. Key data set management....1199
Metadata for key data set records....1199
CKDS Key Record Create (CSNBKRC and CSNEKRC)....1202
CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2)....1203
CKDS Key Record Delete (CSNBKRD and CSNEKRD)....1205
CKDS Key Record Read (CSNBKRR and CSNEKRR)....1207
CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)....1208
CKDS Key Record Write (CSNBKRW and CSNEKRW)....1213
CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)....1215
Coordinated KDS Administration (CSFCRC and CSFCRC6)....1217
ICSF Multi-Purpose Service (CSFMPS and CSFMPS6)....1221
Key Data Set List (CSFKDSL and CSFKDSL6)....1225
Key Data Set Metadata Read (CSFKDMR and CSFKDMR6)....1239
Key Data Set Metadata Write (CSFKDMW and CSFKDMW6)....1246
Key Data Set Record Retrieve (CSFRRT and CSFRRT6)....1252
Key Data Set Update (CSFKDU and CSFKDU6)....1254
PKDS Key Record Create (CSNDKRC and CSNFKRC)....1257
PKDS Key Record Delete (CSNDKRD and CSNFKRD)....1259
PKDS Key Record Read and PKDS Key Record Read2 (CSNDKRR or CSNDKRR2 and CSNFKRR or CSNFKRR2)....1261
PKDS Key Record Write (CSNDKRW and CSNFKRW)....1263

Chapter 15. Utilities....1267
Character/Nibble Conversion (CSNBXBC and CSNBXCB)....1267
Code Conversion (CSNBXEA and CSNBXAE)....1269
Cryptographic Usage Statistic (CSFSTAT and CSFSTAT6)....1271
ICSF Query Algorithm (CSFIQA and CSFIQA6)....1273
ICSF Query Facility (CSFIQF and CSFIQF6)....1278
ICSF Query Facility2 (CSFIQF2 and CSFIQF26)....1316
SAF ACEE Selection (CSFACEE and CSFACEE6)....1321
X9.9 Data Editing (CSNB9ED)....1322

Chapter 16. Trusted interfaces....1325
Key Token Wrap (CSFWRP and CSFWRP6)....1325
PCI Interface (CSFPCI and CSFPCI6)....1327

Part 3. PKCS #11 callable services....1337

Chapter 17. Using PKCS #11 tokens and objects....1339
PKCS #11 Derive Multiple Keys (CSFPDMK and CSFPDMK6)....1339
PKCS #11 Derive Key (CSFPDVK and CSFPDVK6)....1347
PKCS #11 Get Attribute Value (CSFPGAV and CSFPGAV6)....1355
PKCS #11 Generate Key Pair (CSFPGKP and CSFPGKP6)....1357
PKCS #11 Generate Secret Key (CSFPGSK and CSFPGSK6)....1360
PKCS #11 Generate Keyed MAC (CSFPHMG and CSFPHMG6)....1364

PKCS #11 Verify Keyed MAC (CSFPHMV and CSFPHMV6)....1368
PKCS #11 One-Way Hash, Sign, or Verify (CSFPOWH and CSFPOWH6).... 1372
PKCS #11 Private Key Sign (CSFPPKS and CSFPPKS6)....1379
PKCS #11 Public Key Verify (CSFPPKV and CSFPPKV6).... 1382
PKCS #11 Pseudo-Random Function (CSFPPRF and CSFPPRF6).... 1386
PKCS #11 Set Attribute Value (CSFPSAV and CSFPSAV6).... 1389
PKCS #11 Secret Key Decrypt (CSFPSKD and CSFPSKD6).... 1391
PKCS #11 Secret Key Encrypt (CSFPSKE and CSFPSKE6).... 1396
PKCS #11 Secret Key Reencrypt (CSFPSKR and CSFPSKR6).... 1403
PKCS #11 Token Record Create (CSFPTRC and CSFPTRC6).... 1407
PKCS #11 Token Record Delete (CSFPTRD and CSFPTRD6).... 1411
PKCS #11 Token Record List (CSFPTRL and CSFPTRL6).... 1413
PKCS #11 Unwrap Key (CSFPUWK and CSFPUWK6).... 1417
PKCS #11 Wrap Key (CSFPWPK and CSFPWPK6).... 1422

Chapter 18. Using the PKCS #11 key structure callable services.... 1429
PKCS #11 Private Key Structure Decrypt (CSFPPD2 and CSFPPD26).... 1429
PKCS #11 Private Key Structure Sign (CSFPPS2 and CSFPPS26).... 1432
PKCS #11 Public Key Structure Encrypt (CSFPPE2 and CSFPPE26).... 1434
PKCS #11 Public Key Structure Verify (CSFPPV2 and CSFPPV26).... 1437

Appendix A. ICSF and cryptographic coprocessor return/reason codes.... 1441
Return codes and reason codes.... 1441
Obtaining a dump for ICSF reason codes.... 1441
Return codes.... 1441
Reason codes for return code 0 (0).... 1442
Reason codes for return code 4 (4).... 1444
Reason codes for return code 8 (8).... 1447
Reason codes for return code C (12).... 1489
Reason codes for return code 10 (16).... 1499

Appendix B. Key token formats.... 1501
Master key verification pattern (MKVP).... 1501
Null key tokens.... 1501
Symmetric key tokens.... 1502
Token validation value (fixed-length symmetric tokens).... 1502
AES internal fixed-length key token.... 1502
DES fixed-length key token.... 1503
External RKX DES key token.... 1508
Variable-length symmetric key token formats.... 1509
Variable-length symmetric key token.... 1509
Variable-length symmetric null key token.... 1537
X9.143 (TR-31) key block header and optional block data.... 1538
PKA key tokens.... 1546
PKA key token sections.... 1547
Integrity of PKA private key sections containing an encrypted RSA key.... 1548
Number representation in PKA key tokens.... 1549
AESKW external format.... 1584
Trusted blocks.... 1588

Appendix C. Changing control vectors with the CVT callable service.... 1603
DES control vector table.... 1603
Control-Vector Base Bits.... 1605
Key form bits, 'fff'.... 1608
Specifying a control-vector-base value.... 1608
Changing control vectors with the Control Vector Translate callable service.... 1613
Providing the control information for testing the control vectors.... 1613

Mask array preparation....1613
Selecting the key-half processing mode....1615
When the target key token CV is null....1617
Control Vector Translate example....1617

Appendix D. Coding examples....1619
C....1619
COBOL....1621
High Level Assembler....1623
PL/I....1625
Rexx....1627

Appendix E. Cryptographic algorithms and processes....1631
PIN formats and algorithms....1631
PIN Notation....1631
PIN block formats....1632
PIN extraction rules....1633
IBM PIN algorithms....1635
VISA PIN algorithms....1641
Cipher processing rules....1643
CBC and ANSI INCITS 106....1643
ANSI X9.23 and IBM 4700....1644
CUSP....1644
The Information Protection System (IPS)....1645
PKCS padding method....1646
Wrapping methods for symmetric key tokens....1647
ECB wrapping of DES keys in a fixed-length token (WRAP-ECB)....1647
CBC wrapping of AES keys in fixed-length tokens....1648
Enhanced CBC wrapping of DES keys in fixed-length tokens (WRAP-ENH, WRAPENH2, WRAPENH3)....1648
Variable length token (AESKW method)....1648
PKA92 key format and encryption process....1649
Formatting hashes and keys in public-key cryptography....1650
ANSI X9.31 hash format....1650
PKCS #1 formats....1651
Visa, MasterCard, and EMV-related smart card formats and processes....1652
Deriving the smart-card-specific authentication code....1652
Constructing the PIN-block for transporting an EMV smart-card PIN....1652
Deriving the CCA TDES-XOR session key....1653
Deriving the EMV TDESEMVn tree-based session key....1653
PIN-block self-encryption....1654
Key test verification pattern algorithms....1654
DES algorithm (single-length and double-length keys)....1654
SHAVP1 algorithm....1654
SHA-256 algorithm....1654

Appendix F. EBCDIC and ASCII default conversion tables....1657

Appendix G. Access control points and callable services....1661

Appendix H. Impact of compliance mode on callable services....1695

Appendix I. Resource names for CCA and ICSF entry points....1705

Appendix J. Cryptographic hardware engines and software....1715
IBM Common Cryptographic Architecture (CCA)....1715

Hardware support....1716

Appendix K. AES-DUKPT reference information....1719
AES-DUKPT derivation data....1719
Supported CCA key types for AES-DUKPT derived working keys....1722
AES-DUKPT allowed derived working key sizes....1722

Appendix L. CCA release levels....1723

Appendix M. Accessibility....1727

Notices....1729
Terms and conditions for product documentation....1730
IBM Online Privacy Statement....1731
Policy for unsupported hardware....1731
Minimum supported hardware....1731
Trademarks....1732
Glossary....1733

Index....1747

Figures

1. Overview of trusted block contents....46
2. Simplified RKX key-token structure....50
3. Trusted block creation....50
4. Exporting keys using a trusted block....51
5. Generating keys using a trusted block....53
6. Typical flow of callable services for remote key export....55
7. PKA Key Management....102
8. Keyword combinations for DES CIPHER keys....281
9. Keyword combinations for DES DECIPHER and ENCIPHER keys....282
10. CSNBCVG and CSNBKTB keyword combinations for DES CIPHERXI, CIPHERXL, and CIPHERXO keys....283
11. CSNBCVG and CSNBKTB keyword combinations for DES DATA keys....284
12. CSNBCVG and CSNBKTB keyword combinations for DES DATAC, DATAM, and DATAMV keys....285
13. CSNBCVG and CSNBKTB keyword combinations for DES MAC and MACVER keys....286
14. CSNBCVG and CSNBKTB keyword combinations for DES SECMSG keys....287
15. CSNBCVG and CSNBKTB keyword combinations for DES IPINENC keys....288
16. CSNBCVG and CSNBKTB keyword combinations for DES OPINENC keys....289
17. CSNBCVG and CSNBKTB keyword combinations for DES PINGEN keys....290
18. CSNBCVG and CSNBKTB keyword combinations for DES PINVER keys....291
19. CSNBCVG and CSNBKTB keyword combinations for DES EXPORTER keys....292
20. CSNBCVG and CSNBKTB, keyword combinations for DES IMPORTER keys....293
21. CSNBCVG and CSNBKTB keyword combinations for DES IKEYXLAT and OKEYXLAT keys....294
22. CSNBCVG and CSNBKTB keyword combinations for DES DKYGENKY keys....295

23. CSNBCVG and CSNBKTB keyword combinations for DES KEYGENKY keys....296

24. CSNBCVG and CSNBKTB keyword combinations for DES CVARDEC, CVARENC, CVARPINE, CVARXCVL, and CVARXCVR keys....297

25. Key Token Build2 keyword combinations for AES CIPHER keys....303

26. Key Token Build2 keyword combinations for AES MAC keys....306

27. Key_Token_Build2 keyword combinations for HMAC MAC keys....309

28. Key Token Build2 keyword combinations for AES EXPORTER keys....312

29. Key Token Build2 keyword combinations for AES IMPORTER keys....315

30. Key Token Build2 keyword combinations for AES DKYGENKY keys....319

31. Key Token Build2 keyword combinations for AES PINCALC keys....324

32. Key Token Build2 keyword combinations for AES PINPROT keys....326

33. Key Token Build2 keyword combinations for AES KDKGENKY keys....329

34. Key Token Build2 keyword combinations for AES PINPRW keys....334

35. Key Token Build2 keyword combinations for AES SECMSG keys....336

36. Control Vector Translate Callable Service Mask_Array Processing....1615

37. Control Vector Translate Callable Service....1616

38. 3624 PIN Generation Algorithm....1636

39. GBP PIN Generation Algorithm....1637

40. PIN-Offset Generation Algorithm....1638

41. PIN Verification Algorithm....1640

42. GBP PIN Verification Algorithm....1641

43. PVV Generation Algorithm....1642

Tables

1. ICSF Callable Services Naming Conventions....3
2. Standard Return Code Values From ICSF Callable Services....7
3. Key label....8
4. Deprecated callable services....10
5. Descriptions of DES key types and service usage....24
6. Descriptions of AES key types and service usage....26
7. Descriptions of HMAC key types and service usage....27
8. Descriptions of Clear key types and service usage....28
9. CCA callable services and parameters that support operational X9.143 key blocks....29
10. AES EXPORTER strength required for exporting an HMAC key under an AES EXPORTER....34
11. Minimum RSA modulus length to adequately protect an AES key....34
12. Combinations of the callable services....73
13. Summary of ICSF callable services....77
14. AES EXPORTER strength required for exporting an HMAC key under an AES EXPORTER....94
15. Minimum RSA modulus length to adequately protect an AES key....94
16. Summary of PKA key token sections....101
17. Key label format....103
18. Summary of PKA callable services....104
19. Summary of PKCS #11 callable services....109
20. Summary of PKCS #11 callable services that offer a fast-path alternative....111
21. Clear Key Import required hardware....118
22. Rule array keywords for Control Vector Generate....120
23. Keywords for Control Vector Translate....126

24. Control Vector Translate required hardware....127

25. Cryptographic Variable Encipher required hardware....130

26. Required access control points for Data Key Export....132

27. Data Key Export required hardware....133

28. Required access control points for Data Key Import....135

29. Data Key Import required hardware....136

30. Rule array keywords for Derive ICC MK....138

31. Derive ICC MK: Key requirements....140

32. Derive ICC MK: Key type and key usage attributes of the generated keys....140

33. Derive ICC MK required hardware....143

34. Rule array keywords for Derive Session Key....145

35. Derive Session Key: Key requirements....147

36. Derive Session Key: Attributes of the key generated....148

37. Derive Session Key required hardware....150

38. Rule Array Keywords for Diversified Key Generate....153

39. Key identifier requirements....155

40. Input requirements for the key identifier....158

41. Required access control points for Diversified Key Generate....160

42. Diversified Key Generate required hardware....161

43. Rule array keywords for Diversified Key Generate2....164

44. Summary of input generating key tokens, input generated key tokens, and output generated key tokens....167

45. Required access control points for Diversified Key Generate2....168

46. Diversified Key Generate2 required hardware....169

47. Keywords for Diversify Directed Key....171

48. Summary of KTV tables....176

49. KTV for MAC generate/verify, Type A active and Type B passive..... 177

50. KTV for MAC generate/verify, Type B active and Type A passive..... 177

51. KTV for data encryption (cipher), Type A active and Type B passive..... 177

52. KTV for data encryption (cipher), Type B active and Type A passive..... 177

53. KTV for PIN encryption, Type A active and Type B passive..... 178

54. KTV for PIN encryption, Type B active and Type A passive..... 178

55. KTV for PIN encryption, Type A active and Type B passive..... 178

56. KTV for PIN encryption, Type B active and Type A passive..... 178

57. KTV for key wrapping, Type A active and Type B passive..... 179

58. KTV for key wrapping, Type B active and Type A passive..... 179

59. Diversify Directed Key required hardware..... 181

60. Keywords for ECC Diffie-Hellman..... 184

61. Valid key bit lengths and minimum curve size required for the supported output key types..... 190

62. CSNDEDH concatenation string format for DERIV01..... 191

63. CSNDEDH concatenation string format for DERIV02..... 192

64. ECC Diffie-Hellman required hardware..... 195

65. Rule array keywords for Generate Issuer MK..... 199

66. Generate Issuer MK: Attributes of the generated key..... 201

67. Generate Issuer MK required hardware..... 203

68. Keywords for Key Encryption Translate..... 205

69. Required access control points for Key Encryption Translate 207

70. Key Encryption Translate required hardware..... 207

71. Required access control points for Key Export..... 210

72. Key export required hardware..... 211

73. Key Form values for the Key Generate callable service..... 213

74. Key Length values for the Key Generate callable service..... 214

75. Key lengths for DES keys..... 215

76. Key lengths for AES keys..... 216

77. Key attributes for key-encrypting keys for CSNBKGN..... 217

78. Key Generate Valid Key Types and Key Forms for a Single Key.....220

79. Key Generate Valid Key Types and Key Forms for a Key Pair..... 221

80. Required access control points for Key Generate.....222

81. Key generate required hardware..... 223

82. Keywords for Key Generate2 Control Information..... 226

83. Keywords and associated algorithms for key_type_1 parameter..... 228

84. Keywords and associated algorithms for key_type_2 parameter..... 228

85. Key Generate2 valid key type and key form for one AES or HMAC key..... 232

86. Key Generate2 Valid key type and key forms for two AES or HMAC keys..... 233

87. Valid key pairs that can be generated and their required access points.....234

88. Key type and key form keywords for AES keys - DK PIN methods.....235

89. AES KEK strength required for generating an HMAC key under an AES KEK..... 236

90. Required access control points for Key Generate2..... 236

91. Key Generate2 required hardware..... 237

92. Required access control points for Key Import 241

93. Key import required hardware.....241

94. Keywords for Key Part Import Control Information.....243

95. Required access control points for Key Part Import.....246

96. Key Part Import required hardware..... 246

97. Keywords for Key Part Import2 Control Information..... 249

98. Required access control points for Key Part Import2.....251

99. Key Part Import2 required hardware..... 251

100. Keywords for Key Test Control Information.....253

101. Key Test required hardware..... 255

102. Keywords for Key Test2 Control Information.....258

103. AES keys and input key description..... 260

104. DES keys and input key description..... 260

105. HMAC keys and input key description.....262

106. Length of the verification pattern for each algorithm or process rule supported.....263

107. Returned data for KEY-LEN process rule..... 264

108. Required access control points for Key Test2.....264

109. Key Test2 required hardware..... 265

110. Keywords for Key Test Extended Control Information..... 269

111. Key Test Extended required hardware.....271

112. Valid key_type keywords for AES..... 273

113. Valid key_type keywords for DES..... 273

114. Keywords for Key Token Build Control Information..... 275

115. Key types and field lengths for AES keys..... 278

116. Control Vector Generate and Key Token Build keyword combinations by DES key types.....280

117. Keywords for Key Token Build2 Control Information..... 299

118. Rule array keywords for AES CIPHER keys.....303

119. Rule array keywords for AES MAC keys..... 306

120. Rule array keywords for HMAC MAC keys.....309

121. Rule array keywords for AES EXPORTER keys.....313

122. Rule array keywords for AES IMPORTER keys.....316

123. Rule array keywords for AES DKYGENKY keys..... 320

124. Meaning of service_data parameter when DKYUSAGE specified....322
125. Rule array keywords for AES PINCALC keys....324
126. Allowable keyword combinations for PINPROT keys....327
127. Rule array keywords for AES PINPROT keys....327
128. Rule array keywords for AES KDKGENKY keys....330
129. Allowable keywords for AES KDKGENKY keys....333
130. Rule array keywords for AES PINPRW keys....334
131. Rule array keywords for AES SECMSG keys....337
132. AES DKYGENKY and AES KDKGENKY active/passive related key-usage field block....340
133. AES KDKGENKY key usage fields format....342
134. AES DKYGENKY key usage fields format....343
135. Key Translate required hardware....345
136. Key Translate2 Access Control Points....352
137. Key Translate2 required hardware....353
138. Keywords for Multiple Clear Key Import Rule Array Control Information....356
139. Required access control points for Multiple Clear Key Import....358
140. Multiple Clear Key Import required hardware....358
141. Keywords for Multiple Secure Key Import Rule Array Control Information....360
142. Required access control points for Multiple Secure Key Import....363
143. Multiple Secure Key Import required hardware....364
144. Keywords for PKA Decrypt....367
145. PKA Decrypt access controls....370
146. PKA Decrypt required hardware....371
147. Keywords for PKA Encrypt....375
148. PKA Encrypt access controls....380

149. PKA Encrypt required hardware....381
150. Prohibit Export required hardware....386
151. Prohibit Export Extended required hardware....388
152. Keywords for the Form Parameter....390
153. Keywords for Random Number Generate Control Information....390
154. Random Number Generate required hardware....392
155. rule_array keywords....395
156. Structure of values used by RKX....396
157. Transport_key_identifer used by RKX....397
158. Examination of key token for source_key_identifier....399
159. Remote Key Export required hardware....402
160. Keywords for Restrict Key Attribute Control Information....404
161. Required access control points for Restrict Key Attribute....407
162. Restrict Key Attribute required hardware....407
163. Required access control points for Secure Key Import....411
164. Secure Key Import required hardware....411
165. Keywords for Secure Key Import2 Control Information....413
166. Required access control points for Secure Key Import2....416
167. Secure Key Import2 required hardware....417
168. CSNDSYX key formatting for fixed length AES and DES key tokens....418
169. CSNDSYX key formatting for variable length AES and HMAC key tokens....418
170. Keywords for Symmetric Key Export Control Information....420
171. Minimum RSA modulus strength required to contain a PKOAEP2 block when exporting an AES key....423
172. Required access control points for Symmetric Key Export....424
173. Symmetric Key Export required hardware....424

174. Keywords for Symmetric Key Export with Data (CSNDSXD)..... 427
175. Required access control points for Symmetric Key Export with Data.....429
176. Required access control points based on the key-formatting method and the token algorithm.....429
177. Symmetric Key Export with Data required hardware..... 429
178. Keywords for Symmetric Key Generate Control Information.....431
179. requirements for the key identifier..... 435
180. Required access control points for Symmetric Key Generate.....437
181. Symmetric Key Generate required hardware..... 438
182. Keywords for Symmetric Key Import Control Information..... 440
183. Required access control points for Symmetric Key Import.....443
184. Symmetric Key Import required hardware..... 444
185. Keywords for Symmetric Key Import2 Control Information..... 446
186. PKCS#1 OAEP encoded message layout (PKOAEP2).....448
187. Symmetric Key Import2 Access Control Points.....449
188. Symmetric Key Import2 required hardware..... 449
189. Rule_array keywords for Trusted Block Create (CSNDTBC)..... 452
190. Required access control points for Trusted Block Create..... 453
191. Trusted Block Create required hardware..... 453
192. Keywords for Unique Key Derive..... 456
193. Contents of the TR-31 block header of the generated TR-31 key block and their meaning.....462
194. Valid Control Vectors for Derived Keys..... 464
195. DES-DUKPT key variants for derived keys..... 465
196. Unique Key Derive required hardware..... 466
197. Keywords for Cipher Text Translate2..... 473
198. Restrictions for ciphertext_in_length and ciphertext_out_length.....479

199. Cipher Text Translate2 key usage 482
200. Cipher Text Translate2 access control points 482
201. Cipher Text Translate2 required hardware 483
202. Keywords for the Decipher Rule Array Control Information 487
203. Decipher required hardware 490
204. Decode required hardware 492
205. Keywords for the Encipher Rule Array Control Information 496
206. Encipher required hardware 499
207. Encode required hardware 501
208. Symmetric Algorithm Decipher Rule Array Keywords 503
209. Symmetric Algorithm Decipher required hardware 508
210. Symmetric Algorithm Encipher Rule Array Keywords 512
211. Access controls for Symmetric Algorithm Encipher 519
212. Symmetric Algorithm Encipher required hardware 519
213. Symmetric Key Decipher Rule Array Keywords 524
214. Required access control points for Symmetric Key Decipher 529
215. Symmetric Key Decipher required hardware 530
216. Symmetric Key Encipher Rule Array Keywords 535
217. Required access control points for Symmetric Key Encipher 540
218. Symmetric Key Encipher required hardware 541
219. Keywords for HMAC Generate Control Information 546
220. Minimum HMAC key size in bits based on hash method 548
221. HMAC Generate Access Control Points 549
222. HMAC Generate required hardware 549
223. Keywords for HMAC Verify Control Information 552

224. HMAC Verify Access Control Points....554
225. HMAC Verify required hardware....555
226. Keywords for MAC Generate control information....558
227. MAC Generate required hardware....561
228. Keywords for MAC Generate2 control information....564
229. MAC Generate2 Access Control Points....566
230. MAC Generate2 required hardware....567
231. Keywords for MAC Verify control information....571
232. MAC Verify required hardware....573
233. Keywords for MAC Verify2 control information....576
234. MAC Verify2 Access Control Points....578
235. MAC Verify2 required hardware....578
236. Keywords for MDC Generate control information....582
237. MDC Generate required hardware....583
238. Blocksize and hash length for hash methods....584
239. Keywords for One-Way Hash Generate Rule Array Control Information....586
240. One-Way Hash Generate required hardware....589
241. Keywords for Symmetric MAC Generate control information....592
242. Symmetric MAC Generate required hardware....594
243. Keywords for Symmetric MAC Verify control information....597
244. Symmetric MAC Verify required hardware....599
245. Valid translation rules....604
246. ANSI X9.8 PIN - Allow only ANSI PIN blocks....607
247. Callable services affected by enhanced PIN security mode....608
248. Format of a PIN profile....609

249. Format values of PIN blocks..... 610
250. PIN block format and PIN extraction method keywords.....610
251. Format of a pad digit.....612
252. Pad digits for PIN block formats..... 613
253. Format of the current key serial number field..... 613
254. Base-10 alphabet..... 614
255. FPE base-15 alphabet..... 615
256. FPE track 1 cardholder name alphabet.....616
257. FPE track 1 discretionary data alphabet.....618
258. VFPE track 2 discretionary data alphabet.....619
259. Authentication Parameter Generate Rule Array Keywords.....621
260. Access Control Points for Authentication Parameter Generate (CSNBAPG and CSNEAPG).....623
261. Authentication Parameter Generate required hardware.....623
262. Process Rules for the Clear PIN Encryption Callable Service..... 626
263. Clear PIN Encrypt required hardware..... 628
264. Process Rules for the Clear PIN Generate Callable Service.....631
265. Array Elements for the Clear PIN Generate Callable Service.....632
266. Array Elements Required by the Process Rule..... 632
267. Required access control points for Clear PIN Generate.....633
268. Clear PIN Generate required hardware..... 633
269. PAN data structure..... 636
270. Rule Array Elements for the Clear PIN Generate Alternate Service..... 637
271. Rule Array Keywords (First Element) for the Clear PIN Generate Alternate Service..... 637
272. Data Array Elements for the Clear PIN Generate Alternate Service (IBM-PINO).....638
273. Data Array Elements for the Clear PIN Generate Alternate Service (VISA-PVV)..... 638

274. Required access control points for Clear PIN Generate Alternate....639
275. Clear PIN Generate Alternate required hardware....639
276. Keywords for CVV Key Combine Rule Array Control Information....642
277. Key type combinations for the CVV Key Combine callable service....644
278. WRAPENH3 Key-wrapping matrix for CVV_Key_Combine....644
279. Wrapping combinations for the CVV Combine Callable Service....644
280. CVV Key Combine required hardware....645
281. Rule array keywords for EMV Scripting Service....648
282. EMV Scripting Service: Key requirements....650
283. Key type requirements for actions SMCON and SMCONINT....650
284. Key type requirements for actions SMCONPIN, SMCIPIN, and VISAPIN....651
285. EMV Scripting Service required hardware....657
286. Rule array keywords for EMV Transaction (ARQC/ARPC) Service....659
287. EMV Transaction (ARQC/ARPC) Service required hardware....665
288. Rule array keywords for EMV Verification Functions....667
289. EMV Verification Functions: Key requirements....668
290. EMV Verification Functions required hardware....671
291. Process Rules for the Encrypted PIN Generate Callable Service....674
292. Array Elements for the Encrypted PIN Generate Callable Service....675
293. Array Elements Required by the Process Rule....675
294. PAN data structure....676
295. Required access control points for Encrypted PIN Generate....676
296. Encrypted PIN Generate required hardware....677
297. Keywords for Encrypted PIN Translate....681
298. Additional Names for PIN Formats....683

299. Required access control points for Encrypted PIN Translate..... 683
300. Encrypted PIN Translate required hardware..... 684
301. Keywords for Encrypted PIN Translate2..... 687
302. Supported Encrypted PIN Translate2 PIN profile lengths..... 692
303. Key usage requirements for PIN encrypting keys..... 696
304. Valid encrypted PIN Translate2 DUKPT keyword combinations..... 697
305. Required access control points for Encrypted PIN Translate2..... 698
306. Required access controls for ISO-4 PIN blocks..... 698
307. Encrypted PIN Translate2 required hardware..... 700
308. VMDS pairings for enciphered PAN data..... 702
309. Rule array keywords for Encrypted PIN Translate Enhanced..... 704
310. Supported Encrypted PIN Translate Enhanced PIN profile lengths..... 708
311. Valid encrypted PIN Translate Enhanced DUKPT keyword combinations..... 711
312. Encrypted PIN Translate Enhanced required hardware..... 712
313. PAN data structure..... 716
314. Keywords for Encrypted PIN Verify..... 716
315. Array Elements for the Encrypted PIN Verify Callable Service..... 717
316. Array Elements Required by the Process Rule..... 718
317. Required access control points for Encrypted PIN Verify..... 718
318. Encrypted PIN Verify required hardware..... 719
319. Keywords for Encrypted PIN Verify2..... 722
320. Keywords for Encrypted PIN Verify2..... 723
321. Supported Encrypted PIN Verify2 input PIN profile lengths..... 725
322. Supported Encrypted PIN Verify2 reference PIN profile lengths..... 726
323. Required access control points for Encrypted PIN Verify2..... 729

324. Encrypted PIN Verify2 required hardware....729

325. Rule array keywords for Field Level Decipher....731

326. Access control points for Field Level Decipher....737

327. Field Level Decipher required hardware....737

328. Rule array keywords for Field Level Encipher....741

329. Access control points for Field Level Encipher....747

330. Field Level Encipher required hardware....747

331. Rule array keywords for Format Preserving Algorithms Decipher control information....750

332. Access controls in the domain role that control the function of the Format Preserving Algorithms Decipher service....753

333. Format Preserving Algorithms Decipher required hardware....754

334. Rule array keywords for Format Preserving Algorithms Encipher control information....756

335. Access controls in the domain role that control the function of the Format Preserving Algorithms Encipher service....759

336. Format Preserving Algorithms Encipher required hardware....759

337. Rule array keywords for Format Preserving Algorithms Translate control information....761

338. Access controls in the domain role that control the function of the Format Preserving Algorithms Translate service....767

339. Format Preserving Algorithms Translate required hardware....767

340. Rule array keywords for FPE Decipher....770

341. FPE Decipher required hardware....776

342. Rule array keywords for FPE Encipher....779

343. FPE Encipher required hardware....785

344. Rule array keywords for FPE Translate....788

345. FPE Translate required hardware....795

346. Rule Array Keywords for PIN Change/Unblock....798

347. PAN data structure....802

348. PAN data structure..... 803

349. Required access control points for PIN Change/Unblock..... 804

350. PIN Change/Unblock hardware.....805

351. PAN data structure..... 809

352. Recover PIN from Offset required hardware..... 810

353. Rule Array Keywords for Secure Messaging for Keys..... 813

354. Secure Messaging for Keys required hardware..... 815

355. Rule Array Keywords for Secure Messaging for PINs.....817

356. PAN data structure..... 819

357. Secure Messaging for PINs required hardware..... 822

358. Keywords for SET Block Compose Control Information.....824

359. SET Block Compose required hardware..... 828

360. Keywords for SET Block Compose Control Information.....830

361. Required access control points for PIN-block encrypting key.....833

362. SET Block Decompose required hardware..... 833

363. Rule Array Keywords for Transaction Validation.....835

364. Output description for validation values.....837

365. Required access control points for Transaction Validation.....837

366. Transaction Validation required hardware.....838

367. CVV Generate Rule Array Keywords.....840

368. VISA CVV Service Generate required hardware..... 842

369. CVV Verify Rule Array Keywords..... 844

370. VISA CVV Service Verify required hardware..... 846

371. Rule array keywords for the DK Deterministic PIN Generate service.....851

372. DK Deterministic PIN Generate required hardware..... 856

373. Rule array keywords for the DK Migrate PIN service....858
374. DK Migrate PIN required hardware....863
375. Keywords for the DK PIN Verify Service....865
376. DK PAN Modify in Transaction required hardware....870
377. DK PAN Translate required hardware....877
378. Rule array keywords for the DK PIN Change Service....880
379. DK PIN Change required hardware....891
380. Keywords for the DK PIN Verify Service....893
381. DK PIN Verify required hardware....896
382. Keywords for the DK PRW Card Number Update service....899
383. DK PRW Card Number Update required hardware....903
384. Keywords for the DK PRW Card Number Update2 service....906
385. DK PRW Card Number Update2 required hardware....911
386. DK PRW CMAC Generate required hardware....915
387. Rule array keywords for DK Random PIN Generate with Reference Value Service....918
388. DK Random PIN Generate required hardware....922
389. Keywords for DK Random PIN Generate2....924
390. DK Random PIN Generate2 required hardware....929
391. DK Regenerate PRW required hardware....935
392. Keywords for TR-31 Create control information....940
393. Access controls specific to TR-31 Create and DK-specific access controls used in TR-31 Create... 954
394. Key usage keywords for a single key....955
395. Key usage and key form keywords....956
396. TR-31 Create required hardware....958
397. Keywords for TR-31 Import Rule Array Control Information....960

398. Import translation table for a TR-31 BDK base derivation key (usage "B0")..... 965

399. Import translation table for a TR-31 CVK card verification key (usage "C0").....966

400. Import translation table for a TR-31 data encryption key (usage "D0").....967

401. Import translation table for a TR-31 data encryption key (usage "F0")..... 969

402. Import translation table for a TR-31 key encryption or wrapping, or key block protection key (usages "K0", "K1")..... 972

403. Import translation table for a TR-31 ISO MAC algorithm key (usages "M0", "M1", "M3")..... 974

404. Import translation table for a TR-31 HMAC algorithm key (usages "M7")..... 976

405. Import translation table for a TR-31 PIN encryption or PIN verification key (usages "P0", "V0", "V1", "V2")..... 976

406. Import translation table for a initialization vector (usage "I0")..... 980

407. Import translation table for a TR-31 EMV/chip issuer master-key key (usages "E0", "E1", "E2", "E3", "E4", "E5")..... 980

408. Export attributes of an imported CCA token 988

409. TR-31 to CCA Import required access controls.....989

410. TR-31 Import required hardware.....995

411. Keywords for TR-31 Optional Data Read Rule Array Control Information..... 1002

412. Keywords for TR-31 Translate Rule Array Control Information.....1009

413. Keywords for TR-31 Translate Rule Array Control Information.....1011

414. Export translation table for a initialization vector..... 1020

415. Export translation table for a TR-31 BDK base derivation key..... 1020

416. Export translation table for a TR-31 CVK card verification key (CVK)..... 1022

417. Export translation table for a TR-31 data encryption key (ENC).....1024

418. Export translation table for a TR-31 key encryption or wrapping, or key block protection key (KEK or KEK-WRAP)..... 1025

419. Export translation table for a TR-31 ISO MAC algorithm key (ISOMACn)..... 1027

420. Export translation table for a TR-31 EMV/chip card key (DKYGENKY, DATA).....1032

421. Export translation table for a TR-31 HMAC algorithm key (MAC)..... 1034

422. Export translation table for a TR-31 PIN encryption or PIN verification key (PINENC, PINVO, PINV3624, VISAPVV)..... 1035

423. Export translation table for a TR-31 EMV/chip issuer master-key key (DKYGENKY, DATA).....1040

424. Export translation table for a TR-31 key with proprietary DK key usage.....1042

425. Export translation table for an AES TR-31 key..... 1043

426. Export translation table for CCA key types..... 1044

427. Recommended values for the key_field_length parameter..... 1046

428. Valid CCA to TR-31 Translate Translations and Required Access Controls.....1050

429. TR-31 Translate required hardware.....1053

430. Keywords for TR-34 Bind-Begin..... 1074

431. Access control points for TR-34 Bind-Begin..... 1078

432. TR-34 Bind-Begin required hardware.....1079

433. Keywords for TR-34 Bind-Complete.....1081

434. Access control points for TR-34 Bind-Complete.....1085

435. TR-34 Bind-Complete required hardware..... 1085

436. Keywords for TR-34 Key Distribution.....1088

437. TR-31 key usage value for output key block.....1090

438. TR-31 mode of key use..... 1090

439. TR-31 exportability 1091

440. Export translation table for DES keys in TR-34 key blocks..... 1091

441. Export translation table for AES keys in TR-34 key blocks..... 1091

442. Access control points for TR-34 Key Distribution.....1096

443. Valid CCA to TR-34 Export Translations and Required Access Controls..... 1097

444. TR-34 Key Distribution required hardware.....1097

445. Keywords for TR-34 Key Receive.....1100

446. Input translation table for DES key usage..... 1102

447. Input translation table for AES key usage..... 1102
448. Access control points for TR-34 Key Receive..... 1105
449. Valid TR-34 to CCA import translations and required access controls..... 1106
450. TR-34 Key Receive required hardware..... 1106
451. Keywords for Digital Signature Generate Control Information..... 1111
452. Digital Signature Generate required hardware..... 1117
453. Keywords for Digital Signature Verify Control Information..... 1121
454. Digital Signature Verify required hardware..... 1127
455. CSNDPKB keywords and the required master key..... 1131
456. Keywords for PKA Key Generate Rule Array..... 1134
457. Required access control points for PKA Key Generate rule array keys..... 1137
458. PKA Key Generate required hardware..... 1138
459. Keywords for PKA Key Import..... 1142
460. PKA Key Import required hardware..... 1145
461. Keywords for PKA Key Token Build Control Information..... 1150
462. Key Value Structure Length Maximum Values for Key Types..... 1153
463. Key Value Structure Elements for PKA Key Token Build..... 1154
464. Clear key format X'01' CRYSTALS-Dilithium key object layout with sizes..... 1161
465. Clear Key format X'01' CRYSTALS-Kyber key object layout with sizes..... 1161
466. Clear key format X'02' CRYSTALS-Dilithium key object layout with sizes..... 1161
467. Clear Key format X'02' CRYSTALS-Kyber key object layout with sizes..... 1162
468. Clear key format X'03' CRYSTALS-Dilithium key object layout with sizes..... 1162
469. Clear Key format X'03' CRYSTALS-Kyber key object layout with sizes..... 1162
470. PKA Key Token Build key-derivation-data contents, ECC keys..... 1163
471. Rule Array Keywords for PKA Key Token Change..... 1167

472. PKA Key Token Change required hardware 1168
473. Keywords for PKA Key Translate Rule Array 1171
474. Required access control points for PKA Key Translate 1177
475. Required access control points for source/target transport key combinations 1178
476. PKA Key Translate required hardware 1179
477. Keywords for Public Infrastructure Certificate 1185
478. Required access control points for Public Infrastructure Certificate 1190
479. Public Infrastructure Certificate required hardware 1191
480. Retained Key Delete required hardware 1193
481. Retained Key List required hardware 1196
482. Format of the metadata block 1200
483. Key fingerprint metadata format 1201
484. Keywords for CKDS Key Record Read2 1209
485. Required access control points for CKDS Key Record Read2 1211
486. CKDS Key Record Read2 required hardware 1212
487. Coordinated KDS Administration required hardware 1220
488. Keywords for ICSF Multi-Purpose Service 1222
489. ICSF Multi-Purpose Service required hardware 1224
490. Keywords for KDS list control information 1226
491. Search criteria entry 1228
492. Search criteria with date tag 1229
493. Search criteria with metadata tag 1230
494. Search criteria with TKDS object type 1231
495. Search criteria with CKDS key type 1231
496. Search criteria with a metadata flag 1233

497. Search criteria with an unsupported CCA key..... 1234
498. Search criteria with a weak CCA key..... 1235
499. Output area data when DETAILED is specified..... 1236
500. Output area data when REPORT-A is specified..... 1236
501. Keywords for KDS metadata read control information..... 1241
502. Metadata entry..... 1242
503. Output structure for variable metadata block..... 1243
504. Output structure for record create and update dates..... 1244
505. Output structure for key material validity, archive, recall, and last reference dates..... 1244
506. Output structure for flags..... 1245
507. Keywords for KDS metadata write control information..... 1247
508. Metadata entry..... 1249
509. Structure for variable metadata block..... 1249
510. Structure for key material validity and last reference date..... 1250
511. Structure for flag..... 1250
512. Keywords for PKDS Key Record Delete..... 1260
513. Keywords for PKDS Key Record Read2 control information..... 1263
514. Keywords for PKDS Key Record Write..... 1264
515. Keywords for Cryptographic Usage update..... 1272
516. Keywords for ICSF Query Algorithm..... 1274
517. Output for ICSF Query Algorithm..... 1276
518. Keywords for ICSF Query Facility..... 1279
519. Output for option GETCOMPD..... 1281
520. Output for option ICSFOPTN..... 1283
521. Output for option ICSFSP11..... 1291

522. Output for option ICSFSTAT 1291
523. Output for option ICSFSTAT (con't) 1295
524. Output for option ICSFST2 1295
525. Output for option MKCVCMAC 1303
526. Output for option MKCVENCZ 1303
527. Output for option STATAES 1304
528. Output for option STATAPKA 1304
529. Output for option STATCARD 1305
530. Output for option STATCCA 1306
531. Output for option STATCCAE 1308
532. Output for option STATDECT 1309
533. Output for option STATDIAG 1310
534. Output for option STATEID 1312
535. Output for option STATEXPT 1312
536. Output for option STATP11 1313
537. Output for option STATWPIN 1314
538. Output for option WRAPMTHD 1315
539. ICSF Query Facility required hardware 1316
540. Format of returned ICSF Query Facility 2 data 1317
541. Key Token Wrap access control points 1327
542. Key Token Wrap required hardware 1327
543. Keywords for PCI interface callable service 1329
544. ACP group header format 1332
545. ACP description format 1332
546. Current cryptographic coprocessor configuration 1332

547. PCI Interface required hardware..... 1334
548. Keywords for derive multiple keys..... 1341
549. parms_list parameter format for SSL-KM and TLS-KM mechanisms..... 1343
550. parms_list parameter format for IKE1PHA1 mechanism..... 1344
551. parms_list parameter format for IKE2PHA1 mechanism..... 1344
552. parms_list parameter format for IKE1PHA2 and IKE2PHA2 mechanisms..... 1345
553. Keywords for derive key..... 1348
554. parms_list parameter format for PKCS-DH mechanism..... 1350
555. parms_list parameter format for SSL-MS, SSL-MSDH, TLS-MS, and TLS-MSDH mechanisms..... 1350
556. parms_list parameter format for EC-DH mechanism..... 1351
557. parms_list parameter format for IKESEED, IKESHARE, and IKEREKEY mechanisms..... 1352
558. parms_list parameter format for Kyber mechanism..... 1352
559. Get attribute value processing for objects possessing sensitive attributes..... 1357
560. Keywords for generate secret key..... 1361
561. parms_list parameter format for SSL and TLS mechanism..... 1362
562. parms_list parameter format for PBEKEY mechanism..... 1363
563. parms_list parameter format for PBKDF2 mechanism..... 1363
564. Keywords for Generate Keyed MAC..... 1365
565. chain_data parameter format..... 1367
566. Keywords for Verify Keyed MAC..... 1369
567. chain_data parameter format..... 1371
568. Keywords for PKCS #11 One-Way Hash, Sign, or Verify..... 1373
569. chain_data parameter format on input (FIRST and ONLY for SIGN-PSS and VER-PSS)..... 1376
570. chain_data parameter format on input (FIRST and ONLY for non-PSS operations)..... 1377
571. chain_data parameter format on output (all calls) and input (MIDDLE and LAST)..... 1377

572. Keywords for private key sign..... 1380

573. Keywords for public key verify..... 1384

574. Keywords for PKCS #11 Pseudo-random function..... 1387

575. parms_list parameter format for TLS-PRF mechanism..... 1388

576. Authorization requirements for the set attribute value callable service..... 1390

577. Keywords for Secret Key Decrypt.....1392

578. initialization_vector parameter format for GCM mechanism and CHACHA20 mechanisms.....1394

579. initialization_vector parameter format for CTR mechanism..... 1394

580. chain_data parameter format..... 1395

581. Keywords for Secret Key Encrypt.....1398

582. initialization_vector parameter format for GCM mechanism and CHACHA20 mechanisms.....1400

583. initialization_vector parameter format for GCMIVGEN mechanism..... 1400

584. initialization_vector parameter format for CTR mechanism..... 1401

585. chain_data parameter format..... 1401

586. Rule Array Keywords for rule_array..... 1405

587. Token record create keywords..... 1409

588. Authorization requirements for the token record create callable service..... 1410

589. Token record delete keywords..... 1412

590. Authorization requirements for the token record delete callable service..... 1412

591. Token record list keywords..... 1414

592. Keywords for unwrap key..... 1419

593. initialization_vector parameter format for GCM mechanism..... 1420

594. Keywords for wrap key..... 1424

595. initialization_vector parameter format for GCM mechanism..... 1425

596. initialization_vector parameter format for GCMIVGEN mechanism..... 1426

597. Keywords for Private Key Structure Decrypt 1431
598. Keywords for Private Key Structure Sign 1433
599. Keywords for Public Key Structure Encrypt 1436
600. Keywords for Public Key Structure Verify 1438
601. Return Codes 1442
602. Reason codes for return code 0 (0) 1442
603. Reason codes for return code 4 (4) 1444
604. Reason codes for return code 8 (8) 1447
605. Reason codes for return code C (12) 1489
606. Reason codes for return code 10 (16) 1500
607. AES internal fixed-length key token format 1502
608. DES internal fixed-length key token format 1504
609. DES external fixed-length key token format 1506
610. External RKX DES key-token format, version X'10' 1508
611. Variable-length symmetric key token 1509
612. DESUSECV key-usage fields 1514
613. HMAC algorithm key-usage fields 1514
614. AES algorithm MAC key associated data 1516
615. AES algorithm PINCALC key associated data 1517
616. AES algorithm PINPROT key associated data 1518
617. AES algorithm PINPRW key associated data 1520
618. AES algorithm DKYGENKY key associated data 1522
619. AES algorithm SECMSG key associated data 1526
620. AES algorithm KEK key-usage fields 1527
621. AES algorithm CIPHER key associated data 1529

622. AES and HMAC algorithm key-management fields..... 1531

623. DESUSECV key-management fields.....1536

624. AES algorithm KDKGENKY key-usage fields.....1536

625. Variable-length symmetric null token.....1537

626. Format and supported values of the required header for a X9.143 key block.....1538

627. Key usage values and meanings..... 1541

628. IBM optional block data in a TR-31 key block, control vector (ID "10").....1544

629. IBM internal X9-SWKB controls (TLV ID ’02’) after conversion to binary.....1546

630. PKA key token section data structures.....1547

631. PKA key token header..... 1549

632. Null PKA key token format..... 1549

633. RSA private key, 1024-bit Modulus-Exponent format section (X'02') 1550

634. RSA private key, 1024-bit Modulus-Exponent format with OPK section (X'06').....1551

635. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30').....1552

636. RSA private key, 4096-bit Modulus-Exponent format section (X'09').....1557

637. RSA private key, Chinese-Remainder Theorem format with OPK section (X'08').....1559

638. RSA private key, 4096-bit Chinese-Remainder Theorem format with AES-encrypted OPK section (X'31').....1561

639. RSA public-key section (X'04').....1565

640. PKA private-key name section for RSA and QSA keys (X'10').....1566

641. ECC supported Brainpool elliptic curves by size, name, and object identifier..... 1566

642. ECC supported Prime elliptic curves by size, name, and object identifier.....1567

643. EC supported Edwards elliptic curves..... 1567

644. ECC supported Koblitz elliptic curves by size, name, and object identifier..... 1567

645. ECC private-key section (X'20').....1567

646. ECC public-key section (X'21').....1572

647. IBM extended associated data section TLV object....1574
648. ECC key-derivation section (X'23')....1574
649. Supported CRYSTALS-Dilithium strengths....1575
650. QSA Private Key section with OPK (X'50')....1575
651. QSA Public Key section (X'51')....1581
652. AESKW external format structure....1584
653. Trusted block sections....1588
654. Trusted block header....1590
655. Trusted block trusted RSA public-key section (X'11')....1591
656. Trusted block rule section (X'12')....1592
657. Summary of trusted block rule subsection....1593
658. Transport key variant subsection (X'0001') of trusted block rule section (X'12')....1594
659. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12')....1594
660. Common export key parameters subsection (X'0003') of trusted block rule section (X'12')....1595
661. Source key rule reference subsection (X'0004' of trusted block rule section (X'12')....1596
662. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12')....1597
663. Trusted block key label (name) section X'13'....1599
664. Trusted block information section X'14'....1599
665. Summary of trusted block information subsections....1599
666. Protection information subsection (X'0001') of trusted block information section (X'14')....1600
667. Activation/expiration dates subsection (X'0002') of trusted block information section (X'14')....1601
668. Trusted block application-defined data section X'15'....1601
669. Default control vector values....1603
670. Main key type for bits 8 to 11....1608
671. Key Subtype....1609

672. Padding bytes added according to the data length..... 1646

673. PKA96 Clear DES Key Record.....1649

674. EBCDIC to ASCII default conversion table..... 1657

675. ASCII to EBCDIC default conversion table..... 1658

676. Access control points affecting multiple services or requiring special consideration..... 1661

677. Access control points - Callable Services..... 1672

678. Callable services not compliant with PCI-HSM 2016..... 1695

679. Callable services that do not support compliant-tagged key tokens.....1700

680. Callable services that support compliant-tagged keys in cryptographic operations..... 1700

681. Using compliant-tagged keys to translate between PIN block formats in cryptographic operations..... 1703

682. Resource names for CCA and ICSF entry points..... 1705

683. Cryptographic functions used by ICSF 1717

684. AES-DUKPT derivation data..... 1719

685. Supported CCA keys types for AES-DUKPT derived working keys..... 1722

686. AES-DUKPT allowed derived working key sizes..... 1722

687. CCA release levels for IBM z16.....1723

688. CCA release levels for IBM z15.....1723

689. CCA release levels for the IBM z14.....1725

690. CCA release levels for the IBM z13.....1726

About this information

This information describes how to use the callable services that are provided by the Integrated Cryptographic Service Facility (ICSF). The z/OS Cryptographic Services include these components:

- z/OS Integrated Cryptographic Service Facility (ICSF)
- z/OS System Secure Socket Level Programming (SSL)
- z/OS Public Key Infrastructure Services (PKI)

ICSF is a software element of z/OS that works with hardware cryptographic features and the Security Server RACF to provide secure, high-speed cryptographic services. ICSF provides the application programming interfaces by which applications request the cryptographic services.

Who should use this information

This information is intended for application programmers who:

- Are responsible for writing application programs that use the security application programming interface (API) to access cryptographic functions.
- Want to use ICSF callable services in high-level languages such as C, COBOL, FORTRAN, and PL/I, as well as in assembler.

How to use this information

ICSF supports the IBM Common Cryptographic Architecture (CCA) and PKCS #11 APIs. This document describes the CCA callable services and the services that provide the functions behind the PKCS #11 API.

Topics focusing on programming the APIs

- "Introducing programming for ICSF" describes the programming considerations for using the ICSF callable services. It also explains the syntax and parameter definitions that are used in callable services.
- "Introducing CCA symmetric key cryptography" gives an overview of AES and DES cryptography and provides general guidance information on how the callable services use different key types and key forms. It also discusses how to write your own callable services that are called installation-defined callable services and provides suggestions on what to do if there is a problem.
- "Introducing CCA PKA cryptography and using PKA callable services" introduces Public Key Algorithm (PKA) support and describes programming considerations for using the ICSF PKA callable services, such as the PKA key token structure and key management.
- "Introducing PKCS #11 and using PKCS #11 callable services" gives an overview of PKCS #11 support and management services.

Topics focusing on CCA callable services

- "Managing symmetric cryptographic keys" describes the callable services for generating and maintaining cryptographic keys and the random number generate callable service. It also presents utilities to build AES and DES tokens and generate and translate control vectors and describes the PKA callable services that support AES and DES key distribution.
- "Protecting data" describes the callable services for deciphering ciphertext from one key and enciphering it under another key. It also describes enciphering and deciphering data with clear and encrypted keys.
- "Verifying data integrity and authenticating messages" describes the callable services for generating and verifying message authentication codes (MACs), generating modification detection codes (MDCs) and generating hashes (SHA-1, SHA-2, MD5, RIPEMD-160).

- "Financial services" describes the callable services for generating, verifying, and translating personal identification numbers (PINs), services for generating and verifying payment card security codes, services for format preserving encryption, and services for EMV processing.
- "Financial services for DK PIN methods" describes the financial services that are based on the PIN methods of and meet the requirements that are specified by the German Banking Industry Committee (Deutsche Kreditwirtschaft (DK)). DK is an association of the German banking industry. The intellectual property rights regarding the methods and specification belongs to the German Banking Industry Committee.
- "X9.143 (TR-31) symmetric-key management" provides information on services for the ANSI TR-31 protocol for key distribution.
- "TR-34 symmetric key management" provides information on services for the ANSI TR-34 protocol for key distribution.
- "Using digital signatures" describes the PKA callable services that support the use of digital signatures to authenticate messages.
- "Managing PKA cryptographic keys" describes the PKA callable services that generate and manage PKA keys.
- "Key data set management" describes the callable services that manage key tokens in the Cryptographic Key Data Set (CKDS) and the Public Key Data Set (PKDS).
- "Utilities" describes callable services that convert data between EBCDIC and ASCII format, convert between binary strings and character strings, and query ICSF services and algorithms.
- "Trusted interfaces" describes the service that supports Trusted Key Entry (TKE) workstation, an optional feature available with ICSF.

Topics focusing on PKCS #11 services

- "Using PKCS #11 tokens and objects" describes the callable services for managing the PKCS #11 tokens and objects in the TKDS.
- "Using the PKCS #11 key structure callable services" describes the callable services that use a PKCS #11 key structure instead of an object.

Appendixes

- "ICSF and cryptographic coprocessor return/reason codes" lists the return and reason codes that are returned by the ICSF callable services.
- "Key token formats" describes the formats for AES and DES key tokens including the variable-length symmetric key token, all formats and sections of RSA and ECC key tokens, and trusted blocks.
- "Changing control vectors with the CVT callable service" contains a table of the default control vector values that are associated with each key type and describes the control information for testing control vectors, mask array preparation, selecting the key-half processing mode, and an example of Control Vector Translate.
- "Coding examples" provides examples for COBOL, assembler, C, and PL/I.
- "Cryptographic algorithms and processes" describes the PIN formats and algorithms, cipher processing and segmenting rules, multiple encipherment and decipherment and their equations, and the PKA92 encryption process.
- "EBCDIC and ASCII default conversion tables" contains the EBCDIC to ASCII and ASCII to EBCDIC conversion tables.
- "Access control points and callable services" lists which access control points correspond to which callable services.
- "Impact of compliance mode on callable services" contains information on a compliance mode's effect on ICSF callable services and other operations.
- "Resource names for CCA and ICSF entry points" contains the resource names for CCA and ICSF entry points.
- "Cryptographic hardware engines and software" contains information about the cryptographic hardware engines and software used by ICSF.

- "AES-DUKPT reference information" contains information about the use of AES keys with derived unique key per transaction processing.
- "CCA release levels" lists the CCA release level, the associated ICSF release, APAR number (if applicable), and the hardware supported.

Where to find more information

The publications in the z/OS ICSF library include:

- *z/OS Cryptographic Services ICSF Overview.*
- *z/OS Cryptographic Services ICSF Administrator's Guide.*
- *z/OS Cryptographic Services ICSF System Programmer's Guide.*
- *z/OS Cryptographic Services ICSF Application Programmer's Guide.*
- *z/OS Cryptographic Services ICSF Messages.*
- *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications.*

Related Publications

- *z/OS Cryptographic Services ICSF TKE Workstation User's Guide.*
- *z/OS MVS Programming: Callable Services for High-Level Languages.*
- *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU.*
- *z/OS Security Server RACF Command Language Reference.*
- *z/OS Security Server RACF Security Administrator's Guide.*
- *IBM Common Cryptographic Architecture (CCA) Basic Services API.*

How to send your comments to IBM

We invite you to submit comments about the z/OS® product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead “If you have a technical problem” on page xlvii.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Documentation function
If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdoc@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS ICSF Application Programmer's Guide, SC14-7508-10
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under IBM z/OS Product Documentation Update Policy (www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument).

Changes made in Cryptographic Support for z/OS V2R5 (FMID HCR77D2)

This document is for ICSF FMID HCR77D2. This release of ICSF runs on z/OS V2R5 and only on zSeries hardware. The most recent updates are listed at the top of each section.

New

May 2023 refresh

- "Reason codes for return code 8 (8)" on page 1447:
  - 972 (2418)

April 2023 refresh

- New for APAR OA61978, which also applies to ICSF FMID HCR77D1:
  - "CCA DES control vectors" on page 16
  - "X9.143 (TR-31) key blocks" on page 28
  - "TR-31 Create Callable Service (CSNBT31C and CSNET31C)" on page 70
  - "TR-31 Create (CSNBT31C and CSNET31C)" on page 938
  - "Reason codes for return code 8 (8)" on page 1447:
    - 3EE (1006), 3EF (1007), 3F1 (1009), 3F2 (1010), 3F3 (1011), 3F5 (1013), 3F6 (1014), 3F7 (1015), 1391 (5009)
  - "Reason codes for return code C (12)" on page 1489:
    - D5C (3420)
  - "X9.143 (TR-31) key block header and optional block data" on page 1538

April 2022 refresh

- New for APAR OA61609, which also applies to ICSF FMID HCR77D1:
  - Information about IBM z16 and Crypto Express8 adapter (CEX8C, CEX8P, and CEX8A).
  - "Reason codes for return code 8 (8)" on page 1447:
    - 3E5 (997)
    - 3E6 (998)
  - "AESKW external format" on page 1584

March 2022 refresh

- New for APAR OA62763, which also applies to ICSF FMID HCR77D1:
  - "Reason codes for return code 0 (0)" on page 1442:

- 2715 (10005)
- 2716 (10006)

– “Reason codes for return code 8 (8)” on page 1447:

- 3ED (1005)

January 2022 refresh

- Updated for APAR OA61253, which also applies to ICSF FMID HCR77D1:
  – “Australian Payment Network support” on page 37
  – “Encrypted PIN Verify2 Callable Service (CSNBPVR2 and CSNEPVR2)” on page 66
  – “Encrypted PIN Verify2” on page 606
  – “Enhanced PIN checking for CSNBPTR and CSNBPTR2” on page 609
  – “PIN block error processing mode” on page 609
  – “Encrypted PIN Verify2 (CSNBPVR2 and CSNEPVR2)” on page 720
  – “PKCS #11 Secret Key Reencrypt (CSFPSKR and CSFPSKR6)” on page 1403
  – “Reason codes for return code 8 (8)” on page 1447:
    - 8FA (2298)
    - 9D2 (2514)

Prior to the January 2022 refresh

- “Reason codes for return code 0 (0)” on page 1442:
  – D5F (3423)
- “Reason codes for return code 8 (8)” on page 1447:
  – D5E (3422)
  – D60 (3424)
- “Reason codes for return code C (12)” on page 1489:
  – DDF (3551)

Changed

June 2023 refresh

- Miscellaneous editorial updates.

April 2023 refresh

- Updated for APAR OA61978, which also applies to ICSF FMID HCR77D1:
  – “Key identifier for key token” on page 7
  – “Key label” on page 8
  – “Functions of symmetric cryptographic keys” on page 15
  – “Key separation” on page 16
  – “Key token” on page 17
  – “Compliant-tagged key tokens” on page 19
  – “CCA key wrapping” on page 20
  – “ANSI X9.143 (TR-31) key block support” on page 70
  – “TR-31 Translate Callable Service (CSNBT31X and CSNET31X)” on page 71
  – “Data Key Export (CSNBDKX and CSNEDKX)” on page 130
  – “Data Key Import (CSNBDKM and CSNEDKM)” on page 133

- “Diversified Key Generate (CSNBDKG and CSNEDKG)” on page 151
- “Diversified Key Generate2 (CSNBDKG2 and CSNEDKG2)” on page 162
- “ECC Diffie-Hellman (CSNDEDH and CSNFEDH)” on page 182
- “Key Export (CSNBKEX and CSNEKEX)” on page 207
- “Key Generate (CSNBKGN and CSNEKGN)” on page 211
- “Key Generate2 (CSNBKGN2 and CSNEKGN2)” on page 224
- “Key Import (CSNBKIM and CSNEKIM)” on page 237
- “Key Part Import2 (CSNBKPI2 and CSNEKPI2)” on page 247
- “Key Test2 (CSNBKYT2 and CSNEKYT2)” on page 256
- “Key Translate2 (CSNBKTR2 and CSNEKTR2)” on page 346
- “Multiple Secure Key Import (CSNBSKM and CSNESKM)” on page 358
- “PKA Decrypt (CSNDPKD and CSNFPKD)” on page 365
- “PKA Encrypt (CSNDPKE and CSNFPKE)” on page 374
- “Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)” on page 388
- “Restrict Key Attribute (CSNBRKA and CSNERKA)” on page 402
- “Secure Key Import2 (CSNBSKI2 and CSNESKI2)” on page 412
- “Symmetric Key Export (CSNDSYX and CSNFSYX)” on page 417
- “Symmetric Key Export with Data (CSNDSXD and CSNFSXD)” on page 426
- “Symmetric Key Generate (CSNDSYG and CSNFSYG)” on page 430
- “Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)” on page 444
- “Unique Key Derive (CSNBUKD and CSNEUKD)” on page 454
- “Cipher Text Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3)” on page 471
- “Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)” on page 484
- “Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)” on page 492
- “Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)” on page 501
- “Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)” on page 509
- “Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)” on page 521
- “Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)” on page 531
- “HMAC Generate (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)” on page 545
- “HMAC Verify (CSNBHMV or CSNBHMV1 and CSNEHMV or CSNEHMV1)” on page 550
- “MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)” on page 555
- “MAC Generate2 (CSNBMGN2, CSNBMGN3, CSNEMGN2, and CSNEMGN3)” on page 562
- “MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)” on page 567
- “MAC Verify2 (CSNBMVR2, CSNBMVR3, CSNEMVR2, and CSNEMVR3)” on page 574
- “Authentication Parameter Generate (CSNBAPG and CSNEAPG)” on page 620
- “Clear PIN Encrypt (CSNBCPE and CSNECPE)” on page 624
- “Clear PIN Generate (CSNBPGN and CSNEPGN)” on page 629
- “Clear PIN Generate Alternate (CSNBCPA and CSNECPA)” on page 634
- “Encrypted PIN Generate (CSNBEPG and CSNEEPG)” on page 671
- “Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)” on page 685
- “Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)” on page 701
- “Encrypted PIN Verify (CSNBPVR and CSNEPVR)” on page 713

- "Encrypted PIN Verify2 (CSNBPVR2 and CSNEPVR2)" on page 720
- "Field Level Decipher (CSNBFLD and CSNEFLD)" on page 730
- "Field Level Encipher (CSNBFLE and CSNEFLE)" on page 739
- "Format Preserving Algorithms Decipher (CSNBFFXD and CSNEFFXD)" on page 749
- "Format Preserving Algorithms Encipher (CSNBFFXE and CSNEFFXE)" on page 754
- "Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT)" on page 760
- "FPE Decipher (CSNBFPED and CSNEFPED)" on page 767
- "FPE Encipher (CSNBFPEE and CSNEFPEE)" on page 776
- "FPE Translate (CSNBFPET and CSNEFPET)" on page 786
- "PIN Change/Unblock (CSNBPCU and CSNEPCU)" on page 795
- "Recover PIN from Offset (CSNBPFO and CSNEPFO)" on page 806
- "Secure Messaging for Keys (CSNBSKY and CSNESKY)" on page 811
- "Secure Messaging for PINs (CSNBSPN and CSNESPN)" on page 816
- "Transaction Validation (CSNBTRV and CSNETRV)" on page 834
- "VISA CVV Service Generate (CSNBCSG and CSNECSG)" on page 838
- "VISA CVV Service Verify (CSNBCSV and CSNECSV)" on page 843
- "TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)" on page 1007
- "TR-31 Import (CSNBT31I and CSNET31I)" on page 959
- "TR-34 Key Distribution (CSNDT34D and CSNFT34D)" on page 1086
- "TR-34 Key Receive (CSNDT34R and CSNFT34R)" on page 1098
- "Signature algorithms and formatting methods" on page 1109
- "Digital Signature Generate (CSNDDSG and CSNFDSG)" on page 1110
- "Digital Signature Verify (CSNDDSV and CSNFDSV)" on page 1119
- "PKA Key Generate (CSNDPKG and CSNFPKG)" on page 1131
- "PKA Key Import (CSNDPKI and CSNFPKI)" on page 1140
- "PKA Key Translate (CSNDPKT and CSNFPKT)" on page 1169
- "CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2)" on page 1203
- "CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)" on page 1208
- "CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)" on page 1215
- "Key Data Set List (CSFKDSL and CSFKDSL6)" on page 1225
- "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 1278
- "Reason codes for return code 8 (8)" on page 1447:

  - 807 (2055), 86A (2154), 2AF8 (11000)
- "PKCS #1 formats" on page 1651
- Appendix G, "Access control points and callable services," on page 1661
- Appendix H, "Impact of compliance mode on callable services," on page 1695
- Appendix I, "Resource names for CCA and ICSF entry points," on page 1705
- Appendix L, "CCA release levels," on page 1723

August 2022 refresh

- Updated for APAR OA63531, which also applies to ICSF FMID HCR77D1 and FMID HCR77D0:
  - Key usages now allow Mode of use "N" with all wrapping methods in "TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)" on page 1007 and "TR-31 Import (CSNBT31I and CSNET31I)" on page 959.

- "TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)" on page 1007 allows export of an IMPORTER/EXPORTER as "K0" key usage with 'B' Mode of use.
- Appendix G, "Access control points and callable services," on page 1661 was updated with two new ACPs (T31X - Permit EXPORTER to K0:B and T31X - Permit IMPORTER to K0:B).

June 2022 refresh

"TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)" on page 1007 was updated.

April 2022 refresh

- Updated for APAR OA61609, which also applies to ICSF FMID HCR77D1:
  - "Trusted Key Entry (TKE) support" on page 71
  - "PKA key algorithms" on page 93
  - "PKA key tokens" on page 100
  - "Key identifier for PKA key token" on page 103
  - "Key token" on page 103
  - "Summary of the PKA callable services" on page 104
  - "Control Vector Generate (CSNBCVG and CSNECVG)" on page 118
  - "ECC Diffie-Hellman (CSNDEDH and CSNFEDH)" on page 182
  - "Key Test2 (CSNBKYT2 and CSNEKYT2)" on page 256
  - "Key Token Build (CSNBKTB and CSNEKTB)" on page 271
  - "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297
  - "PKA Decrypt (CSNDPKD and CSNFPKD)" on page 365
  - "PKA Encrypt (CSNDPKE and CSNFPKE)" on page 374
  - "FPE Decipher (CSNBFPED and CSNEFPED)" on page 767
  - "FPE Encipher (CSNBFPEE and CSNEFPEE)" on page 776
  - "FPE Translate (CSNBFPET and CSNEFPET)" on page 786
  - "Digital Signature Generate (CSNDDSG and CSNFDSG)" on page 1110
  - "Digital Signature Verify (CSNDDSV and CSNFDSV)" on page 1119
  - "PKA Key Generate (CSNDPKG and CSNFPKG)" on page 1131
  - "PKA Key Import (CSNDPKI and CSNFPKI)" on page 1140
  - "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 1147
  - "PKA Key Token Change (CSNDKTC and CSNFKTC)" on page 1165
  - "PKA Key Translate (CSNDPKT and CSNFPKT)" on page 1169
  - "PKA Public Key Extract (CSNDPKX and CSNFPKX)" on page 1181
  - "ICSF Query Algorithm (CSFIQA and CSFIQA6)" on page 1273
  - "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 1278
  - "PCI Interface (CSFPCI and CSFPCI6)" on page 1327
  - "PKCS #11 Derive Key (CSFPDVK and CSFPDVK6)" on page 1347
  - "PKCS #11 Generate Key Pair (CSFPGKP and CSFPGKP6)" on page 1357
  - "PKCS #11 Private Key Sign (CSFPPKS and CSFPPKS6)" on page 1379
  - Appendix G, "Access control points and callable services," on page 1661

March 2022 refresh

- Updated for APAR OA62763, which also applies to ICSF FMID HCR77D1:
  - "TR-34 Bind-Begin (CSNDT34B and CSNFT34B)" on page 1072
  - "TR-34 Bind-Complete (CSNDT34C and CSNFT34C)" on page 1079

- "TR-34 Key Distribution (CSNDT34D and CSNFT34D)" on page 1086
- "TR-34 Key Receive (CSNDT34R and CSNFT34R)" on page 1098
- Appendix G, "Access control points and callable services," on page 1661

January 2022 refresh

- "TR-34 Key Distribution (CSNDT34D and CSNFT34D)" on page 1086 (APAR OA62253, which also applies to ICSF FMID HCR77D0 and HCR77D1)
- Updated for APAR OA61253, which also applies to ICSF FMID HCR77D1:
  - "Summary of callable services" on page 76
  - "Summary of the PKA callable services" on page 104
  - "PKCS #11 services" on page 109
  - "Diversified Key Generate (CSNBDKG and CSNEDKG)" on page 151
  - "Diversified Key Generate2 (CSNBDKG2 and CSNEDKG2)" on page 162
  - "Diversify Directed Key (CSNBDDK and CSNEDDK)" on page 170
  - "Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)" on page 388
  - "Symmetric Key Export (CSNDSYX and CSNFSYX)" on page 417
  - "Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)" on page 501
  - "Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)" on page 509
  - "Verifying a PIN" on page 603
  - "Derived unique key per transaction algorithms" on page 605
  - "Encrypted PIN Translate and Encrypted PIN Translate2 " on page 605
  - "Encrypted PIN Translate Enhanced" on page 605
  - "Encrypted PIN Verify" on page 606
  - "Encrypted PIN Translate (CSNBPTR and CSNEPTR)" on page 678
  - "Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)" on page 685
  - "PIN Change/Unblock (CSNBPCU and CSNEPCU)" on page 795
  - "Secure Messaging for PINs (CSNBSPN and CSNESPN)" on page 816
  - "SET Block Decompose (CSNDSBD and CSNFSBD)" on page 828
  - "DK PIN Change (CSNBDPC and CSNEDPC)" on page 878
  - "DK PIN Verify (CSNBDPV and CSNEDPV)" on page 892
  - "Signature algorithms and formatting methods" on page 1109
  - "Digital Signature Generate (CSNDDSG and CSNFDSG)" on page 1110
  - "Digital Signature Verify (CSNDDSV and CSNFDSV)" on page 1119
  - "PKA Key Translate (CSNDPKT and CSNFPKT)" on page 1169
  - "ICSF Query Facility2 (CSFIQF2 and CSFIQF26)" on page 1316
  - "PKCS #11 Private Key Sign (CSFPPKS and CSFPPKS6)" on page 1379
  - "PKCS #11 Public Key Verify (CSFPPKV and CSFPPKV6)" on page 1382
  - "Reason codes for return code 8 (8)" on page 1447: DD6 (3542), DD8 (3544)
  - Appendix G, "Access control points and callable services," on page 1661

Prior to the January 2022 refresh

- "Master keys" on page 94
- "Security and integrity of the token" on page 102

- "Key Test2 (CSNBKYT2 and CSNEKYT2)" on page 256
- "TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)" on page 1007
- "TR-31 Optional Data Read (CSNBT31R and CSNET31R)" on page 1001
- "TR-31 Parse (CSNBT31P and CSNET31P)" on page 1004
- "PKA Key Generate (CSNDPKG and CSNFPKG)" on page 1131
- "PKA Key Import (CSNDPKI and CSNFPKI)" on page 1140
- "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 1147
- "Metadata for key data set records" on page 1199
- "CKDS Key Record Delete (CSNBKRD and CSNEKRD)" on page 1205
- "CKDS Key Record Read (CSNBKRR and CSNEKRR)" on page 1207
- "CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)" on page 1208
- "CKDS Key Record Write (CSNBKRW and CSNEKRW)" on page 1213
- "CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)" on page 1215
- "Coordinated KDS Administration (CSFCRC and CSFCRC6)" on page 1217
- "Key Data Set List (CSFKDSL and CSFKDSL6)" on page 1225
- "Key Data Set Metadata Read (CSFKDMR and CSFKDMR6)" on page 1239
- "Key Data Set Metadata Write (CSFKDMW and CSFKDMW6)" on page 1246
- "Key Data Set Record Retrieve (CSFRRT and CSFRRT6)" on page 1252
- "PKDS Key Record Create (CSNDKRC and CSNFKRC)" on page 1257
- "PKDS Key Record Write (CSNDKRW and CSNFKRW)" on page 1263
- "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 1278
- "PKCS #11 Generate Secret Key (CSFPGSK and CSFPGSK6)" on page 1360

Deleted

No content was removed from this information.

Changes made in Cryptographic Support for z/OS V2R2 - z/OS V2R4 (FMID HCR77D1)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SC14-7508-08.

This document is for ICSF FMID HCR77D1. This release of ICSF runs on z/OS V2R2, z/OS V2R3, and z/OS V2R4 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

New

May 2021 refresh

- "Reason codes for return code 8 (8)" on page 1447:
  - AF (175), 043B (1083), and 043D (1085) (APAR OA60318)
  - 9D1 (2513)
- "Reason codes for return code C (12)" on page 1489:
  - C59 (3161)

December 2020 refresh

- "Reason codes for return code 8 (8)" on page 1447: 43A (1082) (APAR OA60165)

October 2020 refresh

- "Format preserving encryption" on page 66 (APAR OA59593)
- "Format Preserving Algorithms Decipher (CSNBFFXD and CSNEFFXD)" on page 749 (APAR OA59593)
- "Format Preserving Algorithms Encipher (CSNBFFXE and CSNEFFXE)" on page 754 (APAR OA59593)
- "Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT)" on page 760 (APAR OA59593)
- The following reason codes are new:
  - "Reason codes for return code 8 (8)" on page 1447:
    - 09CE (2510), 09CF (2511), DD6 (3542), DD7 (3543), DD8 (3544), and DD9 (3545) (APAR OA59593)
- Appendix K, "AES-DUKPT reference information," on page 1719 (APAR OA59593)

Prior to the October 2020 refresh

- Information about IBM z15.
- "Deprecated callable services" on page 10
- The following reason codes are new:
  - "Reason codes for return code 8 (8)" on page 1447:
    - 439 (1081) (APAR OA58306)
    - 762 (2FA) (APAR OA58880)
    - 2182 (886) (APAR OA58880)
  - "Reason codes for return code C (12)" on page 1489:
    - DDE (3550)
- Appendix J, "Cryptographic hardware engines and software," on page 1715

Changed

May 2021 refresh

- "Compliant-tagged key tokens" on page 19 (APAR OA60318)
- "CCA key wrapping" on page 20 (APAR OA60318)
- "MAC Generate callable service (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)" on page 61 (APAR OA60318)
- "MAC Verify callable service (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)" on page 62 (APAR OA60318)
- "Summary of callable services" on page 76 (APAR OA60318)
- "Control Vector Translate (CSNBCVT and CSNECVT)" on page 124 (APAR OA60318)
- "Data Key Export (CSNBDKX and CSNEDKX)" on page 130 (APAR OA60318)
- "Data Key Import (CSNBDKM and CSNEDKM)" on page 133 (APAR OA60318)
- "Diversified Key Generate (CSNBDKG and CSNEDKG)" on page 151 (APAR OA60318)
- "ECC Diffie-Hellman (CSNDEDH and CSNFEDH)" on page 182 (APAR OA60318)
- "Key Export (CSNBKEX and CSNEKEX)" on page 207 (APAR OA60318)
- "Key Generate (CSNBKGN and CSNEKGN)" on page 211 (APAR OA60318)

- “Key Import (CSNBKIM and CSNEKIM)” on page 237 (APAR OA60318)
- “Key Part Import (CSNBKPI and CSNEKPI)” on page 242 (APAR OA60318)
- “Key Test2 (CSNBKYT2 and CSNEKYT2)” on page 256 (APAR OA60318)
- “Key Token Build (CSNBKTB and CSNEKTB)” on page 271 (APAR OA60318)
- “Key Translate (CSNBKTR and CSNEKTR)” on page 343 (APAR OA60318)
- “Key Translate2 (CSNBKTR2 and CSNEKTR2)” on page 346 (APAR OA60318)
- “Multiple Clear Key Import (CSNBCKM and CSNECKM)” on page 355 (APAR OA60318)
- “Multiple Secure Key Import (CSNBSKM and CSNESKM)” on page 358 (APAR OA60318)
- “Prohibit Export (CSNBPEX and CSNEPEX)” on page 384 (APAR OA60318)
- “Prohibit Export Extended (CSNBPEXX and CSNEPEXX)” on page 386 (APAR OA60318)
- “Remote Key Export (CSNDRKX and CSNFRKX)” on page 393 (APAR OA60318)
- “Secure Key Import (CSNBSKI and CSNESKI)” on page 408 (APAR OA60318)
- “Symmetric Key Generate (CSNDSYG and CSNFSYG)” on page 430 (APAR OA60318)
- “Symmetric Key Import (CSNDSYI and CSNFSYI)” on page 439 (APAR OA60318)
- “Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)” on page 444 (APAR OA60318)
- “TR-31 Import (CSNBT31I and CSNET31I)” on page 959 (APAR OA60318)
- “Unique Key Derive (CSNBUKD and CSNEUKD)” on page 454(APAR OA60318)
- “MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)” on page 555 (APAR OA60318)
- “MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)” on page 567 (APAR OA60318)
- “CVV Key Combine (CSNBCKC and CSNECKC)” on page 640 (APAR OA60318)
- “TR-34 Key Distribution (CSNDT34D and CSNFT34D)” on page 1086 (APAR OA60365)
- “TR-34 Key Receive (CSNDT34R and CSNFT34R)” on page 1098 (APAR OA60318)
- “ICSF Query Facility (CSFIQF and CSFIQF6)” on page 1278 (APAR OA60318)
- “Reason codes for return code 4 (4)” on page 1444: 8D10 (36112) (APAR OA60318)
- “Reason codes for return code 8 (8)” on page 1447: 3E90 (16016) (APAR OA60318)
- “DES fixed-length key token” on page 1503 (APAR OA60318)
- “Wrapping methods for symmetric key tokens ” on page 1647 (APAR OA60318)
- Appendix G, “Access control points and callable services,” on page 1661 (APAR OA60318)
- Appendix H, “Impact of compliance mode on callable services,” on page 1695 (APAR OA60318)

December 2020 refresh

- “HMAC Generate (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)” on page 545 (APAR OA60317)
- “HMAC Verify (CSNBHMV or CSNBHMV1 and CSNEHMV or CSNEHMV1)” on page 550 (APAR OA60317)
- “MAC Generate2 (CSNBMGN2, CSNBMGN3, CSNEMGN2, and CSNEMGN3)” on page 562 (APAR OA60317)
- “MAC Verify2 (CSNBMVR2, CSNBMVR3, CSNEMVR2, and CSNEMVR3)” on page 574 (APAR OA60317)
- Appendix J, “Cryptographic hardware engines and software,” on page 1715 (APAR OA60317)
- “Key strength and key wrapping access control points” on page 34 (APAR OA60165)
- Appendix G, “Access control points and callable services,” on page 1661 (APAR OA60165)

October 2020 refresh

- "AES key types" on page 26 (APAR OA59593)
- "Summary of callable services" on page 76 (APAR OA59593)
- "Digital Signature Generate callable service (CSNDDSG and CSNFDSG)" on page 96 (APAR OA59593)
- "Digital Signature Verify callable service (CSNDDSV and CSNFDSV)" on page 96 (APAR OA59593)
- "ECC Diffie-Hellman (CSNDEDH and CSNFEDH)" on page 182 (APAR OA59593)
- "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297 (APAR OA59593)
- "TR-31 Import (CSNBT31I and CSNET31I)" on page 959 (APAR OA59593)
- "Unique Key Derive (CSNBUKD and CSNEUKD)" on page 454 (APAR OA59593)
- "The PIN profile" on page 609 (APAR OA59593)
- "Pad digit" on page 612 (APAR OA59593)
- "Format preserving encryption" on page 614 (APAR OA59593)
- "Clear PIN Generate Alternate (CSNBCPA and CSNECPA)" on page 634 (APAR OA59593)
- "Encrypted PIN Generate (CSNBEPG and CSNEEPG)" on page 671 (APAR OA59593)
- "Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)" on page 685 (APAR OA59593)
- "Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)" on page 701 (APAR OA59593)
- "Encrypted PIN Verify (CSNBPVR and CSNEPVR)" on page 713 (APAR OA59593)
- "FPE Decipher (CSNBFPED and CSNEFPED)" on page 767 (APAR OA59593)
- "FPE Encipher (CSNBFPEE and CSNEFPEE)" on page 776 (APAR OA59593)
- "FPE Translate (CSNBFPET and CSNEFPET)" on page 786 (APAR OA59593)
- "PIN Change/Unblock (CSNBPCU and CSNEPCU)" on page 795 (APAR OA59593)
- "Recover PIN from Offset (CSNBPFO and CSNEPFO)" on page 806 (APAR OA59593)
- "Secure Messaging for PINs (CSNBSPN and CSNESPN)" on page 816 (APAR OA59593)
- "Signature algorithms and formatting methods" on page 1109 (APAR OA59593)
- "Digital Signature Generate (CSNDDSG and CSNFDSG)" on page 1110 (APAR OA59593)
- "Digital Signature Verify (CSNDDSV and CSNFDSV)" on page 1119 (APAR OA59593)
- "PKA Key Generate (CSNDPKG and CSNFPKG)" on page 1131 (APAR OA59593)
- "PKA Key Import (CSNDPKI and CSNFPKI)" on page 1140 (APAR OA59593)
- "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 1147 (APAR OA59593)
- "PKA Key Token Change (CSNDKTC and CSNFKTC)" on page 1165 (APAR OA59593)
- "ICSF Query Algorithm (CSFIQA and CSFIQA6)" on page 1273 (APAR OA59593)
- "ICSF Query Facility2 (CSFIQF2 and CSFIQF26)" on page 1316 (APAR OA59593)
- The following reason code is changed:
  - "Reason codes for return code 8 (8)" on page 1447:
    - 377 (887), 37B (891), 8C3 (2243), 177F (6015) (APAR OA59593)
- "Variable-length symmetric key token" on page 1509 (APAR OA59593)
- "Number representation in PKA key tokens" on page 1549 (APAR OA59593)
- "DES control vector table" on page 1603 (APAR OA59593)
- Appendix G, "Access control points and callable services," on page 1661 (APAR OA59593)
- Appendix I, "Resource names for CCA and ICSF entry points," on page 1705 (APAR OA59593)

June 2020 refresh

- "EMV simplification services" on page 67 (APAR OA58880)

- "ANSI X9.143 (TR-31) key block support" on page 70 (APAR OA58880)
- "PKA key algorithms" on page 93 (APAR OA58880)
- "Callable services supporting digital signatures" on page 96 (APAR OA58880)
- "PKA key tokens" on page 100 (APAR OA58880)
- "Key token" on page 103 (APAR OA58880)
- "Summary of the PKA callable services" on page 104 (APAR OA58880)
- "Derive ICC MK (CSNBDCM and CSNEDCM)" on page 136 (APAR OA58880)
- "Derive Session Key (CSNBDSK and CSNEDSK)" on page 143 (APAR OA58880)
- "Diversify Directed Key (CSNBDDK and CSNEDDK)" on page 170 (APAR OA58880)
- "Generate Issuer MK (CSNBGIM and CSNEGIM)" on page 197 (APAR OA58880)
- "Key Generate2 (CSNBKGN2 and CSNEKGN2)" on page 224 (APAR OA58880)
- "Key Test2 (CSNBKYT2 and CSNEKYT2)" on page 256 (APAR OA58880)
- "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297 (APAR OA58880)
- "TR-31 Import (CSNBT31I and CSNET31I)" on page 959 (APAR OA58880)
- "TR-31 Parse (CSNBT31P and CSNET31P)" on page 1004 (APAR OA58880)
- "EMV Transaction (ARQC/ARPC) Service (CSNBEAC and CSNEEAC)" on page 657 (APAR OA58880)
- "DK PIN Change (CSNBDPC and CSNEDPC)" on page 878 (APAR OA58880)
- "Signature algorithms and formatting methods" on page 1109 (APAR OA58880)
- "Digital Signature Generate (CSNDDSG and CSNFDSG)" on page 1110 (APAR OA58880)
- "Digital Signature Verify (CSNDDSV and CSNFDSV)" on page 1119 (APAR OA58880)
- "PKA Key Generate (CSNDPKG and CSNFPKG)" on page 1131 (APAR OA58880)
- "PKA Key Import (CSNDPKI and CSNFPKI)" on page 1140 (APAR OA58880)
- "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 1147 (APAR OA58880)
- "PKA Key Token Change (CSNDKTC and CSNFKTC)" on page 1165 (APAR OA58880)
- "PKA Public Key Extract (CSNDPKX and CSNFPKX)" on page 1181 (APAR OA58880)
- "PKDS Key Record Create (CSNDKRC and CSNFKRC)" on page 1257 (APAR OA58880)
- "PKDS Key Record Write (CSNDKRW and CSNFKRW)" on page 1263 (APAR OA58880)
- "ICSF Query Algorithm (CSFIQA and CSFIQA6)" on page 1273 (APAR OA58880)
- "PKCS #11 Private Key Sign (CSFPPKS and CSFPPKS6)" on page 1379 (APAR OA58880)
- "PKA key tokens" on page 1546 (APAR OA58880)
- Appendix G, "Access control points and callable services," on page 1661 (APAR OA58880)

Prior to the June 2020 refresh

- "Key Data Set Update (CSFKDU and CSFKDU6)" on page 1254 (APAR OA56203)
- "TR-34 Key Distribution (CSNDT34D and CSNFT34D)" on page 1086 (APAR OA59020)
- "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297 (APAR OA58306)
- "Clear PIN Encrypt (CSNBCPE and CSNECPE)" on page 624 (APAR OA58306)
- "Clear PIN Generate Alternate (CSNBCPA and CSNECPA)" on page 634 (APAR OA58306)
- "Encrypted PIN Generate (CSNBEPG and CSNEEPG)" on page 671 (APAR OA58306)
- "Encrypted PIN Translate (CSNBPTR and CSNEPTR)" on page 678 (APAR OA58306)
- "Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)" on page 685 (APAR OA58306)
- "Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)" on page 701 (APAR OA58306)
- "Encrypted PIN Verify (CSNBPVR and CSNEPVR)" on page 713 (APAR OA58306)
- "PIN Change/Unblock (CSNBPCU and CSNEPCU)" on page 795 (APAR OA58306)

- "Recover PIN from Offset (CSNBPFO and CSNEPFO)" on page 806 (APAR OA58306)
- "Secure Messaging for PINs (CSNBSPN and CSNESPN)" on page 816 (APAR OA58306)
- "DK Migrate PIN (CSNBDMP and CSNEDMP)" on page 857 (APAR OA58306)
- "DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)" on page 863 (APAR OA58306)
- "DK PIN Change (CSNBDPC and CSNEDPC)" on page 878 (APAR OA58306)
- "DK PIN Verify (CSNBDPV and CSNEDPV)" on page 892 (APAR OA58306)
- "Variable-length symmetric key token" on page 1509 (APAR OA58306)
- Appendix G, "Access control points and callable services," on page 1661 (APAR OA58306)
- "PKCS #11 Derive Key (CSFPDVK and CSFPDVK6)" on page 1347 (APAR OA58358)
- "PKCS #11 Generate Keyed MAC (CSFPHMG and CSFPHMG6)" on page 1364 (APAR OA58358)
- "PKCS #11 Verify Keyed MAC (CSFPHMV and CSFPHMV6)" on page 1368 (APAR OA58358)
- "PKCS #11 Private Key Sign (CSFPPKS and CSFPPKS6)" on page 1379 (APAR OA58358)
- "PKCS #11 Public Key Verify (CSFPPKV and CSFPPKV6)" on page 1382 (APAR OA58358)
- "Key Token Build (CSNBKTB and CSNEKTB)" on page 271 (APAR OA58186)
- "Digital Signature Generate (CSNDDSG and CSNFDSG)" on page 1110
- "Digital Signature Verify (CSNDDSV and CSNFDSV)" on page 1119
- "Key Data Set List (CSFKDSL and CSFKDSL6)" on page 1225
- "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 1278
- "ICSF Query Facility2 (CSFIQF2 and CSFIQF26)" on page 1316
- The following reason codes have been updated:
  - "Reason codes for return code 8 (8)" on page 1447:
    - CE8 (3304)
    - CFC (3324)

Deleted

No content was removed from this information.

Changes made in Cryptographic Support for z/OS V2R2 - z/OS V2R3 (FMID HCR77D0)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SC14-7508-07.

This document is for ICSF FMID HCR77D0. This release of ICSF runs on z/OS V2R2 and z/OS V2R3 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

New

December 2020 refresh

- "Reason codes for return code 8 (8)" on page 1447: 43A (1082) (APAR OA60165)

Prior to the December 2020 refresh

- "Compliant-tagged key tokens" on page 19 (APAR OA57089)
- "Callable services for the TR-34" on page 99 (APAR OA57089)

- "DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2)" on page 904 (APAR OA57089)
- "DK Random PIN Generate2 (CSNBDRG2 and CSNEDRG2)" on page 922 (APAR OA57089)
- Chapter 11, "TR-34 symmetric key management," on page 1059 (APAR OA57089)
  - "TR-34 Bind-Begin (CSNDT34B and CSNFT34B)" on page 1072
  - "TR-34 Bind-Complete (CSNDT34C and CSNFT34C)" on page 1079
  - "TR-34 Key Distribution (CSNDT34D and CSNFT34D)" on page 1086
  - "TR-34 Key Receive (CSNDT34R and CSNFT34R)" on page 1098
- The following new reason codes have been added:
  - "Reason codes for return code 4 (4)" on page 1444:
    - D7 (215) (APAR OA57088)
    - 138F (5007) (APAR OA57089)
  - "Reason codes for return code 8 (8)" on page 1447:
    - 37B (891) (APAR OA57089)
    - 38B (907) (APAR OA57089)
    - 38F (911) (APAR OA57089)
    - 393 (915) (APAR OA57089)
    - 3A5 (933) (APAR OA57089)
    - 3C5 (965) (APAR OA57089)
    - 3C6 (966) (APAR OA57089)
    - 3C7 (967) (APAR OA57089)
    - 3C9 (969) (APAR OA57089)
    - 3CA (970) (APAR OA57089)
    - 3CB (971) (APAR OA57089)
    - 3CD (973) (APAR OA57089)
    - 3CE (974) (APAR OA57089)
    - 3CF (975) (APAR OA57089)
    - 3D1 (977) (APAR OA57089)
    - 3D2 (978) (APAR OA57089)
    - 3D3 (979) (APAR OA57089)
    - 3D5 (981) (APAR OA57089)
    - 3D6 (982) (APAR OA57089)
    - 3D7 (983) (APAR OA57089)
    - 3D9 (985) (APAR OA57089)
    - 3DA (986) (APAR OA57089)
    - 3DB (987) (APAR OA57089)
    - 3DD (989) (APAR OA57089)
    - 3DE (990) (APAR OA57089)
    - 3DF (991) (APAR OA57089)
    - 3E1 (993) (APAR OA57089)
    - 3E2 (994) (APAR OA57089)
    - 3E3 (995) (APAR OA57089)
    - 439 (1081) (APAR OA58306)

Changed

December 2020 refresh

- "Key strength and key wrapping access control points" on page 34 (APAR OA60165)
- Appendix G, "Access control points and callable services," on page 1661 (APAR OA60165)

Prior to the December 2020 refresh

- "TR-34 Key Distribution (CSNDT34D and CSNFT34D)" on page 1086 (APAR OA59020)
- "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297 (APAR OA58306)
- "Clear PIN Encrypt (CSNBCPE and CSNECPE)" on page 624 (APAR OA58306)
- "Clear PIN Generate Alternate (CSNBCPA and CSNECPA)" on page 634 (APAR OA58306)
- "Encrypted PIN Generate (CSNBEPG and CSNEEPG)" on page 671 (APAR OA58306)
- "Encrypted PIN Translate (CSNBPTR and CSNEPTR)" on page 678 (APAR OA58306)
- "Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)" on page 685 (APAR OA58306)
- "Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)" on page 701 (APAR OA58306)
- "Encrypted PIN Verify (CSNBPVR and CSNEPVR)" on page 713 (APAR OA58306)
- "PIN Change/Unblock (CSNBPCU and CSNEPCU)" on page 795 (APAR OA58306)
- "Recover PIN from Offset (CSNBPFO and CSNEPFO)" on page 806 (APAR OA58306)
- "Secure Messaging for PINs (CSNBSPN and CSNESPN)" on page 816 (APAR OA58306)
- "DK Migrate PIN (CSNBDMP and CSNEDMP)" on page 857 (APAR OA58306)
- "DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)" on page 863 (APAR OA58306)
- "DK PIN Change (CSNBDPC and CSNEDPC)" on page 878 (APAR OA58306)
- "DK PIN Verify (CSNBDPV and CSNEDPV)" on page 892 (APAR OA58306)
- "Variable-length symmetric key token" on page 1509 (APAR OA58306)
- Appendix G, "Access control points and callable services," on page 1661 (APAR OA58306)
- "Key Token Build (CSNBKTB and CSNEKTB)" on page 271 (APAR OA58186)
- "DES key types" on page 23 (APAR OA57089)
- "AES key types" on page 26 (APAR OA57089)
- "X.509 certificates" on page 101 (APAR OA57089)
- "Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)" on page 388 (APAR OA57089)
- "TR-31 Import (CSNBT31I and CSNET31I)" on page 959 (APAR OA57089)
- "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 1278 (APAR OA57089)
- Appendix B, "Key token formats," on page 1501 (APAR OA57089)
- Appendix G, "Access control points and callable services," on page 1661 (APAR OA57089)
- Appendix H, "Impact of compliance mode on callable services," on page 1695 (APAR OA57089)
- Appendix I, "Resource names for CCA and ICSF entry points," on page 1705 (APAR OA57089)
- "ICSF Query Facility2 (CSFIQF2 and CSFIQF26)" on page 1316 (APAR OA56349)
- "PKCS #11 Derive Key (CSFPDVK and CSFPDVK6)" on page 1347 (APAR OA56349)
- "PKCS #11 Private Key Sign (CSFPPKS and CSFPPKS6)" on page 1379 (APAR OA56349)
- "PKCS #11 Public Key Verify (CSFPPKV and CSFPPKV6)" on page 1382 (APAR OA56349)
- "Diversified Key Generate (CSNBDKG and CSNEDKG)" on page 151 (APAR OA56261)
- "Key Generate (CSNBKGN and CSNEKGN)" on page 211 (APAR OA56261)
- "Key Test Extended (CSNBKYTX and CSNEKYTX)" on page 267 (APAR OA56261)

- "Clear PIN Generate (CSNBPGN and CSNEPGN)" on page 629 (APAR OA56261)
- "Encrypted PIN Translate (CSNBPTR and CSNEPTR)" on page 678 (APAR OA56261)
- "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297 (APAR OA57088)
- "Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)" on page 685 (APAR OA57088)
- Appendix G, "Access control points and callable services," on page 1661 (APAR OA57088)
- "Control Vector Generate (CSNBCVG and CSNECVG)" on page 118 (APAR OA56265)
- "Key Export (CSNBKEX and CSNEKEX)" on page 207 (APAR OA56265)
- "Key Generate (CSNBKGN and CSNEKGN)" on page 211 (APAR OA56265)
- "Key Import (CSNBKIM and CSNEKIM)" on page 237 (APAR OA56265)
- "Key Part Import (CSNBKPI and CSNEKPI)" on page 242 (APAR OA56265)
- "Key Token Build (CSNBKTB and CSNEKTB)" on page 271 (APAR OA56265)
- "Multiple Secure Key Import (CSNBSKM and CSNESKM)" on page 358 (APAR OA56265)
- "TR-31 Import (CSNBT31I and CSNET31I)" on page 959 (APAR OA56265)
- "Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)" on page 521 (APAR OA56265)
- "Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)" on page 531 (APAR OA56265)
- "Field Level Decipher (CSNBFLD and CSNEFLD)" on page 730 (APAR OA56265)
- "Field Level Encipher (CSNBFLE and CSNEFLE)" on page 739 (APAR OA56265)
- "CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)" on page 1208 (APAR OA56265)
- "DES control vector table" on page 1603 (APAR OA56265)
- "Specifying a control-vector-base value" on page 1608 (APAR OA56265)
- Appendix H, "Impact of compliance mode on callable services," on page 1695 (APAR OA56265)
- "Key Test (CSNBKYT and CSNEKYT)" on page 252
- "CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)" on page 1208
- "ICSF Multi-Purpose Service (CSFMPS and CSFMPS6)" on page 1221
- "Key Data Set List (CSFKDSL and CSFKDSL6)" on page 1225
- "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 1278
- "ICSF Query Facility2 (CSFIQF2 and CSFIQF26)" on page 1316
- "PKCS #11 Generate Secret Key (CSFPGSK and CSFPGSK6)" on page 1360
- "PKCS #11 Secret Key Decrypt (CSFPSKD and CSFPSKD6)" on page 1391
- "PKCS #11 Secret Key Encrypt (CSFPSKE and CSFPSKE6)" on page 1396
- "PKCS #11 Token Record Create (CSFPTRC and CSFPTRC6)" on page 1407
- "PKCS #11 Unwrap Key (CSFPUWK and CSFPUWK6)" on page 1417
- "PKCS #11 Wrap Key (CSFPWPK and CSFPWPK6)" on page 1422
- The following reason codes have been updated:
  - "Reason codes for return code 8 (8)" on page 1447:
    - 01F (31) (APAR OA56265)
    - 7EC (2028)
    - 96D (2413) (APAR OA57089)
    - 272C (10028) (APAR OA56265)

Deleted

No content was removed from this information.

Changes made in Cryptographic Support for z/OS V2R1 - z/OS V2R3 (FMID HCR77C1)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SC14-7508-06.

This document is for ICSF FMID HCR77C1. This release of ICSF runs on z/OS V2R1, V2R2, and V2R3 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

New

- Information about IBM z14 and IBM z14 ZR1.
- "Diversify Directed Key (CSNBDDK and CSNEDDK)" on page 170 (APAR OA55184).
- "Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)" on page 685 (APAR OA55184).
- "Compliance mode" on page 11.
- "X.509 certificates" on page 101.
- "Public Infrastructure Certificate (CSNDPIC and CSNFPIC)" on page 1184.
- "Cryptographic Usage Statistic (CSFSTAT and CSFSTAT6)" on page 1271.
- The following new reason codes have been added:
  - "Reason codes for return code 0 (0)" on page 1442
    - F9D (3997) (APAR OA54509)
  - "Reason codes for return code 4 (4)" on page 1444:
    - D7 (215) (APAR OA57088)
    - DD5 (3541)
  - "Reason codes for return code 8 (8)" on page 1447
    - 0C7 (199)
    - 309 (777) (APAR OA55184)
    - 335 (0821) - 3BF (0959)
    - 82E (2094) (APAR OA55184)
    - 83E (2110)
    - 8C7 (2247)
    - 8CE (2254)
    - 962 (2402)
    - 963 (2403)
    - 965 (2405)
    - 966 (2406)
    - 967 (2407)
    - 969 (2409)
    - 96A (2410)
    - 96D (2413)
    - 96E (2414)
    - 971 (2417)

    - 973 (2419)
    - 97A (2426)
    - 97F (2431)
    - 985 (2437)
    - 9C5 (2501) (APAR OA55184)
    - 9C6 (2502) (APAR OA55184)
    - 9C7 (2503) (APAR OA55184)
    - 9C9 (2505) (APAR OA55184)
    - 9CA (2506) (APAR OA55184)
    - 9CB (2507) (APAR OA55184)
    - 9CD (2509) (APAR OA55184)
    - BE6 (3046)
    - BF3 (3059)
    - C5B (3163) (APAR OA55184)
    - DCF (3535)
    - DD2 (3538)
    - 177F (6015)
  - "Reason codes for return code C (12)" on page 1489:
    - DD0 (3536)
    - DD1 (3537)
    - DD3 (3539)
    - DD4 (3540)
- Appendix H, "Impact of compliance mode on callable services," on page 1695.

Changed

- "Diversified Key Generate (CSNBDKG and CSNEDKG)" on page 151 (APAR OA56261).
- "Key Generate (CSNBKGN and CSNEKGN)" on page 211 (APAR OA56261).
- "Key Test Extended (CSNBKYTX and CSNEKYTX)" on page 267 (APAR OA56261).
- "Clear PIN Generate (CSNBPGN and CSNEPGN)" on page 629 (APAR OA56261).
- "Encrypted PIN Translate (CSNBPTR and CSNEPTR)" on page 678 (APAR OA56261).
- "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297 (APAR OA57088).
- "Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)" on page 685 (APAR OA57088).
- Appendix G, "Access control points and callable services," on page 1661 (APAR OA57088).
- "CCA key wrapping" on page 20 (APAR OA55184).
- "DES key types" on page 23 (APAR OA55184).
- "AES key types" on page 26 (APAR OA55184).
- "MAC Generate callable service (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)" on page 61 (APAR OA55184).
- "MAC Verify callable service (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)" on page 62 (APAR OA55184).
- "ANSI X9.143 (TR-31) key block support" on page 70 (APAR OA55184).
- "TR-31 Translate Callable Service (CSNBT31X and CSNET31X)" on page 71 (APAR OA55184).
- "TR-31 Import Callable Service (CSNBT31I and CSNET31I)" on page 70 (APAR OA55184).

- “Summary of callable services” on page 76 (APAR OA55184).
- “Key Generate (CSNBKGN and CSNEKGN)” on page 211 (APAR OA55184).
- “Key Generate2 (CSNBKGN2 and CSNEKGN2)” on page 224 (APAR OA55184).
- “Key Import (CSNBKIM and CSNEKIM)” on page 237 (APAR OA55184).
- “Key Part Import (CSNBKPI and CSNEKPI)” on page 242 (APAR OA55184).
- “Key Test (CSNBKYT and CSNEKYT)” on page 252 (APAR OA55184).
- “Key Test2 (CSNBKYT2 and CSNEKYT2)” on page 256 (APAR OA55184).
- “Key Test Extended (CSNBKYTX and CSNEKYTX)” on page 267 (APAR OA55184).
- “Key Token Build (CSNBKTB and CSNEKTB)” on page 271 (APAR OA55184).
- “Key Token Build2 (CSNBKTB2 and CSNEKTB2)” on page 297 (APAR OA55184).
- “Key Translate (CSNBKTR and CSNEKTR)” on page 343 (APAR OA55184).
- “Key Translate2 (CSNBKTR2 and CSNEKTR2)” on page 346 (APAR OA55184).
- “Multiple Secure Key Import (CSNBSKM and CSNESKM)” on page 358 (APAR OA55184).
- “Prohibit Export (CSNBPEX and CSNEPEX)” on page 384 (APAR OA55184).
- “Prohibit Export Extended (CSNBPEXX and CSNEPEXX)” on page 386 (APAR OA55184).
- “Secure Key Import (CSNBSKI and CSNESKI)” on page 408 (APAR OA55184).
- “Symmetric Key Generate (CSNDSYG and CSNFSYG)” on page 430 (APAR OA55184).
- “TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)” on page 1007 (APAR OA55184).
- “TR-31 Import (CSNBT31I and CSNET31I)” on page 959 (APAR OA55184).
- “MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)” on page 555 (APAR OA55184).
- “MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)” on page 567 (APAR OA55184).
- “Translating a PIN” on page 604 (APAR OA55184).
- “Using PINs on different systems” on page 605 (APAR OA55184).
- “PIN-encrypting keys” on page 605 (APAR OA55184).
- “ANSI X9.8 PIN restrictions” on page 606 (APAR OA55184).
- “Authentication Parameter Generate (CSNBAPG and CSNEAPG)” on page 620 (APAR OA55184).
- “Clear PIN Encrypt (CSNBCPE and CSNECPE)” on page 624 (APAR OA55184).
- “Clear PIN Generate (CSNBPGN and CSNEPGN)” on page 629 (APAR OA55184).
- “Encrypted PIN Translate (CSNBPTR and CSNEPTR)” on page 678 (APAR OA55184).
- “Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)” on page 701 (APAR OA55184).
- “DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)” on page 863 (APAR OA55184).
- “DK PIN Change (CSNBDPC and CSNEDPC)” on page 878 (APAR OA55184).
- “DK PIN Verify (CSNBDPV and CSNEDPV)” on page 892 (APAR OA55184).
- “PKA Key Generate (CSNDPKG and CSNFPKG)” on page 1131 (APAR OA55184).
- “PKA Key Translate (CSNDPKT and CSNFPKT)” on page 1169 (APAR OA55184).
- “Key Data Set List (CSFKDSL and CSFKDSL6)” on page 1225 (APAR OA55184).
- “Diversified Key Generate (CSNBDKG and CSNEDKG)” on page 151 (APAR OA54132).
- “Key Export (CSNBKEX and CSNEKEX)” on page 207 (APAR OA54132).
- “Key Generate (CSNBKGN and CSNEKGN)” on page 211 (APAR OA54132).
- “Key Import (CSNBKIM and CSNEKIM)” on page 237 (APAR OA54132).
- “Key Part Import (CSNBKPI and CSNEKPI)” on page 242 (APAR OA54132).
- “Key Test (CSNBKYT and CSNEKYT)” on page 252 (APAR OA54132).

- “Key Test2 (CSNBKYT2 and CSNEKYT2)” on page 256 (APAR OA54132).
- “Key Token Build (CSNBKTB and CSNEKTB)” on page 271 (APAR OA54132).
- “Multiple Secure Key Import (CSNBSKM and CSNESKM)” on page 358 (APAR OA54132).
- “Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)” on page 484 (APAR OA54132).
- “Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)” on page 492 (APAR OA54132).
- “MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)” on page 555 (APAR OA54132).
- “Clear PIN Encrypt (CSNBCPE and CSNECPE)” on page 624 (APAR OA54132).
- “Encrypted PIN Translate (CSNBPTR and CSNEPTR)” on page 678 (APAR OA54132).
- “VISA CVV Service Generate (CSNBCSG and CSNECSG)” on page 838 (APAR OA54132).
- “VISA CVV Service Verify (CSNBCSV and CSNECSV)” on page 843 (APAR OA54132).
- “CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2)” on page 1203 (APAR OA54132).
- “CKDS Key Record Delete (CSNBKRD and CSNEKRD)” on page 1205 (APAR OA54132).
- “CKDS Key Record Read (CSNBKRR and CSNEKRR)” on page 1207 (APAR OA54132).
- “CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)” on page 1208 (APAR OA54132).
- “CKDS Key Record Write (CSNBKRW and CSNEKRW)” on page 1213 (APAR OA54132).
- “Restrict Key Attribute (CSNBRKA and CSNERKA)” on page 402.
- “ICSF Query Facility (CSFIQF and CSFIQF6)” on page 1278.
- “Control Vector Generate (CSNBCVG and CSNECVG)” on page 118.
- “Derive ICC MK (CSNBDCM and CSNEDCM)” on page 136.
- “Derive Session Key (CSNBDSK and CSNEDSK)” on page 143.
- “Diversified Key Generate (CSNBDKG and CSNEDKG)” on page 151.
- “Generate Issuer MK (CSNBGIM and CSNEGIM)” on page 197.
- “Key Test (CSNBKYT and CSNEKYT)” on page 252.
- “Key Test2 (CSNBKYT2 and CSNEKYT2)” on page 256.
- “Key Test Extended (CSNBKYTX and CSNEKYTX)” on page 267.
- “Key Token Build (CSNBKTB and CSNEKTB)” on page 271.
- “Key Token Build2 (CSNBKTB2 and CSNEKTB2)” on page 297.
- “Key Translate2 (CSNBKTR2 and CSNEKTR2)” on page 346.
- “TR-31 Import (CSNBT31I and CSNET31I)” on page 959.
- “Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)” on page 521.
- “Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)” on page 531.
- “One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1)” on page 584.
- “Field Level Decipher (CSNBFLD and CSNEFLD)” on page 730.
- “Field Level Encipher (CSNBFLE and CSNEFLE)” on page 739.
- “Digital Signature Verify (CSNDDSV and CSNFDSV)” on page 1119.
- “CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)” on page 1208.
- “Key Data Set List (CSFKDSL and CSFKDSL6)” on page 1225.
- “ICSF Query Algorithm (CSFIQA and CSFIQA6)” on page 1273.
- “ICSF Query Facility (CSFIQF and CSFIQF6)” on page 1278.
- “ICSF Query Facility2 (CSFIQF2 and CSFIQF26)” on page 1316.
- The following reason codes have been updated:
  - “Reason codes for return code 8 (8)” on page 1447

    - 022 (34) (APAR OA55184)
    - 1BA (442) (APAR OA55184)
    - 85E (2142)
    - 86F (2159) (APAR OA55184)
    - BFB (3067)
- “DES fixed-length key token” on page 1503 (APAR OA55184).
- “Variable-length symmetric key token” on page 1509 (APAR OA55184).
- “DES control vector table” on page 1603 (APAR OA55184).
- “Specifying a control-vector-base value” on page 1608 (APAR OA55184).
- “PIN Notation” on page 1631 (APAR OA55184).
- Appendix G, “Access control points and callable services,” on page 1661 (APAR OA55184).

Deleted

No content was removed from this information.

Part 1. IBM programming

This topic introduces programming for the IBM CCA API and PKCS #11 services, including AES, DES, RSA, and ECC cryptography and explains how to use these callable services. Programming for the PKCS #11 is described in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

Chapter 1. Introducing programming for ICSF

ICSF provides access to cryptographic functions through callable services, which are also known as verbs. A callable service is a routine that receives control using a CALL statement in an application language.

Prior to invoking callable services in an application program, you must link them into the application program. See “Linking a program with the ICSF callable services” on page 12.

To invoke the callable service, the application program must include a procedure call statement that has the entry point name and parameters for the callable service. The parameters that are associated with a callable service provide the only communication between the application program and ICSF.

ICSF callable services naming conventions

The ICSF callable services generally follow the naming conventions outlined in the following table.

There are five exceptions where the CSF*zzz* names would collide and in those cases, the CSF*zzz* alias is CSFP*zzz* instead: PKDS Key Record Create (CSFPKRC), PKDS Key Record Delete (CSFPKRD), PKDS Key Record Read (CSFPKRR), PKDS Key Record Write (CSFPKRW), PKA Key Token Change (CSFPKTC).

In the following table, *zzz* is a 3- or 4-letter service name, such as ENC for the Encipher service or PKG for the PKA Key Generate service. Not all CSNBzzz/CSNEzzz services have ALET-qualified entry points (where certain parameters can be in a dataspace or an address space other than the caller's). See each specific service for details.

Table 1. ICSF Callable Services Naming Conventions

This callable service prefix:	**Identifies:**	
CSNBzzz / CSFzzz	31-bit	Symmetric Key Services and Hashing Services
CSNBzzz1 / CSFzzz1	31-bit ALET-qualified	
CSNEzzz / CSFzzz6	64-bit	
CSNEzzz1 / CSFzzz16	64-bit ALET-qualified	
CSNDzzz / CSFzzz	31-bit	Asymmetric Key Services
CSNFzzz / CSFzzz6	64-bit	
CSFPzzz	31-bit	PKCS #11 Services
CSFPzzz6	64-bit	
CSFzzz	31-bit	Utility Services and TKE Workstation Interfaces
CSFzzz6	64-bit	

Callable service syntax

This publication uses a general call format to show the name of the ICSF callable service and its parameters. An example of that format is shown here:

```
CALL CSNBxxx (return_code,
              reason_code,
              exit_data_length,
              exit_data,
              parameter_5,
              parameter_6,
              .
              .
              .
              parameter_N)
```

where CSNBxxx is the name of the callable service. The return code, reason code, exit data length, exit data, parameter 5 through parameter *N* represent the parameter list. The call generates a fixed length parameter list. You must supply the parameters in the order shown in the syntax diagrams. "Parameter definitions" on page 6 describes the parameters in more detail.

ICSF callable services can be called from application programs written in a number of high-level languages as well as assembler. The high-level languages are:

- C
- COBOL
- FORTRAN
- PL/I

The ICSF callable services comply with the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface. The services can be invoked using the generic format, **CSNBxxx**. Use the generic format if you want your application to work with more than one cryptographic product. The format **CSFxxxx** can be used in place of **CSNBxxx**. Otherwise, use the **CSFxxxx** format.

Specific formats for the languages that can invoke ICSF callable services are as follows:

- **C**

```
CSNBxxxx (return_code,reason_code,exit_data_length,exit_data,
parameter_5,…parameter_N)
```

- **COBOL**

```
CALL 'CSNBxxxx' USING return_code,reason_code,exit_data_length,
exit_data,parameter_5,…parameter_N
```

- **FORTRAN**

```
CALL CSNBxxxx (return_code,reason_code,exit_data_length,exit_data,
parameter_5,…parameter_N)
```

- **PL/I**

```
DCL CSNBxxxx ENTRY OPTIONS(ASM);
CALL CSNBxxxx return_code,reason_code,exit_data_length,exit_data,
parameter_5,…parameter_N;
```

- **Assembler** language programs must use standard linkage conventions when invoking ICSF callable services. An example of how an assembler language program can invoke a callable service is shown as follows:

```
CALL CSNBxxxx,(return_code,reason_code,exit_data_length,exit_data,
parameter_5,…parameter_N)
```

Coding examples using the high-level languages are shown in Appendix D, "Coding examples," on page 1619.

Callable services with ALET parameters

Some callable services have an alternate entry point (with ALET parameters—for data that resides in data spaces). They are in the format of *CSNBxxx1* as shown in the following table. For the associated 64-bit versions of the callable services (*CSNExxx*), the ALET-qualified versions are in the format *CSNExxx1*.

Service description	Callable service without ALET	Callable service with ALET
Cipher Text Translate2	CSNBCTT2	CSNBCTT3
Decipher	CSNBDEC	CSNBDEC1
Encipher	CSNBENC	CSNBENC1

Service description	Callable service without ALET	Callable service with ALET
HMAC Generate	CSNBHMG	CSNBHMG1
HMAC Verify	CSNBHMV	CSNBHMV1
MAC generate	CSNBMGN	CSNBMGN1
MAC generate2	CSNBMGN2	CSNBMGN3
MAC verify	CSNBMVR	CSNBMVR1
MAC verify2	CSNBMVR2	CSNBMVR3
MDC generate	CSNBMDG	CSNBMDG1
One way hash generate	CSNBOWH	CSNBOWH1
Symmetric algorithm decipher	CSNBSAD	CSNBSAD1
Symmetric algorithm encipher	CSNBSAE	CSNBSAE1
Symmetric key decipher	CSNBSYD	CSNBSYD1
Symmetric key encipher	CSNBSYE	CSNBSYE1
Symmetric MAC generate	CSNBSMG	CSNBSMG1
Symmetric MAC verify	CSNBSMV	CSNBSMV1

When choosing which service to use, consider the fact that:

- Callable services that do not have an ALET parameter require data to reside in the caller's primary address space. A program using these services adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- Callable services that have an ALET parameter allow data to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using these services does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to running with other cryptographic products that follow this programming interface.

Rules for defining parameters and attributes

These rules apply to the callable services:

- Parameters are required and positional.
- Each parameter list has a fixed number of parameters.
- Each parameter is defined as an integer or a character string. Null pointers are not acceptable for any parameter.
- Keywords that are passed to the callable services, such as CLEAR, CBC, and FIRST can be in lowercase, uppercase, or mixed case. The callable services fold them to uppercase prior to using them.

Each callable service defines its own list of parameters. The entire list must be supplied on every call. If you do not use a specific parameter, you must supply that parameter with hexadecimal zeros or binary zeros.

Parameters are passed to the callable service. All information that is exchanged between the application program and the callable service is through parameters that are passed on the call.

Each parameter definition begins with the direction that the data flows and the attributes that the parameter must possess (called “type”). This describes the direction.

Direction
Meaning

Input
The application sends (*supplies*) the parameter to the callable service. The callable service does not change the value of the parameter.

Output
The callable service *returns* the parameter to the application program. The callable service may have changed the value of the parameter on return.

Input/Output
The application sends (*supplies*) the parameter to the callable service. The callable service may have changed the value of the parameter on return.

This describes the attributes or type.

Type
Meaning

Integer (I)
A 4-byte (32-bit), twos complement, binary number that has sign significance.

String
A series of bytes where the sequence of the bytes must be maintained. Each byte can take on any bit configuration. The string consists only of data bytes. No string terminators, field-length values, or type-casting parameters are included. The maximum size of a string is X'7FFFFFFF' or 2 gigabytes. In some of the callable services, the length of some string data has an upper bound defined by the installation. The upper bound of a string can also be defined by the service.

Alphanumeric character string
A string of bytes in which each byte represents characters from this set:

Character	EBCDIC Value	Character	EBCDIC Value	Character	EBCDIC Value
A-Z		(	X'4D'	/	X'61'
a-z		)	X'5D'	,	X'6B'
0-9		+	X'4E'	%	X'6C'
Blank	X'40'	&	X'50'	?	X'6F'
*	X'5C'	.	X'4B'	:	X'7A'
<	X'4C'	;	X'5E'	=	X'7E'
>	X'6E'	-	X'60'	'	X'7D'

Parameter definitions

This topic describes these parameters, which are used by most of the callable services:

- *Return_code*
- *Reason_code*
- *Exit_data_length*
- *Exit_data*
- *Key_identifier*

Note: The *return_code* parameter, the *reason_code* parameter, the *exit_data_length* parameter, and the *exit_data* parameter are required with every callable service.

Return and reason codes

Return_code and *reason_code* parameters return integer values upon completion of the call.

Return_code
The return code parameter contains the general results of processing as an integer.

Table 2 on page 7 shows the standard return code values that the callable services return. A complete list of return codes is shown in Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441.

Table 2. Standard Return Code Values From ICSF Callable Services	
Value Hex (Decimal)	**Meaning**
00 (00)	Successful. Normal return.
04 (04)	A warning. Execution was completed with a minor, unusual event encountered.
08 (08)	An application error occurred. The callable service was stopped due to an error in the parameters. Or, another condition was encountered that needs to be investigated.
0C (12)	Error. ICSF is not active or an environment error was detected.
10 (16)	System error. The callable service was stopped due to a processing error within the software or hardware.

Generally, PCF macros will receive identical error return codes if they execute on PCF or on ICSF. A single exception has been noted: if a key is installed on the ICSF CKDS with the correct label but with the wrong key type, PCF issues a return code of 8, indicating that the key type was incorrect. ICSF issues a return code of 12, indicating that the key could not be found.

Reason_code
The reason code parameter contains the results of processing as an integer. You can specify which set of reason codes (ICSF or TSS) are returned from callable services. The default value is ICSF. For more information about the REASONCODES installation option, see *z/OS Cryptographic Services ICSF System Programmer's Guide*. Different results are assigned to unique reason code values under a return code.

A list of reason codes is shown in Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441.

Exit data length and exit data

The *exit_data_length* and *exit_data* parameters are described here. The parameters are input to all callable services. Although all services require these parameters, several services ignore them.

The parameters are for use by service exits. The description of the parameters states the intended use of the parameter, but ICSF does not enforce this usage. ICSF does not make any references to these parameters.

Exit_data_length
The integer that has the string length of the data passed to the exit. The data is identified in the *exit_data* parameter.

Exit_data
The installation exit data string that is passed to the callable service's preprocessing exit. The installation exit can use the data for its own processing.

ICSF provides two installation exits for each callable service. The preprocessing exit is invoked when an application program calls a callable service, but prior to when the callable service starts processing. For example, this exit is used to check or change parameters passed on the call or to stop the call. It can also be used to perform additional security checks.

The post-processing exit is invoked when the callable service has completed processing, but prior to when the callable service returns control to the application program. For example, this exit can be used to check and change return codes from the callable service or perform clean-up processing.

For more information about the exits, see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Key identifier for key token

A *key identifier* for a key token is an area that contains one of the following:

- Key label identifies keys that are in the CKDS or PKDS. Ask your ICSF administrator for the key labels that you can use.
- Key token can be either an internal key token, an external key token, or a null key token. Key tokens are generated by an application (for example, using the key generate callable service), or received from another system that can produce external key tokens.

  An internal key token can be used only on ICSF because the master key encrypts the key value. Internal key tokens contain keys in operational form only.

  An external key token can be exchanged with other systems because a transport key that is shared with the other system encrypts the key value. External key tokens contain keys in either exportable or importable form.

  A null key token can be used to import a key from a system that cannot produce external key tokens. A null key token can contain a key encrypted under an importer key-encrypting key but does not contain the other information present in an external key token.

  Note: X9.143 (TR-31) key blocks are supported in the same manner as CCA key tokens. Operational key block support is available with ICSF FMID HCR77D1 and later and CCA release 8.1 or later licensed internal code.
- An X.509 certificate can be used instead of a public key token in some PKA services.

The term *key identifier* is used to indicate that different inputs are possible for a parameter. One or more of the previously described items can be accepted by the callable service.

Key label

For callable services that support X9.143 (TR-31) key blocks, if the first byte of the key identifier is greater than X'5A', the field is considered to be holding a key label.

For callable services that do not support X9.143 (TR-31) key blocks, if the first byte of the key identifier is greater than X’40’, the field is considered to be holding a key label.

The contents of a key label are interpreted as a pointer to a CKDS or PKDS key entry. The key label is an indirect reference to an internal key token.

A key label is specified on callable services with the *key_identifier* parameter as a 64-byte character string, left-justified, and padded on the right with blanks. In most cases, the callable service does not check the syntax of the key label beyond the first byte. One exception is the CKDS key record create callable service, which enforces the KGUP rules for key labels unless syntax checking is bypassed by a preprocessing exit.

A key label has this form:

Table 3. Key label

Offset	Length	Data
00-63	64	Key label name

There are some general rules for creating labels for CKDS key records.

- Each label can consist of up to 64 characters. The first character must be alphabetic or a national character (#, $, @). The remaining characters can be alphanumeric, a national character (#, $, @), or a period (.).
- All alphabetic characters must be uppercase (A-Z). All labels in the key data sets are created with uppercase characters.
- Labels must be unique for all CCA key types except these DES key types in CCA key tokens: EXPORTER, IMPORTER, PINGEN, PINVER, OPINENC, and IPINENC.
- Labels must be unique for all X9.143 (TR-31) key blocks.
- Transport and PIN keys can have duplicate labels for different key types. However, keys that use the dynamic CKDS update services to create or update must have unique key labels.

- Labels must be unique for any key record, including transport and PIN keys, which are created or updated by using the dynamic CKDS update services.

Invocation requirements

Applications that use ICSF callable services must meet these invocation requirements:

- All output parameters must be in storage that the caller is allowed to modify in their execution key.
- All input parameters must be in storage that the caller is allowed to read in their execution key.
- Data can be located higher or lower than 16 MB, but must be 31-bit addressable.

  Data can be located above 2 GB if the service is invoked in AMODE(64).
- Problem or supervisor state.
- Any PSW key.
- Task mode or Service Request Block (SRB) mode.
- No mode restrictions.
- Enabled for interrupts.
- No locks held.
- Cross memory mode: Any PASN, any HASN, any SASN.
- ICSF will percolate ABENDs related to caller parameters being non-addressable. The caller should ensure that appropriate recovery is set up.

The exceptions to this list are documented with the individual callable services.

All ICSF callable services support invocation in AMODE(64). Applications which are written for AMODE(64) operation must be linked with the ICSF 64-bit service stubs, and must invoke the service with the appropriate service name. (Refer to the description of the individual callable service to determine the service name to be used.)

Security considerations

Your installation can use the Security Server RACF or an equivalent product to control who can use ICSF callable services or key labels. Prior to using an ICSF callable service or a key label, ask your security administrator to ensure that you have the necessary authorization. For more information, see *z/OS Security Server RACF Security Administrator's Guide*.

ICSF supports a key store policy using the RACF XFACILIT class. See *z/OS Security Server RACF Security Administrator's Guide*.

RACF does not control all services. The usage notes topic in the callable service description will highlight those services which are not controlled.

If program control is enabled, there are additional steps that must be performed for ICSF callable services to function. For more information, see 'Controlling the program environment' in *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Performance considerations

In most cases, the z/OS operating system dispatcher provides optimum performance. However, if your application makes extensive use of ICSF functions, you should consider using the IEAAFFN callable service (processor affinity) to avoid system overhead in selecting which processor your program (specifically, a particular TCB in the application) runs in. Note that you do **not** have to use the IEAAFFN service to ensure that the system runs a program on a processor with a cryptographic feature; the system ensures that automatically. However, you can avoid some of the system overhead involved in the selection process by using the IEAAFFN service, thus improving the program's performance. For more information on using the IEAAFFN callable service, refer to *z/OS MVS Programming: Callable Services for High-Level Languages*.

IBM recommends that you run applications first without using this option. Consider this option when you are tuning your application for performance. Use this option only if it improves the performance of your application.

Deprecated callable services

Some callable services have been marked as deprecated. These deprecated callable services are not being removed from ICSF and will continue to be supported. There are better callable services that should be used for new applications that provide the same functionality and more. The deprecated callable services will not be enhanced.

Table 4. Deprecated callable services

Deprecated callable service	New applications should use this service instead
Chapter 5, “Managing symmetric cryptographic keys,” on page 115	
“Clear Key Import (CSNBCKI and CSNECKI)” on page 116	“Multiple Clear Key Import (CSNBCKM and CSNECKM)” on page 355
“Key Translate (CSNBKTR and CSNEKTR)” on page 343	“Key Translate2 (CSNBKTR2 and CSNEKTR2)” on page 346
“Prohibit Export (CSNBPEX and CSNEPEX)” on page 384	“Restrict Key Attribute (CSNBRKA and CSNERKA)” on page 402
“Prohibit Export Extended (CSNBPEXX and CSNEPEXX)” on page 386	“Restrict Key Attribute (CSNBRKA and CSNERKA)” on page 402
“Secure Key Import (CSNBSKI and CSNESKI)” on page 408	“Multiple Secure Key Import (CSNBSKM and CSNESKM)” on page 358
Chapter 6, “Protecting data,” on page 469	
“Decode (CSNBDCO and CSNEDCO)” on page 490	“Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)” on page 521
“Encode (CSNBECO and CSNEECO)” on page 499	“Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)” on page 531
Chapter 8, “Financial services,” on page 601	
“Encrypted PIN Translate (CSNBPTR and CSNEPTR)” on page 678	“Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)” on page 685

Special secure mode

Special secure mode is a special processing mode in which:

- The Secure Key Import, Secure Key Import2, and Multiple Secure Key Import callable services, which work with clear keys, can be used.
- The Clear PIN Generate callable service, which works with clear PINs, can be used.
- The key generator utility program (KGUP) can be used to enter clear keys into the CKDS.

To use special secure mode, the following condition must be met:

- The installation options data set must specify YES for the SSM installation option or the CSF.SSM.ENABLE SAF profile must be defined in the XFACILIT SAF resource class.

  For information about specifying installation options, see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

  This is required for all systems.

Compliance mode

Beginning with the Crypto Express6 adapter, a CCA coprocessor can be configured in a compliance mode. When running in a compliance mode, the specific requirements of that mode govern how the coprocessor can be administered and used. To use compliant-tagged key tokens, at least one coprocessor must be in a compliance mode. PCI-HSM 2016 is the only supported compliance mode.

Using the callable services

This topic discusses how ICSF callable services use the different key types and key forms. It also provides suggestions on what to do if there is a problem.

ICSF provides callable services that perform cryptographic functions. You call and pass parameters to a callable service from an application program. Besides the callable services ICSF provides, you can write your own callable services called *installation-defined callable services*.

Note: Only an experienced system programmer should attempt to write an installation-defined callable service.

To write an installation-defined callable service, you must first write the callable service and link-edit it into a load module. Then define the service in the installation options data set.

You must also write a service stub. To execute an installation-defined callable service, you call a service stub from your application program. In the service stub, you specify the service number that identifies the callable service.

For more information about installation-defined callable services, see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

When the call succeeds

If the return code is **0**, ICSF has successfully completed the call. If a reason code other than 0 is included, refer to Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441, for additional information. For instance, reason code 10000 indicates the key in the key identifier (or more than one key identifier, for services that use two internal key identifiers) has been reenciphered from encipherment under the old master key to encipherment under the current master key. Keys in external tokens are not affected by this processing because they contain keys enciphered under keys other than the host master key. If you manage your key identifiers on disk, then reason code 10000 should be a "trigger" to store these updated key identifiers back on disk.

Your program can now continue providing its function, but you may want to communicate the key that you used to another enterprise. This process is exporting a key.

If you want to communicate the key that you are using to a cryptographic partner, there are several methods to use:

- For DATA keys only, call the data key export callable service. You now have a DATA key type in exportable form.
- Call the key export callable service. You now have the key type in exportable form.
- When you use the key generate callable service to create your operational or importable key form, you can create an exportable form, **at the same time**, and you now have the key type, in exportable form, at the same time as you get the operational or importable form.

When the call does not succeed

Now you have planned your use of the ICSF callable services, made the call, but the service has completed with a return and reason codes other than zero.

If the return code is **4**, there was a minor problem. For example, reason code 8004 indicates the trial MAC that was supplied does not match the message text provided.

If the return code is **8**, there was a problem with one of your parameters. Check the meaning of the reason code value, correct the parameter, and call the service again. You may go through this process several times prior to succeeding.

If the return code is **12**, ICSF is not active, has no access to cryptographic features, or has an environmental problem. Check with your ICSF administrator.

If the return code is **16**, the service has a serious problem that needs the help of your system programmer.

There are several common reason codes that can occur when you have already fully debugged and tested your program. For example:

- Reason code 10004 indicates that you provided a key identifier that holds a key enciphered under a host master key. The host master key is not installed in the cryptographic coprocessor. If this happens, you have to go back and import your importable key form again and call the service again. You need to build this flow into your program logic.
- Reason code 10012 indicates a key corresponding to the label that you specified is not in the CKDS or PKDS. Check with your ICSF administrator to see if the label is correct.
- Reason code 3063 indicates RACF failed your request to use a token.
- Reason code 16000 indicates RACF failed your request to use a service.
- Reason code 16004 indicates RACF failed your request to use the key label. Examine your CSFKEYS profiles and key store policies for possible errors.

Return and reason codes are described in Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441.

Linking a program with the ICSF callable services

For sample routines using the ICSF callable services for the following languages: C, COBOL, Assembler, and PL/I, see Appendix D, “Coding examples,” on page 1619.

ICSF provides two methods for linking ICSF callable services into an application program. Use the appropriate sample that follows.

For applications that use OS linkage (such as COBOL, High Level Assembler, or PL/I)

In the SYSLIB concatenation, include the CSF.SCSFSTUB module in the link edit step. This provides the application program access to all ICSF callable services (those that can be invoked in AMODE(24)/AMODE(31) as well as those that can be invoked in AMODE(64)).

```
//LKEDENC  JOB
//*-------------------------------------------------------------------*
//*                                                                   *
//*  The JCL links the ICSF encipher callable service, CSNBENC,       *
//*  into an application called ENCIPHER.                             *
//*                                                                   *
//*-------------------------------------------------------------------*
//LINK     EXEC PGM=IEWL,
//   PARM='XREF,LIST,LET'
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSN=CSF.SCSFSTUB,DISP=SHR        * SERVICES ARE IN HERE
//SYSLMOD  DD DSN=MYAPPL.LOAD,DISP=SHR         * MY APPLICATION LIBRARY
//SYSLIN   DD DSN=MYAPPL.ENCIPHER.OBJ,DISP=SHR * MY ENCIPHER PROGRAM
//         DD *
      ENTRY ENCIPHER
  NAME ENCIPHER(R)
/*
```

For applications written in C/C++

The following dynamic link libraries (DLLs) are linked into SYS1.SIEALNKE:

CSFDLL31
ICSF services in 31-bit addressing mode.

CSFDLL64
ICSF services in 64-bit addressing mode.

CSFDLL3X
ICSF services in 31-bit addressing mode using XPLINK.

Link with the appropriate side deck as you would with any other DLL. For example, to link with an application running in 64-bit addressing mode, include the header for the service prototypes in your C/C++ application with:

```
#include <csfbext.h>
```

and then compile normally. Finally, to link:

```
c89 -Wc,dll,lp64 -Wl,dll,lp64 -o MyApplication MyApp_main.o MyApp_support.o
/usr/lpp/pkcs11/lib/CSFDLL64.x
```

See Appendix A (SMP/E installation data sets, directories, and files) in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information about compiling and linking C/C++ applications.

Chapter 2. Introducing CCA symmetric key cryptography

This topic provides an overview of the symmetric key cryptographic functions provided in ICSF, explains the functions of the cryptographic keys, and introduces the topic of building key tokens. Many services have hardware requirements. See each service for details.

The Integrated Cryptographic Service Facility protects data from unauthorized disclosure or modification. ICSF protects data stored within a system, stored in a file off a system on magnetic tape, and sent between systems. ICSF also authenticates the identity of customers in the financial industry and authenticates messages from originator to receiver. It uses cryptography to accomplish these functions.

ICSF provides access to cryptographic functions through callable services. A callable service is a routine that receives control using a CALL statement in an application language. Each callable service performs one or more cryptographic functions, including:

- Generating and managing cryptographic keys.
- Enciphering and deciphering data with encrypted keys using the U.S. National Institute of Standards and Technology (NIST) Data Encryption Standard (DES) or Advanced Encryption Standard (AES).
- Enciphering and deciphering data with clear keys using either the NIST Data Encryption Standard (DES), or Advanced Encryption Standard (AES).
- Reenciphering text from encryption under one key to encryption under another key.
- Generating random numbers.
- Ensuring data integrity and verifying message authentication.
- Generating, verifying, and translating personal identification numbers (PINs) that identify a customer on a financial system.
- Generating and verifying payment card security codes and other payment card processing.

The German Banking Industry Committee (Deutsche Kreditwirtschaft (DK)) designed methods of creating, processing, and verifying PINs for its members. The methods use a PIN reference value (PRW) which is generated when a PIN is created or changed and used to verify the PIN supplied in a transaction. The methods are not dependent on a specific cryptographic algorithm, but DK has chosen the AES algorithm for its implementation. See “AES key types” on page 26 for more information about the AES key types added for the DK PIN methods.

Note: Additional triple-length DES key support is introduced by SPE OA55184 for ICSF FMID HCR77C1 and later releases and licensed internal code for IBM z13, IBM z13s, IBM z14, and later servers. In general, any service where a double-length key can be used, a triple-length key can be used as well. The service description should be checked for any restrictions.

Functions of symmetric cryptographic keys

ICSF provides functions to create, import, and export AES, DES, and HMAC keys. This topic gives an overview of these cryptographic keys. Detailed information about how ICSF organizes and protects keys is in *z/OS Cryptographic Services ICSF Administrator's Guide*.

ICSF supports two formats of symmetric key tokens: fixed-length and variable-length. In fixed-length format key tokens, key type and usage are defined by the control vector. In variable-length format key tokens, the key type and usage are defined in the associated data section. The control vector and associated data section are cryptographically bound to the encrypted key value in the token.

ICSF supports X9.14 (TR-31) key blocks. The key usage, algorithm, and mode of use are defined in the block header. The block header is cryptographically bound to the key block. Support for external key blocks with a key context field of '0' is available on all servers and releases of ICSF. Support for external

key blocks with a key context field of '2' and operational (internal) key blocks is available on z16 and later servers with a CEX8 or later coprocessor and CCA release 8.1 or later licensed internal code.

Key separation

The cryptographic feature controls the use of keys by separating them into unique types, allowing you to use a specific type of key only for its intended purpose. For example, a key used to protect data cannot be used to protect a key.

An ICSF system has one DES master key and one AES master key. To provide for key separation for fixed-length tokens, the cryptographic feature automatically encrypts each type of key in a fixed-length token under a unique variation of the master key. Each variation of the master key encrypts a different type of key. Although you enter only one master key, you have a unique master key to encrypt all other keys of a certain type.

Note: The enhanced wrapping method version 3 does not use a variant.

Key separation for variable-length tokens is provided by the associated data (key usage and key management fields). When the key is encrypted, the associated data is cryptographically bound to the key.

CCA DES control vectors

Control vectors contains key type, key management, and key usage attributes. For each type of DES key that the master key enciphers, there is a unique control vector. The control vector ensures that an operational key can only be used in cryptographic functions for which it is intended. For example, the control vector for an input PIN-encrypting key ensures that such a key can be used only in the PIN translation and PIN verification functions. “DES key types” on page 23 describes the different DES key types.

For wrapping methods WRAP-ECB, WRAP-ENH, and WRAPENH2, the control vector is cryptographically bound to the key during the wrapping of the key. The length of the key is in the control vector in the key form bits (bits 40-42). For double-length keys, the left and right control vectors have different key form bits. For triple-length keys, the two control vectors are the same.

For the WRAPENH3 method, only one control vector is stored in the key token. The control vector is not used in the wrapping of the key. The key form bits are not used to determine the length of the key. The control vector is cryptographically bound to the key by the TDES-CMAC of the key token by a MAC key derived when the key is wrapped.

Key forms

A key that is protected under the master key is in *operational form*, which means ICSF can use it in cryptographic functions on the system.

When you store a key with a file or send it to another system, the key is enciphered under a transport key rather than the master key because, for security reasons, the key should no longer be active on the system. When ICSF enciphers a key under a transport key, the key is not in operational form and cannot be used to perform cryptographic functions.

When a key is enciphered under a transport key, the sending system considers the key in *exportable form*. The receiving system considers the key in *importable form*. When a key is reenciphered from under a transport key to under a system's master key, it is in operational form again.

Enciphered keys appear in three forms. The form you need depends on how and when you use a key.

- **Operational** key form is used at the local system. Many callable services can *use* an operational key form.

  The Key Token Build, Key Token Build2, Key Generate, Key Generate2, Key Import, Data Key Import, Clear Key Import, Multiple Clear Key Import, Secure Key Import, Secure Key Import2, Multiple Secure Key Import, Symmetric Key Import, Symmetric Key Import2, and TR-31 Import callable services can *create* an operational key form.

- **Exportable** key form is transported to another cryptographic system. It can only be passed to another system. The ICSF callable services cannot use it for cryptographic functions. The Key Generate, Key Generate2, Data Key Export, and Symmetric Key Export callable services produce the exportable key form.
- **Importable** key form can be transformed into operational form on the local system. Key Import, Data Key Import, and Symmetric Key Import2 callable services can *use* an importable key form. Only the Key Generate callable service can *create* an importable key form. Multiple Secure Key Import and Secure Key Import2 callable services can convert a clear key into an importable key form.

For more information about the key types, see either "Functions of symmetric cryptographic keys" on page 15 or the *z/OS Cryptographic Services ICSF Administrator's Guide*. See "Key forms and types used in the Key Generate callable service" on page 73 for more information about key form.

Key flow

The conversion from one key to another key is considered to be a one-way flow. An operational key form cannot be turned back into an importable key form. An exportable key form cannot be turned back into an operational or importable key form. The flow of ICSF key forms can only be in one direction:

```
IMPORTABLE —to→ OPERATIONAL —to→ EXPORTABLE
```

Key token

ICSF supports internal and external CCA key tokens and X9.143 (TR-31) key blocks. Support for operational (internal) TR-31 key blocks is available on IBM z16 and later servers with a CEX8 or later coprocessor and CCA release 8.1 or later licensed internal code. Operational TR-31 key blocks can be stored in the CKDS using the large common record (KDSRL) format of the CKDS.

Note: Support for the KDSRL format requires z/OS V2R5 ICSF (FMID HCR77D2).

For the purpose of this topic, key token refers to both CCA key tokens and TR-31 key blocks. See "X9.143 (TR-31) key blocks" on page 28 for more details on support for operational key blocks.

ICSF supports two formats of CCA symmetric key tokens: fixed-length and variable-length. The fixed-length format token is a 64-byte field composed of a key value and control information in the control vector. The variable-length format token is composed of a key value and control information in the associated data section of the token. The control information is assigned to the key when ICSF creates the key.

The TR-31 key block format defines a header section. The header contains metadata about the key, including its usage attributes. The header can also be extended with optional blocks, which can either have standardized content or proprietary information. Callable services are also provided for retrieving standard header or optional block information from a TR-31 key block without importing the key and for building an optional block. For the layout of the block header, see "X9.143 (TR-31) key block header and optional block data" on page 1538.

The key token can be either an internal key token, an external key token, or a null key token. Through the use of key tokens, ICSF can:

- Support continuous operation across a master key change.
- Control use of keys in cryptographic services.

When the first byte of the key identifier is X'01', the key identifier is interpreted as an **operational CCA key token.** When the key context (offset 14) of a TR-31 key block is an ASCII '1', the key identifier is interpreted as an **operational TR-31 key block**. An internal key token is a token that can be used only on the ICSF system that created it (or another ICSF system with the same host master key). It contains a key that is encrypted under the master key.

An application obtains an internal key token by using one of the callable services such as those listed here. The callable services are described in detail in Chapter 5, "Managing symmetric cryptographic keys," on page 115.

- CKDS Key Record Read and CKDS Key Record Read2.
- Clear Key Import and Multiple Clear Key Import.
- Data Key Import.
- Key Generate and Key Generate2.
- Key Import.
- Key Part Import and Key Part Import2.
- Key Token Build and Key Token Build2.
- Multiple Secure Key Import and Secure Key Import2.
- Symmetric Key Import2.
- TR-31 Import.

The master keys may be dynamically changed between the time that you invoke a service, such as the key import callable service to obtain a key token, and the time that you pass the key token to the encipher callable service. When a change to the master key occurs, ICSF reenciphers the caller's key from under the old master key to under the new master key. A return code of 0 with a reason code of 10000 notifies you that ICSF reenciphered the key. For information on reenciphering the CKDS or the PKDS, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Note: If an internal key token held in user storage is not used while the master key is changed twice, the internal key token is no longer usable. See "Other considerations" on page 23 for additional information.

For debugging information, see Appendix B, "Key token formats," on page 1501 for the format of an internal key token.

When the first byte of the key identifier is X'02', the key identifier is interpreted as an external key token. When the key context (offset 14) of a TR-31 key block is an ASCII '0' or '2', the key identifier is interpreted as an **external key block**. By using the external key token, you can exchange keys between systems. It contains a key that is encrypted under a key-encrypting key.

An external key token contains an encrypted key and control information to allow compatible cryptographic systems to:

- Have a standard method of exchanging keys.
- Control the use of keys through the control vector.
- Merge the key with other information needed to use the key.

An application obtains the external key token by using one of the callable services such as:

- Key Generate and Key Generate2.
- Key Export.
- Data Key Export.
- Symmetric Key Export and Symmetric Key Export with Data.

For debugging information, see Appendix B, "Key token formats," on page 1501 for the format of an external key token.

If the first byte of the key identifier is X'00', the key identifier is interpreted as a null key token. Use the null key token to import a DES key from a system that cannot produce external key tokens into a fixed-length format token. That is, if you have an 8-byte to 16-byte key that has been encrypted under an importer key, but is not imbedded within a token, place the encrypted key in a null key token and then invoke the key import callable service to get the key in operational form. For debugging information, see Appendix B, "Key token formats," on page 1501 for the format of a null key token.

ICSF supports the TR-31 key block for communication with systems that do not use CCA key tokens. The TR-31 Translate callable service is used to export CCA operational tokens to external TR-31 key blocks. The TR-31 Import callable service is used to import external TR-31 key blocks to CCA operational tokens.

Compliant-tagged key tokens

A compliant-tagged key token must adhere to the requirements of a compliance mode. A coprocessor in compliance mode must be available to use compliant-tagged key tokens. Additional details can be found in Appendix H, “Impact of compliance mode on callable services,” on page 1695.

CCA DES key token

The compliant-tag is indicated by bit 58 in the control vector of the key token. For more information, see Appendix C, “Changing control vectors with the CVT callable service,” on page 1603.

To generate a compliant-tagged key token, a skeleton token must first be built with the compliant-tag bit on. The Control Vector Generate (CSNBCVG/CSNECVG) and Key Token Build (CSNBKTB/CSNEKTB) services allow that. This skeleton token can then be passed to any callable service that generates key tokens and supports compliant-tagged key tokens; for example, Key Generate (CSNBKGN/CSNEKGN). A list of services that support compliant-tagged key tokens can be found in Appendix H, “Impact of compliance mode on callable services,” on page 1695.

The TR-31 Import (CSNBT31I/CSNET31I) and EMV services are exceptions because they do not support skeleton tokens as input. When importing a TR-31 key block, if the key encrypting key is compliant-tagged, the resulting key is also compliant-tagged. Regarding the EMV services, to generate a compliant-tagged issuer master key, you must specify the COMP-TAG keyword with the Generate Issuer MK (CSNBGIM/CSNEGIM) callable service. Subsequent key tokens are generated based on the deriving key. For instance, if the issuer master key is compliant-tagged, the Derive ICC MK (CSNBDCM/CSNEDCM) callable service generates a compliant-tagged ICC MK. The same applies to the derivation of the session key by the Derive Session Key (CSNBDSK/CSNEDSK) callable service. For more information, see the description of the individual services.

Compliant-tagged key tokens are defined by two features of the key token:

1. The compliant-tag bit in the key attributes, which is the CV for DES fixed-length key tokens.
2. The key derivation function (KDF) value which indicates what generation of compliance is applicable.

The KDF value appears at the end of the truncated Master Key Verification Pattern (MKVP) section. For additional information, see Appendix B, “Key token formats,” on page 1501.

Key tokens with the compliant-tag bit on in the control vector and a key derivation function (KDF) of less than X'03' are no longer considered compliant-tagged. They are referred to as DES KDF 01 or 02 tokens throughout the ICSF publications. They were either created on a Crypto Express adapter which does not have the May 2021 or later licensed internal code or by the Diversified Key Generate (CSNBDKG and CSNEDKG) or Unique Key Derive (CSNBUKD and CSNEUKD) services using an input DES KDF 01 or 02 token. A DES KDF 01 key token can only be used with other DES KDF 01 tokens. Beginning with the May 2021 or later licensed internal code (LIC), a DES KDF 02 token can only be used with other DES KDF 02 key tokens. The only exception to this is a DES KDF 01 or 02 key token can also be used with an X.509 certificate or with any other key token in the Cipher Text Translate2 (CSNBCTT2 and CSNECTT2) callable service. It is recommended that DES KDF 01 and 02 tokens be migrated using the Key Translate2 (CSNBKTR2 ad CSNEKTR2) service with the COMP-TAG keyword. The output will be a key token with the compliant-tag bit set in the control vector and a KDF of X'03' or greater. Only DES tokens with a KDF of X'03' or greater are referred to as compliant-tagged key tokens. Key tokens without the compliant-tag bit set or DES KDF 01 or 02 tokens are referred to as non-compliant-tagged tokens and this is reflected in ICSF output (ICSF displays as well as SMF records do not show DES KDF 01 or 02 tokens as compliant-tagged). The CSFCMPLC, CSFCMPCC, and CSFCMPTC samples may help with identifying DES KDF 01 or 02 tokens and migrating them. Though DES KDF 01 or 02 tokens are not compliant-tagged, a coprocessor in compliance mode is required to use them.

CCA AES key token

Only version 05 AES key tokens may become compliant-tagged. The compliant-tag is indicated by a key management flag in the key token. For more information about the compliant-tag, see Appendix B, “Key token formats,” on page 1501.

To generate a compliant-tagged key token, the Key Token Build2 (CSNBKTB2/CSNEKTB2) must first be used to build a skeleton token with the compliant-tag flag on. This skeleton token can then be passed to any callable service that generates version 05 AES key tokens and supports compliant-tagged key tokens, for example Key Generate2 (CSNBKGN2/CSNEKGN2). A list of services that support compliant-tagged key tokens can be found in Appendix H, “Impact of compliance mode on callable services,” on page 1695.

The TR-31 Import (CSNBT31I/CSNET31I) and Diversify Directed Key (CSNBDDK/CSNEDDK) services are exceptions because they do not support skeleton tokens as input. When importing a TR-31 key block, if the key encrypting key is compliant-tagged, the resulting key is also compliant-tagged. When creating a directed key, if the diversifying key is compliant-tagged, the resulting key is also compliant-tagged. For more information, see the description of the individual services.

TR-31 key blocks

The compliant-tag is indicated by a general flag in the IBM proprietary option block (block ID '10') for internal controls. For more information, see “X9.143 (TR-31) key block header and optional block data” on page 1538.

To generate an AES or DES compliant-tagged key block, the TR-31 Create (CSNBT31C/CSNET31C) service is called with the COMP-TAG rule array keyword. A list of services that support compliant-tagged key tokens can be found in Appendix H, “Impact of compliance mode on callable services,” on page 1695.

CCA key wrapping

ICSF supports several methods of wrapping the key value in the CCA key tokens.

All variable-length tokens use the AESKW wrapping method defined in ANSI X9.102.

The key value in AES tokens is always encrypted using AES encryption and cipher block chaining (CBC) mode.

The key value in DES fixed-length tokens may be wrapped:

WRAP-ECB
The original method which has been used by ICSF since it was first released. For this method, the key value is encrypted using triple DES encryption and key parts are encrypted separately.

WRAP-ENH
The enhanced wrapping method version 1 uses a NIST SHA-1 key derivation function. The wrapping key is derived using a NIST counter-mode key derivation function. SHA-1 HMAC is used in the KDF. For this method, the key value is bundled with other token data and encrypted using triple DES encryption and cipher block chaining mode.

WRAPENH2
The enhanced wrapping method version 2 is the same as version 1, but uses SHA-256 in the key derivation function. The key value is bundled with other token data and encrypted using triple DES encryption and cipher block chaining mode. This method applies only to triple-length key tokens, with the exception of DATA keys with a zero control vector. The SHA-256 enhanced method is available on IBM z13 or IBM z13s servers with the July 2019 licensed internal code or the IBM z14 or later servers with the December 2018 licensed internal code.

WRAPENH3
The enhanced wrapping method version 3 is like version 2. This method uses SHA-256 in the key derivation function with the addition of an authentication code. The control vector is not bundled with the key material when it is encrypted. The wrapping and MAC keys are derived using the NIST KDF with SHA-256 HMAC. A TDES-CMAC authentication code is generated over the complete key token. The authentication code is stored in the token where the right control vector was stored. There will always be three key parts encrypted and placed in the key token to obfuscate the key length. All keys with the exception of DATA keys with a zero control vector can be wrapped with this method. The method is available on IBM z13 or IBM z13s servers with the May 2021 licensed internal code (LIC), on IBM z14 or IBM z14 ZR1 servers with the May 2021 licensed internal code (LIC), or on IBM z15 or later hardware with the May 2021 or later licensed internal code (LIC).

Your installation's system programmer can, while customizing installation options data set as described in the *z/OS Cryptographic Services ICSF System Programmer's Guide*, use the DEFAULTWRAP parameter to specify the default key wrapping for symmetric keys. Application programs can override this default method using the rule array keywords for some callable services.

Installation applications which override the wrapping method with a rule array keyword can change the wrapping method from WRAP-ENH to WRAPENH3 with an XFACILIT class profile. For additional information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Note: All variable-length tokens are wrapped using the AESKW wrapping method defined in ANSI X9.102 and are not affected by the DEFAULTWRAP setting.

Payload format

Variable-length tokens have a payload section that contains the encrypted key material. Prior to FMID HCR77A1, these tokens used a variable-length payload which consisted of the encrypted key and padding. FMID HCR77A1 introduces fixed-length payloads for AES keys which will obscure the length of the encrypted key in the payload section

Any new key types will have the fixed-length payload format. Existing AES key types (CIPHER, IMPORTER and EXPORTER) and HMAC key types will default to use the variable-length payloads unless keywords indicate the use of the fixed-length payloads. This ensures compatibility with older releases of ICSF and hardware where fixed-length payloads are not supported.

The following options are available for AES CIPHER, IMPORTER and EXPORTER keys:

- New tokens can be created with fixed-length or variable-length payloads by providing new keywords to Key Token Build2 or Key Generate2.
- Existing tokens can be translated to fixed-length or variable-length payload format by providing new keywords to Key Translate2. To convert to a variable-length payload, the Key Translate2 - Translate fixed to variable payload access control point must be enabled.
- Clear keys can be encrypted into tokens with the fixed-length or variable-length payload format by providing new keywords to Secure Key Import2.

The CKDS Reencipher utility and the Key Part Import2, Key Test2, Restrict Key Attribute, Symmetric Key Export, and Symmetric Key Import2 callable services will maintain the payload format of the source key token.

Fixed-length payload support requires an IBM zEnterprise EC12, zEnterprise BC12, or later with a CCA Cryptographic coprocessor that is a CEX3C or later with Licensed Internal Code (LIC) of September 2013 or later.

Types of keys

The cryptographic keys are grouped into these classes based on the functions they perform.

If you intend to use a key for an extended period, you can store it in the CKDS so that it will be reenciphered if the master key is changed.

Master keys - DES and AES

Master keys are used to encipher operational keys. The ICSF administrator installs and changes the master keys (see *z/OS Cryptographic Services ICSF Administrator's Guide* for details). The administrator does this by using the Master Key Entry panels or the optional Trusted Key Entry (TKE) workstation.

The master key always remains in a secure area in the cryptographic coprocessors.

The master keys are used only to encipher and decipher keys. Other keys also encipher and decipher keys and are mostly used to protect cryptographic keys you transmit on external links. These keys, while on the system, are also encrypted under the master keys.

Data-encrypting keys

Data-encrypting keys are used to protect data privacy. There are two classes of data-encrypting keys:

- DATA keys can be either encrypted under the master key or in the clear (See “Clear keys” on page 28 for details on using clear keys). DATA key can be used to encrypt data and generate MACs.
- CIPHER keys are encrypted under the master key. CIPHER keys can only be used to encrypt data.

Cipher text translation keys

Cipher text translation keys protect data that is transmitted through intermediate systems when the originator and receiver do not share a common key. Data that is enciphered under one cipher text translation key is reenciphered under another cipher text translation key on the intermediate node. During this process, the data never appears in the clear.

MAC keys

Message authentication is the process of verifying the integrity of transmitted messages. Message authentication code (MAC) processing enables you to verify that a message has not been altered. You can use a MAC to check that a message you receive is the same one the message originator sent. The message itself may be in clear or encrypted form.

MAC keys can be used to generate and verify MACs or can be restricted to just verify MACs.

DES supports the ANSI X9.9-1 procedure, ANSI X9.19 optional double key MAC procedure, and EMV Specification and ISO 16609 for encrypted keys.

DES MAC keys can be used to generate CVVs and CSCs for PIN transactions.

AES supports ciphered message authentication code (CMAC) for encrypted keys and CBC-MAC and XCBC-MAC for clear keys.

HMAC supports FIPS-198 hashed message authentication code (HMAC) for encrypted keys.

PIN keys

Personal authentication is the process of validating personal identities in a financial transaction system. The personal identification number (PIN) is the basis for verifying the identity of a customer across the financial industry networks. A PIN is a number that the bank customer enters into an automatic teller machine (ATM) to identify and validate a request for an ATM service.

You can use ICSF to generate PINs and PIN offsets. A PIN offset is a value that is the difference between two PINs. For example, a PIN offset may be the difference between a PIN that is chosen by the customer and one that is assigned by an institution. You can use ICSF to verify the PIN that was generated by ICSF. You can also use ICSF to protect PIN blocks that are sent between systems and to translate PIN blocks from one format to another. A PIN block contains a PIN and non-PIN data. You use PIN keys to generate and verify PINs and PIN offsets and to protect and translate PIN blocks.

“Managing personal authentication” on page 64 gives an overview of the PIN algorithms you need to know to write your own application programs.

Key-encrypting keys

Key-encrypting keys protect a key that is sent to another system, received from another system, or stored with data in a file. A variation of transport keys are also used to rewrap a key from one key-encrypting key to another key-encrypting key.

Key-encrypting keys are always generated in pairs. Both keys have the same clear key value, but have a different encrypted key value due to the control vector or the associated data.

Exporter key-encrypting key

An exporter key-encrypting key protects keys that are sent from your system to another system. The exporter key at the originator has the same clear value as the importer key at the receiver. An exporter key is paired with an importer key-encrypting key.

DES OKEYXLAT keys must be used when rewrapping a key under a key-encrypting key. The AES EXPORTER must have the TRANSLAT key usage enabled when rewrapping a key.

Importer key-encrypting key

An importer key-encrypting key protects keys that are sent from another system to your system. It also protects keys that you store externally in a file that you can import to your system later. The importer key at the receiver has the same clear value as the exporter key at the originator. An importer key is paired with an exporter key-encrypting key.

DES IKEYXLAT keys must be used when rewrapping a key under a key-encrypting key. The AES IMPORTER must have the TRANSLAT key usage enabled when rewrapping a key.

Note:

- Key-encrypting keys replace local, remote, and cross keys used by PCF.
- A key-encrypting key should be as strong or stronger than the key it is wrapping.

DES NOCV key-encrypting keys
DES NOCV importers and exporters are key-encrypting keys used to transport keys with systems that do not recognize CCA control vectors. There are some requirements and restrictions for the use of NOCV key-encrypting keys:

- Use of NOCV IMPORTERs and EXPORTERs is controlled by access control points.
- Only programs in system or supervisor state can use the NOCV key-encrypting key in the form of tokens in callable services. Any problem program may use NOCV key-encrypting key with label names from the CKDS.
- NOCV key-encrypting key on the CKDS should be protected by RACF.
- NOCV key-encrypting key can be used to encrypt single, double, or triple length keys for any key type.

Key-generating keys
Key-generating keys are used to derive unique-key-per transaction keys.

Cryptographic-variable keys
These DES keys are used to encrypt special control values in DES key management.

Secure messaging keys
Secure messaging keys used to encrypt keys and PINs for incorporation into a text block. The text block is then encrypted to preserve the security of the key value. The encrypted text block, normally the value field in a TLV item, can be incorporated into a message sent to an EMV smart card.

Other considerations

These are considerations for keys held in the cryptographic key data set (CKDS) or by applications.

- ICSF ensures that keys held in the CKDS are reenciphered during the master key change. Keys with a long life span (more than one master key change) should be stored in the CKDS.
- Keys enciphered under the host DES master key and held by applications are automatically reenciphered under a new master key as they are used. Keys with a short life span (for example, VTAM SLE data keys) do not need to be stored in the CKDS. However, if you have keys with a long life span and you do not store them in the CKDS, they should be enciphered under the importer key-encrypting key. The importer key-encrypting key itself should be stored in the CKDS.

DES key types

The DES keys are 64-bit, 128-bit, and 192-bit keys that use the DES algorithm to perform the cryptographic function. A 64-bit key is referred to as a single-length key. A 128-bit key is referred to as a double-length key. Triple-length keys are 192-bits in length.

Additional triple-length DES key support is introduced by APAR OA55184 for ICSF FMID HCR77C1 and later releases and licensed internal code for IBM z13, IBM z13s, IBM z14, and later servers. In general, any service where a double-length key can be used, a triple-length key can be used as well. The service description should be checked for any restrictions.

For installations that do not support double-length key-encrypting keys, effective single-length keys are provided. For an effective single-length key, the clear key value of the left key half equals the clear key value of the right key half.

Table 5. Descriptions of DES key types and service usage	
DES key type	**Usable with services**
DATA class (data operation keys) These key are used to encrypt and decrypt data. Single-length keys can be used to generate and verify MACs and CVVs. DATA keys can be single-length, double-length, or triple-length. DATAM and DATAMV keys are double-length. CLRDES keys are DATA keys.	
DATA	Authentication Parameter Generate, Cipher Text Translate2, CVV Key Combine, Decipher, Encipher, EMV Verification Functions, Field Level Decipher, Field Level Encipher, MAC Generate, MAC Verify, Symmetric Key Encipher, Symmetric Key Decipher, VISA CVV Generate, VISA CVV Verify
DATAM	MAC Generate, MAC Verify
DATAMV	MAC Verify
Cipher class (data operation keys) These key are used to encrypt and decrypt data. The keys can be single-length, double-length, or triple-length.	
CIPHER	Cipher Text Translate2, Decipher, Encipher, Encrypted PIN Translate Enhanced, FPE Decipher, FPE Encipher, FPE Translate
DECIPHER	Cipher Text Translate2, Decipher, Encrypted PIN Translate Enhanced, FPE Decipher, FPE Translate
ENCIPHER	Cipher Text Translate2, Encipher, FPE Encipher, FPE Translate
CIPHERXL class (cipher text translate keys) These key are used to translate cipher text. The keys are double-length.	
CIPHERXI	Cipher Text Translate2 (translate inbound key only)
CIPHERXL	Cipher Text Translate2 (translate inbound and outbound key)
CIPHERXO	Cipher Text Translate2 (translate outbound key only)
MAC class (data operation keys) These keys are used to generate and verify MACs, CVVs, and CSCs. The keys can be single-length, double-length, or triple-length.	
MAC	CVV Key Combine, MAC Generate, MAC Verify, Transaction Validation, VISA CVV Generate, VISA CVV Verify
MACVER	CVV Key Combine, MAC Verify, Transaction Validation, VISA CVV Verify
PIN class These keys are used generate and verify PINs and PIN offsets. The keys can be double-length or triple-length.	
PINGEN	Clear PIN Generate, Clear PIN Generate Alternate, Encrypted PIN Generate, Recover PIN from Offset
PINVER	Encrypted PIN Verify

Table 5. Descriptions of DES key types and service usage (continued)	
DES key type	**Usable with services**
These keys are used wrap and unwrap PIN blocks: The keys can be double-length or triple-length.	
IPINENC	Authentication Parameter Generate, Clear PIN Generate Alternate, EMV Scripting Service, Encrypted PIN Translate, Encrypted PIN Translate2, Encrypted PIN Translate Enhanced, Encrypted PIN Verify, Encrypted PIN Verify2, PIN Change/Unblock, Secure Messaging for PINs
OPINENC	Clear PIN Encrypt, Clear PIN Generate Alternate, EMV Scripting Service, Encrypted PIN Generate, Encrypted PIN Translate, Encrypted PIN Translate2, Encrypted PIN Translate Enhanced, PIN Change/Unblock, Recover PIN from Offset
Key-encrypting key class These keys are used to wrap other keys. EXPORTER, IMPORTER, and IMP-PKA keys can be double-length or triple-length. The other key types are double-length keys.	
EXPORTER	Control Vector Translate, Data Key Export, Derive ICC MK, ECC Diffie-Hellman, Generate Issuer MK, Key Encryption Translate, Key Export, Key Generate, Key Test2, Key Test Extended, Key Translate, Key Translate2, PKA Key Generate, PKA Key Translate, Prohibit Export Extended, Remote Key Export, Secure Messaging for Keys, Symmetric Key Generate, TR-31 Translate, TR-31 Import, TR-34 Key Distribution, Unique Key Derive
IMPORTER	Control Vector Translate, Data Key Import, ECC Diffie-Hellman, Generate Issuer MK, Key Encryption Translate, Key Generate, Key Import, Key Test2, Key Test Extended, Key Translate, Key Translate2, Multiple Secure Key Import, PKA Key Generate, PKA Key Import, PKA Key Translate, Prohibit Export Extended, Remote Key Export, Restrict Key Attribute, Secure Key Import, Secure Messaging for Keys, Symmetric Key Generate, TR-31 Translate, TR-31 Import
IMP-PKA	PKA Key Import, Remote Key Export, Trusted Block Create
IKEYXLAT, OKEYXLAT	Control Vector Translate, Key Translate, Key Translate2, TR-31 Translate,TR-31 Import
Key-generate key class These keys are used to derive keys. The keys are double-length keys. The key usage flags in the control vector determine which services the KEYGENKY key may be used with.	
KEYGENKY	Diversified Key Generate, Encrypted PIN Translate, Encrypted PIN Translate2, Encrypted PIN Translate Enhanced, Encrypted PIN Verify, Encrypted PIN Verify2, FPE Decipher, FPE Encipher, FPE Translate, Unique Key Derive
DKYGENKY	Derive ICC MK, Derive Session Key, Diversified Key Generate, EMV Scripting Service, EMV Transaction (ARQC/ARPC) Service, EMV Verification Functions, Generate Issuer MK, PIN Change/Unblock

Table 5. Descriptions of DES key types and service usage (continued)	
DES key type	**Usable with services**
Cryptographic-variable class These keys are used in the special verbs that operate with cryptographic variables The keys are single-length keys.	
CVARENC	Cryptographic Variable Encipher
CVARXCVL	Control Vector Translate
CVARXCVR	Control Vector Translate
Secure-messaging class (data operation keys) These keys are used to encrypt keys or PINs. The keys are double-length keys. The key usage flags in the control vector determine which services the key may be used with.	
SECMSG	Diversified Key Generate, Secure Messaging for Keys, Secure Messaging for PINs

AES key types

The AES keys are 128-bit, 192-bit, and 256-bit keys that use the AES algorithm to perform the cryptographic function.

Table 6. Descriptions of AES key types and service usage	
AES key type	**Usable with services**
Fixed-length AES key-token, version X'04'	
DATA	Field Level Decipher, Field Level Encipher, Symmetric Algorithm Decipher, Symmetric Algorithm Encipher
Variable-length AES key-token, version X'05'	
Cipher class (data operation keys) These keys are used to cipher text.	
CIPHER	Cipher Text Translate2, Format Preserving Algorithms Decipher, Format Preserving Algorithms Encipher, Format Preserving Algorithms Translate, Symmetric Algorithm Decipher, Symmetric Algorithm Encipher, Symmetric Key Decipher, Symmetric Key Encipher
Key-encrypting key class These keys are used to cipher other keys.	
EXPORTER	Key Generate2, Key Translate2, PKA Key Generate, Symmetric Key Export, TR-31 Translate, TR-31 Import, TR-34 Key Distribution
IMPORTER	Key Generate2, PKA Key Generate, Key Test2, Key Translate2, Restrict Key Attribute, Secure Key Import2, Symmetric Key Import2, TR-31 Translate, TR-31 Import
MAC class These keys are used to generate and verify a message authentication code (MAC).	

Table 6. Descriptions of AES key types and service usage (continued)	
AES key type	**Usable with services**
MAC	DK Deterministic PIN Generate, DK Migrate PIN, DK PIN Change, DK PAN Modify in Transaction, DK PAN Translate, DK PRW Card Number Update, DK PRW Card Number Update2, DK PRW CMAC Generate, DK Random PIN Generate, DK Random PIN Generate2, DK Regenerate PRW, MAC Generate2, MAC Verify2
PIN class These keys are used in various financial-PIN processing services.	
PINCALC	DK Deterministic PIN Generate
PINPROT	Clear PIN Encrypt, Clear PIN Generate Alternate, DK Deterministic PIN Generate, DK Migrate PIN, DK PAN Modify in Transaction, DK PAN Translate, DK PIN Change, DK PIN Verify, DK PRW Card Number Update, DK PRW Card Number Update2, DK Random PIN Generate, DK Random PIN Generate2, DK Regenerate PRW, Encrypted PIN Generate, Encrypted PIN Translate2, Encrypted PIN Verify, Encrypted PIN Verify2, PIN Change/Unblock, Recover PIN from Offset, Secure Messaging for PINs
PINPRW	DK Deterministic PIN Generate, DK Migrate PIN, DK PAN Modify in Transaction, DK PAN Translate, DK PIN Change, DK PIN Verify, DK PRW Card Number Update, DK PRW Card Number Update2, DK Random PIN Generate, DK Random PIN Generate2, DK Regenerate PRW
Key generating class These keys are used to derive operational keys.	
DKYGENKY	Diversified Key Generate2, Encrypted PIN Translate2, Encrypted PIN Translate Enhanced, Encrypted PIN Verify, Encrypted PIN Verify2, FPE Decipher, FPE Encipher, FPE Translate, PIN Change/Unblock, Unique Key Derive
KDKGENKY	Diversify Directed Key
Secure-messaging class (data operation keys) These keys are used to encrypt keys or PINs in an EMV script.	
SECMSG	DK PIN Change, Secure Messaging for PINs

HMAC key types

HMAC keys are variable-length (80 - 2048 bits) keys used to generate and verify MACs using the key-hash MAC algorithm.

Table 7. Descriptions of HMAC key types and service usage	
HMAC key type	**Usable with services**
Variable-length AES key-token, version X'05'	
MAC class: • These keys are used to generate and verify a message authentication code (MAC).	
MAC	HMAC Generate, HMAC Verify, MAC Generate2, MAC Verify2

Clear keys

A clear key does not have it's key value encrypted under another key, unlike encrypted keys who do have their key value encrypted by a master key or key encrypting key.

There are four callable services you can use to convert a clear key to an encrypted key:

- To convert a clear key to an encrypted *data* key in operational form, use either the Clear Key Import callable service or the Multiple Clear Key Import callable service.
- To convert a clear key to an encrypted key of any type, in operational or importable form, use the Multiple Secure Key Import callable service.

Note: The Multiple Secure Key Import callable service can only execute in special secure mode.

AES and DES clear keys can be placed in key tokens and stored in the CKDS for use by callable services.

Table 8. Descriptions of Clear key types and service usage

Clear key type	**Usable with services**
Fixed-length DES key-token, version X'00' and X'01'	
DATA class (data operation keys): • These key are used to encrypt and decrypt data. • DES DATA keys can be single-length, double-length, or triple-length.	
DATA	Symmetric Key Decipher, Symmetric Key Encipher
DATA class (data operation keys): • AES DATA keys can be 128-bit, 192-bit and 256-bit keys	
DATA	Symmetric Key Decipher, Symmetric Key Encipher

X9.143 (TR-31) key blocks

X9.143 (TR-31) key block is a format defined by the American National Standards Institute (ANSI) to support the interchange of keys in a secure manner with key attributes included in the exchanged data. The TR-31 key block format has a set of defined key attributes that are securely bound to the key so that they can be transported together between any two systems that both understand the TR-31 format.

Support for operational (internal) TR-31 key blocks is available on IBM z16 and later servers with a CEX8 or later coprocessor and CCA release 8.1 or later licensed internal code. Operational TR-31 key blocks can be stored in the CKDS using the large common record (KDSRL) format of the CKDS.

Note: Support for the KDSRL format requires z/OS V2R5 ICSF (FMID HCR77D2).

Although there is often a one-to-one correspondence between TR-31 key attributes and the attributes defined by CCA, there are also cases where the correspondence is many-to-one or one-to-many. Because there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the TR-31 Translate callable service and the TR-31 Import callable service provide rule array keywords that enable an application to specify the attributes to attach to the exported or imported key.

Table 9 on page 29 list all CCA callable services that support operational x9.143 key blocks. The services are described in other sections of this publication.

Table 9. CCA callable services and parameters that support operational X9.143 key blocks

Descriptive name	**Service name**	**Parameters where an operational key block can be used**
Authentication Parameter Generate	CSNBAPG	inbound_PIN_encrypting_key_identifier(input) AP_encrypting_key_identifier (input)
Cipher Text Translate2	CSNBCTT2, CSNBCTT3	key_identifier_in (input) key_identifier_out (input)
CKDS Key Record Create2	CSNBKRC2	key_token (input)
CKDS Key Record Read2	CSNBKRR2	key_token (output)
CKDS Key Record Write2	CSNBKRW2	key_token (input)
Clear PIN Encrypt	CSNBCPE	PIN_encrypting_key_identifier (input)
Clear PIN Generate	CSNBPGN	PIN_generating_key_identifier (input)
Clear PIN Generate Alternate	CSNBCPA	PIN_encryption_key_identifier (input) PIN_generation_key_identifier (input)
Data Key Export	CSNBDKX	exporter_key_identifier (input)
Data Key Import	CSNBDKM	importer_key_identifier (input)
Decipher	CSNBDEC	CSNBDEC1 key_identifier (input)
Diversified Key Generate	CSNBDKG	generating_key_identifier (input) data_decrypting_key_identifier (input) generated_key_identifier (output)
Diversified Key Generate2	CSNBDKG2	generating_key_identifier (input) generated_key_identifier1 (output)
ECC Diffie-Hellman	CSNDEDH	private_KEK_key_identifier (input) output_KEK_key_identifier (input) hybrid key identifier (input) output_key_identifier (output)
Encipher	CSNBENC, CSNBENC1	key_identifier (input)

Table 9. CCA callable services and parameters that support operational X9.143 key blocks (continued)

Descriptive name	Service name	Parameters where an operational key block can be used
Encrypted PIN Generate	CSNBEPG	PIN_generating_key_identifier (input) outbound_PIN_encrypting_key_identifier (input)
Encrypted PIN Translate2	CSNBPTR2	input_PIN_encrypting_key_identifier (input) output_PIN_encrypting_key_identifier (input) authentication_key_identifier (input)
Encrypted PIN Translate Enhanced	CSNBPTRE	input_PIN_key_identifier (input) output_PIN_key_identifier (input) PAN_key_identifier (input)
Encrypted PIN Verify	CSNBPVR	input_PIN_encrypting_key_identifier (input) PIN_verifying_key_identifier (input)
Encrypted PIN Verify2	CSNBPVR2	input_PIN_encrypting_key_identifier (input) reference_PIN_encrypting_key_identifier (input)
Field Level Decipher	CSNBFLD	key_identifier (input)
Field Level Encipher	CSNBFLE	key_identifier (input)
Format Preserving Algorithms Decipher	CSNBFFXD	key_identifier (input)
Format Preserving Algorithms Encipher	CSNBFFXE	key_identifier (input)
Format Preserving Algorithms Translate	CSNBFFXT	input_key_identifier (input) output_key_identifier (input)
FPE Decipher	CSNBFPED	key_identifier (input) DUKPT_PIN_key_identifier (output)
FPE Encipher	CSNBFPEE	key_identifier (input) DUKPT_PIN_key_identifier (output)

Table 9. CCA callable services and parameters that support operational X9.143 key blocks (continued)

Descriptive name	Service name	Parameters where an operational key block can be used
FPE Translate	CSNBFPET	input_key_identifier (input) output_key_identifier (input) DUKPT_PIN_key_identifier (output)
HMAC Generate	CSNBHMG, CSNBHMG1	key_identifier (input)
HMAC Verify	CSNBHMV, CSNBHMV1	key_identifier (input)
Key Export	CSNBKEX	exporter_key_identifier (input)
Key Generate	CSNBKGN	KEK_key_identifier_1 (input) KEK_key_identifier_2 (input)
Key Generate2	CSNBKGN2	key_encrypting_key_identifier_1 (input) key_encrypting_key_identifier_2 (input)
Key Import	CSNBKIM	importer_key_identifier (input)
Key Part Import2	CSNBKPI2	key_identifier (input/output)
Key Test2	CSNBKYT2	key_identifier (input) key_encrypting_key_identifier (input)
Key Translate2	CSNBKTR2	input_KEK_identifier (input) output_KEK_identifier (input)
MAC Generate	CSNBMGN, CSNBMGN1	key_identifier (input)
MAC Generate2	CSNBMGN2, CSNBMGN3	key_identifier (input)
MAC Verify	CSNBMVR, CSNBMVR1	key_identifier (input)
MAC Verify2	CSNBMVR2, CSNBMVR3	key_identifier (input)
Multiple Secure Key Import	CSNBSKM	key_encrypting_key_identifier (input)
PIN Change/Unblock	CSNBPCU	authentication_issuer_master_key_identifier (input) encryption_issuer_master_key_identifier (input) new_reference_PIN_key_identifier (input) current_reference_PIN_key_identifier (input)
PKA Encrypt	CSNDPKE	sym_key_identifier (input)
PKA Key Generate	CSNDPKG	transport_key_identifier (input)

Table 9. CCA callable services and parameters that support operational X9.143 key blocks (continued)

Descriptive name	Service name	Parameters where an operational key block can be used
PKA Key Import	CSNDPKI	importer_key_identifier (input)
PKA Key Translate	CSNDPKT	source_transport_key_identifier (input) target_transport_key_identifier (input)
Random Number Generate Long	CSNBRNGL	key_identifier (input)
Recover PIN from Offset	CSNBPFO	PIN_encryption_key_identifier (input) PIN_generation_key_identifier (input)
Restrict Key Attribute	CSNBRKA	key_identifier (input) key_encrypting_key_identifier (input)
Secure Key Import2	CSNBSKI2	key_encrypting_key_identifier (input)
Secure Messaging for Keys	CSNBSKY	input_key_identifier (input) key_encrypting_key_identifier (input) secmsg_key_identifier (input)
Secure Messaging for PINs	CSNBSPN	PIN_encrypting_key_identifier (input) secmsg_key_identifier (input)
Symmetric Algorithm Decipher	CSNBSAD, CSNBSAD1	key_identifier (input)
Symmetric Algorithm Encipher	CSNBSAE, CSNBSAE1	key_identifier (input)
Symmetric Key Decipher	CSNBSYD, CSNBSYD1	key_identifier (input)
Symmetric Key Encipher	CSNBSYE, CSNBSYE1	key_identifier (input)
Symmetric Key Export	CSNDSYX	transporter_key_identifier (input)
Symmetric Key Export with Data	CSNDSXD	source_key_identifier (input)
Symmetric Key Generate	CSNDSYG	key_encrypting_key_identifier (input) local_enciphered_key_token (input)
Symmetric Key Import2	CSNDSYI2	transport_key_identifier (input)

Table 9. CCA callable services and parameters that support operational X9.143 key blocks (continued)		
Descriptive name	**Service name**	**Parameters where an operational key block can be used**
TR-31 Create	CSNBT31C	KEK_key_identifier_1 (input) KEK_key_identifier_2 (input) generated_key_identifier_1 (output) generated_key_identifier_2 (output)
TR-31 Import	CSNBT31I	unwrap_kek_identifier (input) wrap_kek_identifier (input) output_key_identifier (output)
TR-31 Translate	CSNBT31X	source_key_identifier (input) unwrap_kek_identifier (input) wrap_kek_identifier (input)
TR-34 Key Distribution	CSNDT34D	source_key_identifier (input) unwrap_kek_identifier (input)
TR-34 Key Receive	CSNDT34R	output_key_identifier (output)
Transaction Validation	CSNBTRV	transaction_key_identifier (input)
Unique Key Derive	CSNBUKD	base_derivation_key_identifier (input) generated_key_identifier1 (input) generated_key_identifier2 (input)
VISA CVV Service Generate	CSNBCSG	CVV_key_A_Identifier (input) CVV_key_B_Identifier (input)
VISA CVV Service Verify	CSNBCSV	CVV_key_A_Identifier (input) CVV_key_B_Identifier (input)

Key strength and wrapping of key

Key strength is measured as "bits of security" as described in the documentation of NIST and other organizations. Each individual key will have its "bits of security" computed, then the different key types (AES, DES, ECC, RSA, HMAC) can then have their relative strengths compared on a single scale. When the raw value of a particular key falls between discrete values of the NIST table, the lower value from the table will be used as the "bits of security".

The following tables show some examples of the restrictions due to key strength.

When wrapping an HMAC key with an AES key-encrypting key, the strength of the AES key-encrypting key depends on the attributes of the HMAC key.

Table 10. AES EXPORTER strength required for exporting an HMAC key under an AES EXPORTER	
Key-usage field 2 in the HMAC key	**Minimum strength of AES EXPORTER to adequately protect the HMAC key**
SHA-256, SHA-384, SHA-512	256 bits
SHA-224	192 bits
SHA-1	128 bits

Table 11. Minimum RSA modulus length to adequately protect an AES key	
Bit length of AES key to be exported	**Minimum strength of RSA wrapping key to adequately protect the AES key**
128	3072
192	7860
256	15360

Key strength and key wrapping access control points

In order to comply with cryptographic standards, including ANSI X9.24 Part 1 and PCI-HSM, ICSF provides a way to ensure that a key is not wrapped with a key weaker than itself. ICSF provides a set of access control points in the domain role to control the wrapping of keys. ICSF administrators can use these access control points to meet an installation's individual requirements.

There are new and existing access control points that control the wrapping of keys by master and key-encrypting keys. These access control points will either prohibit the wrapping of a key by a key of weaker strength or will return a warning (return code 0, reason code non-zero) when a key is wrapped by a weaker key. All of these ACPs are disabled by default in the domain role.

The processing of callable services will be affected by these access control points. Here is a description of the access control points, the wrapping they control, and the effect on services. These access control points apply to symmetric and asymmetric keys.

When the **Prohibit weak wrapping - Transport keys** access control point is enabled, any service that attempts to wrap a key with a weaker transport key will fail.

When the **Prohibit weak wrapping - Master keys** access control point is enabled, any service that wraps a key under a master key will fail if the master key is weaker than the key being wrapped.

When the **Warn when weak wrap - Transport keys** access control point is enabled, any service that attempts to wrap a key with a weaker transport key will succeed with a warning reason code.

When the **Warn when weak wrap - Master keys** access control point is enabled, any service that attempts to wrap a key with a weaker master key will succeed with a warning reason code.

When the **Allow weak wrapping of compliance-tagged keys by DES MK** access control point is enabled, any service which attempts to wrap a compliant-tagged key token with a weaker DES master key will succeed.

24-byte DATA keys with a zero control vector can be wrapped with a 16-byte key, the DES master key, or a key-encrypting key, which violates the wrapping requirements. The **Prohibit weak wrapping – Transport keys** and **Prohibit weak wrapping – Master keys** access control points do not cause services to fail for this case. The **Disallow 24-byte DATA wrapped with 16-byte Key** access control point does control this wrapping. When enabled, services will fail. The **Warn when weak wrap – Transport keys** and **Warn when weak wrap – Master keys** access control points will cause the warning to be returned when the access control points are enabled.

When the **TBC – Disallow triple-length MAC key** access control point is enabled, CSNDRKX will fail to import a triple-length MAC key under a double-length key-encrypting key. CSNDTBC will not wrap a triple-

length MAC key under a double-length key-encrypting key. The **Prohibit weak wrapping – Transport keys** and **Prohibit weak wrapping – Master keys** access control points do not cause services to fail for this case. The **Warn when weak wrap – Transport keys** and **Warn when weak wrap – Master keys** access control points will cause the warning to be returned when the ACPs are enabled.

If the **Prohibit Weak Wrap** access control point is enabled, RSA private keys may not be wrapped using a weaker DES key-encrypting key. Enabling the **Allow weak DES wrap of RSA** access control points will override this restriction.

In addition to key wrapping access control points, ICSF provides a set of access control points in the domain role to control the usage and generation of weak keys. These access control points prohibit the usage and generation of keys that fall below the minimum key strength requirement of the respective access control point. All of these ACPs are disabled by default in the domain role.

When the **Disable 56-bit length DES Keys** access control point is enabled, any service that attempts to accept or generate a 56-bit length DES key will fail.

When the **Disable 56-bit effective length DES keys** access control point is enabled, any service that attempts to accept or generate a 56-bit effective length DES key (112-bit or 168-bit keys with repeated 56-bit sections) will fail. This will also disallow loading a master key that has a 56-bit effective length.

When the **Disable RSA keys with less than 1024-bit modulus length** access control point is enabled, any service that attempts to accept or generate RSA keys with a modulus length less than 1024-bit will fail.

When the **Disable RSA keys with less than 2048-bit modulus length** access control point is enabled, any service that attempts to accept or generate RSA keys with a modulus length less than 2048-bit will fail.

When the **Disable ECC keys weaker than 224-bit** access control point is enabled, any service that attempts to accept or generate ECC keys weaker than 224-bit (P192, BP160, BP192) will fail.

DES master key

The DES master key can be either a 16-byte or 24-byte key. When a 16-byte DES master key is used, ICSF cannot be compliant for key strength for 24-byte operational keys wrapped by the DES master key. Starting with ICSF FMID HCR77A0, a 24-byte master key can be loaded on zEC12, zBC12, and later servers with CEX4 or later coprocessors with the October 2012 or later licensed internal code (LIC). The **DES master key – 24-byte key** access control must be enabled in the domain role. See *z/OS Cryptographic Services ICSF Administrator's Guide* for more details.

DK PIN methods support

This topic describes the financial services that are based on the PIN methods of and meet the requirements specified by the German Banking Industry Committee, *Die Deutsche Kreditwirtschaft*, also known as DK. The intellectual property rights regarding the methods and specification belongs to the German Banking Industry Committee.

The DK services are:

- "Diversify Directed Key (CSNBDDK and CSNEDDK)" on page 36
- "DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG)" on page 36
- "DK Migrate PIN (CSNBDMP and CSNEDMP)" on page 36
- "DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)" on page 36
- "DK PAN Translate (CSNBDPT and CSNEDPT)" on page 36
- "DK PIN Change (CSNBDPC and CSNEDPC)" on page 36
- "DK PIN Verify (CSNBDPV and CSNEDPV)" on page 36
- "DK PRW Card Number Update (CSNBDPNU and CSNEDPNU) and DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2)" on page 36

- “DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG)” on page 36
- “DK Random PIN Generate (CSNBDRPG and CSNEDRPG) and DK Random PIN Generate2 (CSNBDRG2 and CSNEDRG2)” on page 37
- “DK Regenerate PRW (CSNBDRP and CSNEDRP)” on page 37

Diversify Directed Key (CSNBDDK and CSNEDDK)

The Diversify Directed Key callable service is used to generate and derive session keys using the DK direct key diversification key scheme.

DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG)

The DK Deterministic PIN Generate service is used to generate a PIN and PIN reference value (PRW) using an AES PIN calculation key. The PIN reference value is used to verify the PIN in other services.

DK Migrate PIN (CSNBDMP and CSNEDMP)

The DK Migrate PIN service is used to generate a PIN reference value (PRW) for an existing ISO-1 formatted PIN block. The PIN reference value is used to verify the PIN in other services.

DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)

The DK PAN Modify in Transaction service is used to obtain a new PIN reference value (PRW) for an existing PIN when a merger has occurred and the account information has changed.

DK PAN Translate (CSNBDPT and CSNEDPT)

The DK PAN Translate service is used to create an encrypted PIN block with the same PIN and a different PAN. The account data may change, but changing the PIN is to be avoided. This service creates a new encrypted PIN block and MAC on the encrypted PIN block that will be used to accept the PAN change at an authorization node.

DK PIN Change (CSNBDPC and CSNEDPC)

The DK PIN Change service is used to allow a customer to change their PIN. The existing PIN and PIN reference value (PRW) and the new PIN are inputs and a new PRW is generated. Optionally, an encrypted PIN block can be generated or an encrypted script with the PIN embedded can be generated.

DK PIN Verify (CSNBDPV and CSNEDPV)

The DK PIN Verify service is used to verify the PIN in a transaction. The account, the card data, and PRW are used to verify the PIN.

DK PRW Card Number Update (CSNBDPNU and CSNEDPNU) and DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2)

The DK PRW Card Number Update and DK PRW Card Number Update 2 services are used to generate a PIN reference value (PRW) when a replacement card is being issued. The original PAN data and PIN are used with a new card number to generate the new PRW. An optional encrypted PIN block is generated for chip card processing (CSNBDCU2).

DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG)

The DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG) service is used to generate a message authentication code (MAC) over specific values involved in an account number change transaction. The inputs include the current and new PAN and card data and the PIN reference value.

DK Random PIN Generate (CSNBDRPG and CSNEDRPG) and DK Random PIN Generate2 (CSNBDRG2 and CSNEDRG2)

The DK Random PIN Generate and DK Random PIN Generate2 services generate a random PIN and calculates the PRW. The account and card data are used to generate the PRW. An optional encrypted PIN block is generated for printing. An optional encrypted PIN block is generated for chip card processing (CSNBDRG2).

DK Regenerate PRW (CSNBDRP and CSNEDRP)

The DK Regenerate PRW (CSNBDRP and CSNEDRP) service is used to generate a new PIN reference value for a changed account number.

Australian Payment Network support

Support for the Australian Payment Network is based on standard AS2805.5.4:

Key derivation

- CSNBDKG supports key derivation to meet the needs of the APN.
- CSNBRNGL supports encrypting the output under a data-encrypting key.

MAC generation
CSNBSAE supports generating and verifying MACs and related processing.

Generating and managing symmetric keys

Using ICSF, you can generate keys using either the *key generator utility program* or the *key generate callable service*. The dynamic CKDS update callable services allow applications to directly manipulate the CKDS. ICSF provides callable services that support DES and AES key management as defined by the IBM Common Cryptographic Architecture (CCA).

The next few topics describe the key generating and management options ICSF provides.

Key Generator Utility Program

The key generator utility program generates data, data-translation, MAC, PIN, and key-encrypting keys, and enciphers each type of key under a specific master key variant. When the KGUP generates a key, it stores it in the cryptographic key data set (CKDS).

Note: If you specify CLEAR, KGUP uses the random number generate and secure key import callable services rather than the key generate service.

You can access KGUP using ICSF panels. The KGUP path of these panels helps you create the JCL control statements to control the key generator utility program. When you want to generate a key, you can enter the ADD control statement and information, such as the key type on the panels. For a detailed description of the key generator utility program and how to use it to generate keys, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Common Cryptographic Architecture DES Key Management Services

ICSF provides callable services that support CCA key management for DES keys.

Clear Key Import Callable Service (CSNBCKI and CSNECKI)

This service imports a clear single-length DES DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted single-length DES DATA key in operational form in an internal key token.

Control Vector Generate Callable Service (CSNBCVG and CSNECVG)

The control vector generate callable service builds a control vector from keywords specified by the *key_type* and *rule_array* parameters.

Control Vector Translate Callable Service (CSNBCVT and CSNECVT)

The control vector translate callable service changes the control vector used to encipher an external key.

Cryptographic Variable Encipher Callable Service (CSNBCVE and CSNECVE)

The cryptographic variable encipher callable service uses a DES CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. You can use this service to prepare a mask array for the control vector translate service. The plaintext must be a multiple of eight bytes in length.

Data Key Export Callable Service (CSNBDKX and CSNEDKX)

This service reenciphers a DATA key from encryption under the master key to encryption under an exporter key-encrypting key, making it suitable for export to another system.

Data Key Import Callable Service (CSNBDKM and CSNEDKM)

This service imports an encrypted source DES DATA key and creates or updates a target internal key token with the master key enciphered source key.

Diversified Key Generate Callable Service (CSNBDKG and CSNEDKG)

The diversified key generate service generates a key based on the key-generating key, the processing method, and the parameter supplied. The control vector of the key-generating key also determines the type of target key that can be generated.

Key Encryption Translate Callable Service (CSNBKET and CSNEKET)

Use the Key Encryption Translate service to change the method of encryption of the key material.

Key Export Callable Service (CSNBKEX and CSNEKEX)

This service reenciphers any type of key (except IMP-PKA key) from encryption under a master key variant to encryption under the same variant of an exporter key-encrypting key, making it suitable for export to another system.

Key Generate Callable Service (CSNBKGN and CSNEKGN)

The key generate callable service generates data, data-translation, MAC, PIN, and key-encrypting keys. It generates a single key or a pair of keys. Unlike the key generator utility program, the key generate service does not store the keys in the CKDS where they can be saved and maintained. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.

When you call the key generate callable service, include parameters specifying information about the key you want generated. Because the form of the key restricts its use, you need to choose the form you want the generated key to have. You can use the *key_form* parameter to specify the form. The possible forms are:

- **Operational,** if the key is used for cryptographic operations on the local system. Operational keys are protected by master key variants and can be stored in the CKDS or held by applications in internal key tokens.
- **Importable,** if the key is stored with a file or sent to another system. Importable keys are protected by importer key-encrypting keys.

- **Exportable,** if the key is transported or exported to another system and imported there for use. Exportable keys are protected by exporter key-encrypting keys and cannot be used by ICSF callable service.

Importable and exportable keys are contained in external key tokens. For more information on key tokens, refer to "Key token" on page 17.

Key Import Callable Service (CSNBKIM and CSNEKIM)

This service reenciphers a key from encryption under an importer key-encrypting key to encryption under the master key. The reenciphered key is in the operational form.

Key Part Import Callable Service (CSNBKPI and CSNEKPI)

This service combines clear key parts of any key type and returns the combined key value either in an internal token or as an update to the CKDS.

Key Test Callable Service (CSNBKYT, CSNEKYT, CSNBKYTX, and CSNEKYTX)

This service generates or verifies a secure cryptographic verification pattern for keys. A parameter indicates the action you want to perform.

The key to test can be in the clear or encrypted under a master key. The key test extended callable service works on keys encrypted under a KEK.

For generating a verification pattern, the service creates and returns a random number with the verification pattern. For verifying a pattern, you supply the random number from the call to the service that generated the pattern.

Key Token Build Callable Service (CSNBKTB and CSNEKTB)

The key token build callable service is a utility you can use to create skeleton key tokens as input to the key generate or key part import callable service. You can also use this service to build CCA key tokens for all key types that ICSF supports.

Key Translate Callable Service (CSNBKTR and CSNEKTR)

This service uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

Key Translate2 Callable Service (CSNBKTR2 and CSNEKTR2)

This service uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment. The Key Translate2 service supports applying the compliant tag to an existing token as well as checking if an existing token can be successfully tagged.

Multiple Clear Key Import Callable Service (CSNBCKM and CSNECKM)

This service imports a single-length, double-length, or triple-length clear DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted DATA key in operational form in an internal key token.

Multiple Secure Key Import Callable Service (CSNBSKM and CSNESKM)

This service enciphers a single-length, double-length, or triple-length clear key under the host master key or under an importer key-encrypting key. The clear key can then be imported as any of the possible key types. This service can be used only when ICSF is in special secure mode.

Prohibit Export Callable Service (CSNBPEX and CSNEPEX)

This service modifies an operational key so that it cannot be exported. This callable service does not support NOCV key-encrypting keys, DATA, MAC, or MACVER keys with standard control vectors.

Prohibit Export Extended Callable Service (CSNBPEXX and CSNEPEXX)

This service updates the control vector in the external token of a key in exportable form so that the receiver node can import the key but not export it. When the key import callable service imports such a token, it marks the token as non-exportable. The key export callable service does not allow export of this token.

Public Infrastructure Certificate callable service (CSNDPIC and CSNFPIC)

This service creates a self-signed PKCS #10 certificate signing request (CSR) based on an existing RSA or ECC private key/public key pair. The self-signed PKCS #10 request for the input public key is signed by the input private key.

Random Number Generate Callable Service (CSNBRNG, CSNERNG, CSNBRNGL, and CSNERNGL)

The random number generate callable service creates a random number value to use in generating a key. The callable service uses cryptographic hardware to generate a random number for use in encryption.

Remote Key Export Callable Service (CSNDRKX and CSNFRKX)

The remote key export callable service uses the trusted block to generate or export DES keys for local use and for distribution to an ATM or other remote device.

Restrict Key Attribute Callable Service (CSNBRKA and CSNERKA)

This service modifies a DES operational key so that it cannot be exported. This service modifies a operational key so that the key value of a double-length key must have unique key part.

Secure Key Import Callable Service (CSNBSKI and CSNESKI)

This service enciphers a clear key under the host master key or under an importer key-encrypting key. The clear key can then be imported as any of the possible key types. This service can be used only when ICSF is in special secure mode.

Note: The PKA encrypt, PKA decrypt, symmetric key generate, symmetric key import, and symmetric key export callable services provide a way of distributing DES DATA keys protected under a PKA key. See Chapter 3, “Introducing CCA PKA cryptography and using PKA callable services,” on page 93 for additional information.

Symmetric Key Export Callable Service (CSNDSYX, CSNFSYX and CSNDSXD)

This service transfers an application-supplied DES DATA key from encryption under the host master key to encryption under an application-supplied RSA public key. The application-supplied DATA key must be an internal key token or the label of such a token in the CKDS. The symmetric key import callable service can import the PKA-encrypted form at the receiving node.

Symmetric Key Generate Callable Service (CSNDSYG, CSNFSYG)

This service generates a DES DATA key. The generated key is encrypted under either the host master key or a key-encrypting key as well as the supplied RSA public key token.

Symmetric Key Import Callable Service (CSNDSYI and CSNFSYI)

This service imports a symmetric (DES) DATA key enciphered under an RSA public key. This service returns the key in operational form, enciphered under the DES master key.

Trusted Block Create Callable Service (CSNDTBC and CSNFTBC)

This service creates and activates a trusted block under two step process.

Unique Key Derive Callable Service (CSFBUKD and CSFEUKD)

Unique Key Derive will perform the key derivation process as defined in ANSI X9.24 Part 1, Using a Base Derivation Key and Derivation Data as inputs.

Common Cryptographic Architecture AES Key Management Services

ICSF provides callable services that support CCA key management for AES keys.

Diversified Key Generate2 Callable Service (CSNBDKG2 and CSNEDKG2)

The Diversified Key Generate2 callable service generates a AES key based on the AES key-generating key, the processing method, and the parameter supplied. The key usage fields of the key-generating key also determines the type of target key that can be generated.

Key Generate Callable Service (CSNBKGN and CSNEKGN)

The key generate callable service generates AES data keys. It generates a single operational key. Unlike the key generator utility program, the key generate service does not store the keys in the CKDS where they can be saved and maintained. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.

Key Generate2 Callable Service (CSNBKGN2 and CSNEKGN2)

The service generates AES keys. It generates one operational key or an operational key pair. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.

Key Part Import2 Callable Service (CSNBKPI2 and CSNEKPI2)

This service combines clear key parts of any AES key type and returns the combined key value either in an internal token or as an update to the CKDS.

Key Test2 Callable Service (CSNBKYT2 and CSNEKYT2)

This service generates or verifies a secure cryptographic verification pattern for AES keys. A parameter indicates the action you want to perform.

Key Token Build Callable Service (CSNBKTB and CSNEKTB)

The key token build callable service is a utility you can use to create clear fixed-length AES key tokens, secure AES key tokens and skeleton secure AES key tokens for use with other callable services. You can also use this service to build CCA key tokens for all key types that ICSF supports.

Key Token Build2 Callable Service (CSNBKTB2 and CSNEKTB2)

The key token build2 callable service is a utility you can use to create variable-length AES and HMAC skeleton key tokens for use with other callable services.

Multiple Clear Key Import Callable Service (CSNBCKM and CSNECKM)

This service imports a a 128-, 192- or 256-bit clear DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted DATA key in operational form in an internal key token.

Multiple Secure Key Import Callable Service (CSNBSKM and CSNESKM)

This service enciphers 128-, 192- or 256-bit clear DATA key under the host master key. This service can be used only when ICSF is in special secure mode.

Restrict Key Attribute Callable Service (CSNBRKA and CSNERKA)

This service modifies an AES operational key so that it cannot be exported.

Secure Key Import2 Callable Service (CSNBSKI2 and CSNESKI2)

This service enciphers a variable length clear AES or HMAC key under the AES master key or an AES key-encrypting key. This service can be used only when ICSF is in special secure mode.

Symmetric Key Export Callable Service (CSNDSYX, CSNFSYX, CSNDSXD, and CSNFSXD)

Use the symmetric key export callable service to transfer an application-supplied AES key from encryption under a master key to encryption under an application-supplied RSA public key or AES EXPORTER key. The application-supplied key must be an ICSF AES internal key token or the label of such a token in the CKDS. The Symmetric Key Import or Symmetric Key Import2 callable services can import the key encrypted under the RSA public key or AES EXPORTER at the receiving node.

Symmetric Key Generate Callable Service (CSNDSYG and CSNFSYG)

This service generates a symmetric DATA key and returns it encrypted under the host AES master key and encrypted under an RSA public key token.

The AES-encrypted key can only be an internal token encrypted under a host AES master key. You can use the symmetric key import callable service to import the PKA-encrypted form.

Symmetric Key Import Callable Service (CSNDSYI and CSNFSYI)

This service imports a symmetric DATA key enciphered under an RSA public key. This service returns the key in operational form, enciphered under the AES master key.

Symmetric Key Import2 Callable Service (CSNDSYI2 and CSNFSYI2)

This service imports an AES key enciphered under an RSA public key or AES key-encrypting key. This service returns the key in operational form, enciphered under the AES master key.

Common Cryptographic Architecture HMAC Key Management Services

ICSF provides callable services that support CCA key management for HMAC keys. HMAC keys are stored in the cryptographic key data set (CKDS).

Key Generate2 callable service (CSNBKGN2 and CSNEKGN2)

The service generates HMAC keys. It generates operational key or operational key pair. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.

Key Part Import2 callable service (CSNBKPI2 and CSNEKPI2)

This service combines clear key parts of any HMAC key type and returns the combined key value either in an internal token or as an update to the CKDS.

Key Test2 callable service (CSNBKYT2 and CSNEKYT2)

This service generates or verifies a secure cryptographic verification pattern for HMAC keys. A parameter indicates the action you want to perform.

Key Token Build2 callable service (CSNBKTB2 and CSNEKTB2)

This service is a utility you can use to create skeleton HMAC key tokens for use with other callable services.

Restrict Key Attribute callable service (CSNBRKA and CSNERKA)

This service modifies an HMAC operational key so that it cannot be exported.

Secure Key Import2 callable service (CSNBSKI2 and CSNESKI2)

This service enciphers a variable length clear HMAC key under the host master key. This service can be used only when ICSF is in special secure mode.

Symmetric Key Export Callable Service (CSNDSYX and CSNFSYX)

This service transfers an application-supplied symmetric key from encryption under the AES host master key to encryption under an application-supplied RSA public key. The application-supplied key must be an ICSF internal key token or the label of such a token in the CKDS. The symmetric key import callable service can import the PKA-encrypted key at the receiving node.

Symmetric Key Import2 Callable Service (CSNDSYI2 and CSNFSYI2)

This service imports an HMAC key enciphered under an RSA public key. This service returns the key in operational form, enciphered under the AES master key.

ECC Diffie-Hellman key agreement models

Token agreement scheme

The caller must have both the required key tokens and both Parties identifiers including a randomly generated nonce. Combine the exchanged nonce and Party Info into the party identifier. (Both parties must combine this information in the same format.) Then call the ECC Diffie-Hellman callable service. Specify a skeleton token or the label of a skeleton token as the output key identifier for the computed symmetric key material. Note, both parties must specify the same key type in their skeleton key tokens.

- Specify rule array keyword DERIV01 to denote the Static Unified Model key agreement scheme.
- Specify an ECC token as the private key identifier containing this party's ECC public-private key pair.
- Optionally specify a private KEK key identifier, if the key pair is in an external key token.
- Specify an ECC token as the public key identifier containing other party's ECC public key part.
- Specify a skeleton token as the output key identifier for the computed symmetric key material.
- Optionally specify an output KEK key identifier, if the output key is to be in an external key token.
- Specify the combined party info (including nonce) as the party identifier.
- Specify the desired size of the key to be derived (in bits) as the key bit length.

Obtaining the raw “Z” value

To use a key agreement scheme that differs from “Token agreement scheme” on page 43, you can obtain the raw shared secret "Z" and skip the key derivation step. The caller must then derive the final key material using a method of their choice. Do not specify any party info.

- Specify rule array keyword “PASSTHRU" to denote no key agreement scheme.
- Specify an ECC token as the private key identifier containing this party's ECC public-private key pair.
- Optionally specify a private KEK key identifier, if the key pair is in an external key token.
- Specify an ECC token as the public key identifier containing other party's ECC public key part.
- The output key identifier be populated with the resulting shared secret material.

Improved remote key distribution

Note: This improved remote key distribute support is only available on the z9 EC, z9 BC and higher servers.

New methods have been added for securely transferring symmetric encryption keys to remote devices, such as Automated Teller Machines (ATMs), PIN-entry devices, and point of sale terminals. These methods can also be used to transfer symmetric keys to another cryptographic system of any type, such as a different kind of Hardware Security Module (HSM) in an IBM or non-IBM computer server. This change is especially important to banks, since it replaces expensive human operations with network transactions that can be processed quickly and inexpensively. This method supports a variety of requirements, fulfilling the new needs of the banking community while simultaneously making significant interoperability improvements to related cryptographic key-management functions.

For the purposes of this description, the ATM scenario will be used to illustrate operation of the new methods. Other uses of this method are also valuable.

Remote key loading

Remote key loading refers to the process of installing symmetric encryption keys into a remotely located device from a central administrative site. This encompasses two phases of key distributions.

- Distribution of initial key encrypting keys (KEKs) to a newly installed device. A KEK is a type of symmetric encryption key that is used to encrypt other keys so they can be securely transmitted over unprotected paths.
- Distribution of operational keys or replacement KEKs, enciphered under a KEK currently installed in the device.

Old remote key loading example

Use an ATM as an example of the remote key loading process. A new ATM has none of the bank's keys installed when it is delivered from the manufacturer. The process of getting the first key securely loaded is difficult. This has typically been done by loading the first KEK into each ATM manually, in multiple cleartext key parts. Using dual control for key parts, two separate people must carry key part values to the ATM, then load each key part manually. Once inside the ATM, the key parts are combined to form the actual KEK. In this manner, neither of the two people has the entire key, protecting the key value from disclosure or misuse. This method is labor-intensive and error-prone, making it expensive for the banks.

Remote key loading methods

Remote key loading methods have been developed to overcome some of the shortcomings of the old manual key loading methods. These methods define acceptable techniques using public key cryptography to load keys remotely. Using these methods, banks will be able to load the initial KEKs without sending people to the remote device. This will reduce labor costs, be more reliable, and be much less expensive to install and change keys. The cryptographic features added provide methods for the creation and use of the special key forms needed for remote key distribution of this type. In addition, they provide ways to solve long-standing barriers to secure key exchange with non-IBM cryptographic systems.

Once an ATM is in operation, the bank can install new keys as needed by sending them enciphered under a KEK installed previously. This is straightforward in concept, but the cryptographic architecture in ATMs is often different from that of the host system sending the keys, and it is difficult to export the keys in a form understood by the ATM. For example, cryptographic architectures often enforce key-usage restrictions in which a key is bound to data describing limitations on how it can be used - for encrypting data, for encrypting keys, for operating on message authentication codes (MACs), and so forth. The encoding of these restrictions and the method used to bind them to the key itself differs among cryptographic architectures, and it is often necessary to translate the format to that understood by the target device prior to a key being transmitted. It is difficult to do this without reducing security in the system; typically it is done by making it possible to arbitrarily change key-usage restrictions. The methods described here provide a mechanism through which the system owner can securely control these translations, preventing the majority of attacks that could be mounted by modifying usage restrictions.

A data structure called a *trusted block* is defined to facilitate the remote key loading methods. The trusted block is the primary vehicle supporting these methods.

Trusted block

The trusted block is the central data structure to support all remote key loading functions. It provides great power and flexibility, but this means that it must be designed and used with care in order to have a secure system. This security is provided through several features of the design.

- A two step process is used to create a trusted block.
- The trusted block includes cryptographic protection that prevents any modification when it is created.
- A number of fields in the rules of a trusted block offer the ability to limit how the block is used, reducing the risk of it being used in unintended ways or with unintended keys.

The trusted block is the enabler which requires secure approval for its creation, then enables the export or generation of DES and TDES keys in a wide variety of forms as approved by the administrators who created the trusted block. For added security, the trusted blocks themselves can be created on a separate system, such as an xSeries server with an IBM 4764 Cryptographic Coprocessor, locked in a secure room. The trusted block can subsequently be imported into the zSeries server where they will be used to support applications.

There are two CCA callable services to manage and use trusted blocks: Trusted Block Create (CSNDTBC and CSNETBC) and Remote Key Export (CSNDRKX and CSNFRKX). The Trusted Block Create service creates a trusted block, and the Remote Key Export service uses a trusted block to generate or export DES keys according to the parameters in the trusted block. The trusted block consists of a header followed by several sections. Some elements are required, while others are optional.

Figure 1 on page 46 shows the contents of a trusted block. The elements shown in the figure give an overview of the structure and do not provide all of the details of a trusted block.

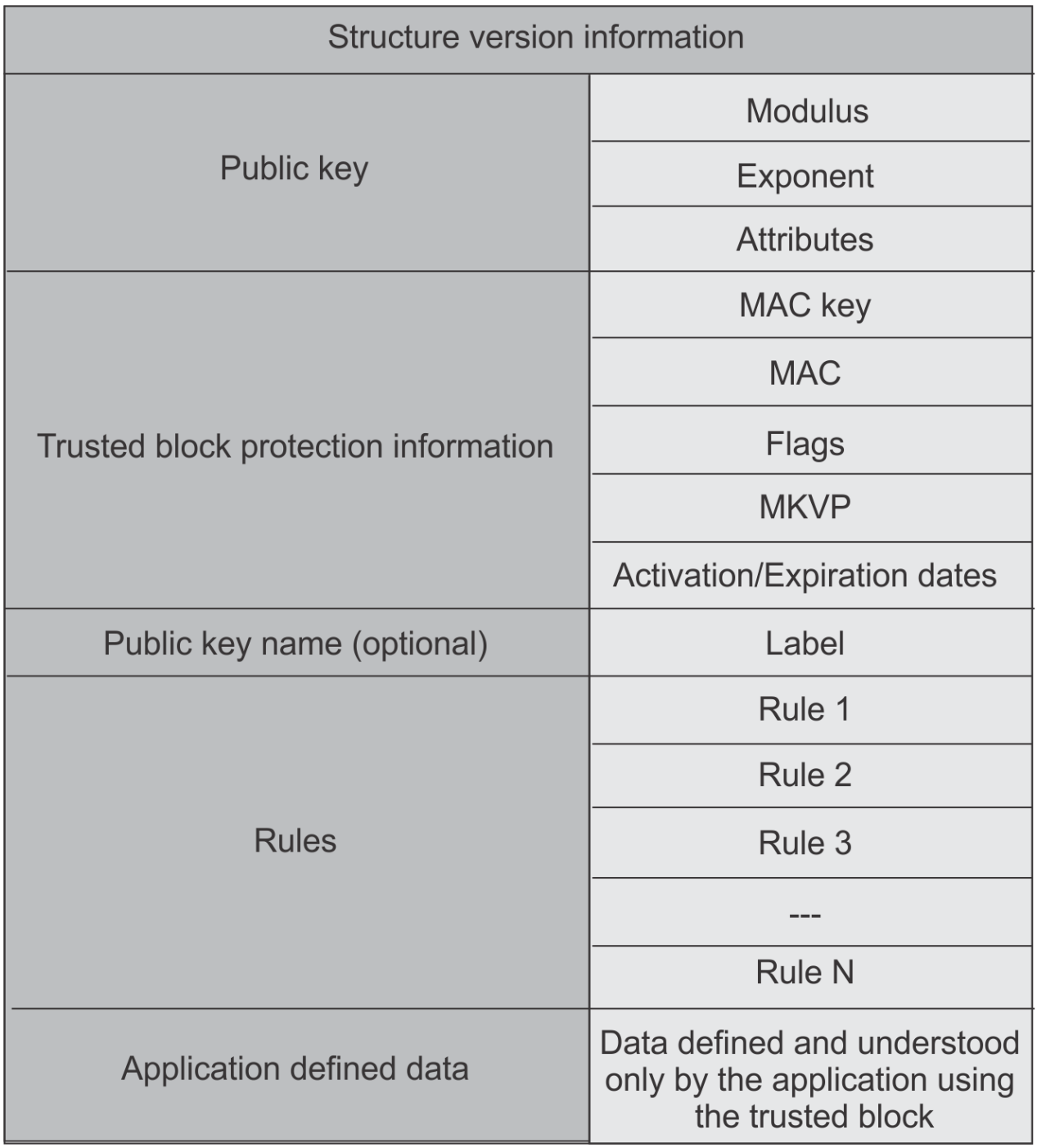


Figure 1. Overview of trusted block contents

Here is a brief description of the elements that are depicted.

Structure version information - This identifies the version of the trusted block structure. It is included so that code can differentiate between this trusted block layout and others that may be developed in the future.

Public key - This contains the RSA public key and its attributes. For distribution of keys to a remote ATM, this will be the root certification key for the ATM vendor, and it will be used to verify the signature on public-key certificates for specific individual ATMs. In this case, the Trusted Block will also contain Rules that will be used to generate or export symmetric keys for the ATMs. It is also possible for the Trusted Block to be used simply as a trusted public key container, and in this case the Public Key in the block will be used in general-purpose cryptographic functions such as digital signature verification. The public key

attributes contain information on key usage restrictions. This is used to securely control what operations will be permitted to use the public key. If desired, the public key can be restricted to use for only digital signature operations, or for only key management operations.

Trusted block protection information - This topic contains information that is used to protect the Trusted Block contents against modification. According to the method in ISO 16609, a CBC-mode MAC is calculated over the Trusted Block using a randomly-generated triple-DES (TDES) key, and the MAC key itself is encrypted and embedded in the block. For the internal form of the block, the MAC key is encrypted with a randomly chosen fixed-value variant of the PKA master key. For the external form, the MAC key is encrypted with a fixed variant of a key-encrypting key. The MKVP field contains the master key verification pattern for the PKA master key that was used, and is filled with binary zeros if the trusted block is in external format. Various flag fields contain these boolean flags.

- ***Active flag*** - Contained within the flags field of the required trusted block information section, this flag indicates whether the trusted block is active and ready for use by other callable services. Combined with the use of two separate access control points, the active flag is used to enforce dual control over creation of the block. A person whose active role is authorized to create a trusted block in inactive form creates the block and defines its parameters. An inactive trusted block can only be used to make it active. A person whose active role is authorized to activate an inactive trusted block must approve the block by changing its status to active. See Figure 3 on page 50. The Remote_Key_Export callable service can only use an internal active trusted block to generate or export DES keys according to the parameters defined in the trusted block.
- ***Date checking flag*** - Contained within the optional activation and expiration date subsection of the required trusted block information subsection, this flag indicates whether the coprocessor checks the activation and expiration dates for the trusted block. If the date checking flag is on, the coprocessor compares the activation and expiration dates in the optional subsection to the coprocessor internal real time clock, and processing terminates if either date is out of range. If this flag is off or the activation and expiration dates subsection is not defined, the device does no date checking. If this flag is off and the activation and expiration dates subsection is defined, date checking can still be performed outside of the device if required. The date checking flag enables use of the trusted block in systems where the coprocessor clock is not set.

Trusted block name - This field optionally contains a text string that is a name (key label) for the trusted block. It is included in the block for use by an external system such as a host computer, and not by the card itself. In the zSeries system, the label can be checked by RACF to determine if use of the block is authorized. It is possible to disable use of trusted blocks that have been compromised or need to be removed from use for other reasons by publishing a revocation list containing the key names for the blocks that must not be used. Code in the host system could check each trusted block prior to it being used in the cryptographic coprocessor, to ensure that the name from that block is not in the revocation list.

Expiration date and activation dates - The trusted block can optionally contain an expiration date and an activation date. The activation date is the first day on which the block can be used, and the expiration date is the last day when the block can be used. If these dates are present, the date checking flag in the trusted block will indicate whether the coprocessor should check the dates using its internal real-time clock. In the case of a system that does set the coprocessor clock, checking would have to be performed by an application program prior to using the trusted block. This is not quite as secure, but it is still valuable, and storing the dates in the block itself is preferable to making the application store it somewhere else and maintain the association between the separate trusted block and activation and expiration dates.

Application-defined data - The trusted block can hold data defined and understood only by the host application program. This data is included in the protected contents of the trusted block, but it is not used or examined in any way by the coprocessor. By including its own data in the trusted block, an application can guarantee that the data is not changed in any way, since it is protected in the same way as the other trusted block contents.

Rules - A variable number of rules can be included in the block. Each rule contains information on how to generate or export a symmetric key, including values for variants to be used in order to provide keys in the formats expected by systems with differing cryptographic architectures. Use of the rules are described in

the topics covering key generation and export using the RKX function. This table summarizes the required and optional values of each rule.

Field name	Required field	Description
Rule ID	Yes	Specifies the 8-character name of the rule.
Operation	Yes	Indicates whether this rule generates a new key or exports an existing key.
Generated key length	Yes	Specifies the length of the key to be generated.
Key-check algorithm ID	Yes	Specifies which algorithm to use to compute the optional key-check value (KCV). Options are: • No KCV. • Encrypt zeros with the key. • Compute MDC-2 hash of the key.
Symmetric-encrypted output format	Yes	Specifies the format of the required symmetric-encrypted key output. Options are: • CCA key token. • RKX key token.
Asymmetric-encrypted output format	Yes	Specifies the format of the optional asymmetric-encrypted key output (key is encrypted with RSA). Options are: • No asymmetric-encrypted key output. • Encrypt in PKCS1.2 format. • Encrypt in RSAOAEP format.
Transport-key variant	No	Specifies the variant to apply to the transport key prior to it being used to encrypt the key being generated or exported.
Export key CV	No	Specifies the CCA CV to apply to the transport key prior to it being used to encrypt the key being generated or exported. The CV defines permitted uses for the exported key.
Export key length limits	No	Defines the minimum and maximum lengths of the key that can be exported with this rule.
Output key variant	No	Specifies the variant to apply to the generated or exported key prior to it being encrypted.
Export-key rule reference	No	Specifies the rule ID for the rule that must have been used to generate the key being exported, if that key is an RKX key token.
Export-key CV restrictions	No	Defines masks and templates to use to restrict the possible CV values that a source key can have when being exported with RKX. Only applies if the key is a CCA key token. This can control the types of CCA keys that can be processed using the rule.
Export-key label template	No	Specifies the *key label* of the key token that contains the source key to be exported. A key label is a name used to identify a key. The rule can optionally contain a key label template, which will be matched against the host-supplied key label, using a wildcard (*) so that the template can match a set of related key labels. The operation will only be accepted if the supplied label matches the wildcard template in the rule.

Changes to the CCA API

These changes have been made to the CCA API to support remote key loading using trusted blocks:

- A new Trusted Block Create (CSNDTBC and CSNETBC) callable service has been developed to securely create trusted blocks under dual control.
- A new Remote Key Export (CSNDRKX and CSNFRKX) callable service has been developed to generate or export DES and TDES keys under control of the rules contained in a trusted block.
- The Digital Signature Verify (CSNDDSV) callable service has been enhanced so that, in addition to verifying ordinary CCA RSA keys, it can use the RSA public key contained in a trusted block to verify digital signatures.
- The PKA Key Import (CSNDPKI) callable service has been enhanced so it can import an RSA key into the CCA domain. In addition, the verb can import an external format trusted block into an internal format trusted block, ready to be used in the local system.
- The PKA Key Token Change (CSNDKTC and CSNFKTC) callable service has been enhanced so that it can update trusted blocks to the current PKA master key when the master key is changed. A trusted block contains an embedded MAC key enciphered under the PKA master key. When the PKA master key is changed, the outdated MAC key and the trusted block itself need to be updated to reflect the current PKA master key.
- The MAC Generate (CSNBMGN) and MAC Verify (CSNBMVR) callable services have been enhanced to add ISO 16609 TDES MAC support in which the text will be CBC-TDES encrypted using a double-length key and the MAC will be extracted from the last block.
- The PKA key storage callable services support trusted blocks.

The RKX key token

CCA normally uses key tokens that are designed solely for the purposes of protecting the key value and carrying metadata associated with the key to control its use by CCA cryptographic functions. The remote key loading design introduces a new type of key token called an RKX key token. The purpose of this token is somewhat different, and its use is connected directly with the Remote Key Export callable service added to CCA of the remote key loading design.

The RKX key token uses a special structure that binds the token to a specific trusted block, and allows sequences of Remote Key Export calls to be bound together as if they were an atomic operation. This allows a series of related key-management operations to be performed using the Remote Key Export callable service. These capabilities are made possible by incorporating these three features into the RKX key token structure:

- The key is enciphered using a variant of the MAC key that is in the trusted block. A fixed, randomly-derived variant is applied to the key prior to it being used. As a result, the enciphered key is protected against disclosure since the trusted block MAC key is itself protected at all times.
- The structure includes the rule ID contained in the trusted block rule that was used to create the key. A subsequent call to the Remote Key Export callable service can use this key with a trusted block rule that references this rule ID, effectively chaining use of the two rules together securely.
- A MAC is computed over the encrypted key and the rule ID, using the same MAC key that is used to protect the trusted block itself. This MAC guarantees that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. In addition, the MAC will only verify if the RKX key token is used with the same trusted block that created the token, thus binding the key to that specific trusted block.

This figure shows a simplified conceptual view of the RKX key token structure.

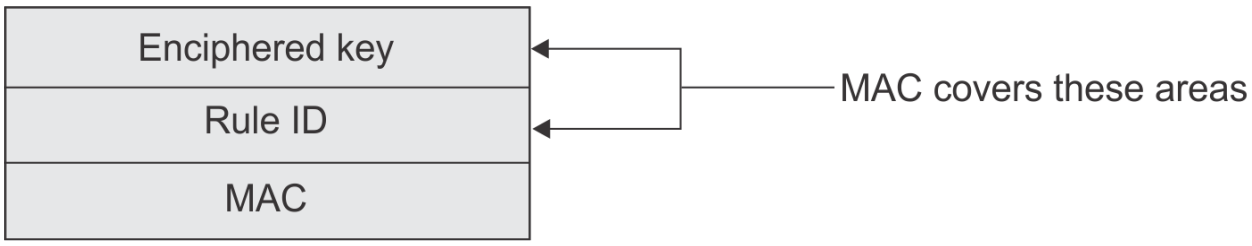


Figure 2. Simplified RKX key-token structure

Using trusted blocks

These examples illustrate how trusted blocks are used with the new and enhanced CCA callable services.

Creating a trusted block

This figure illustrates the steps used to create a trusted block.

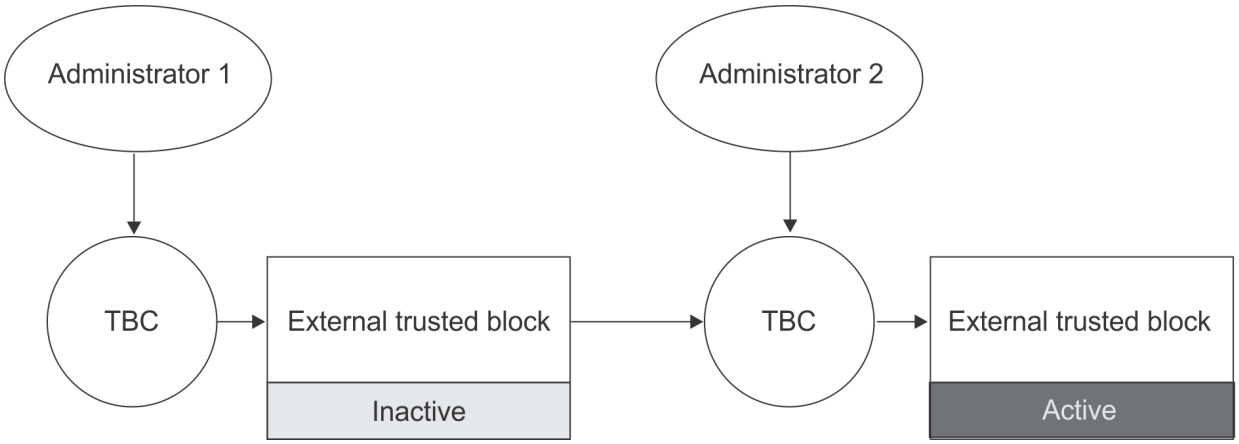
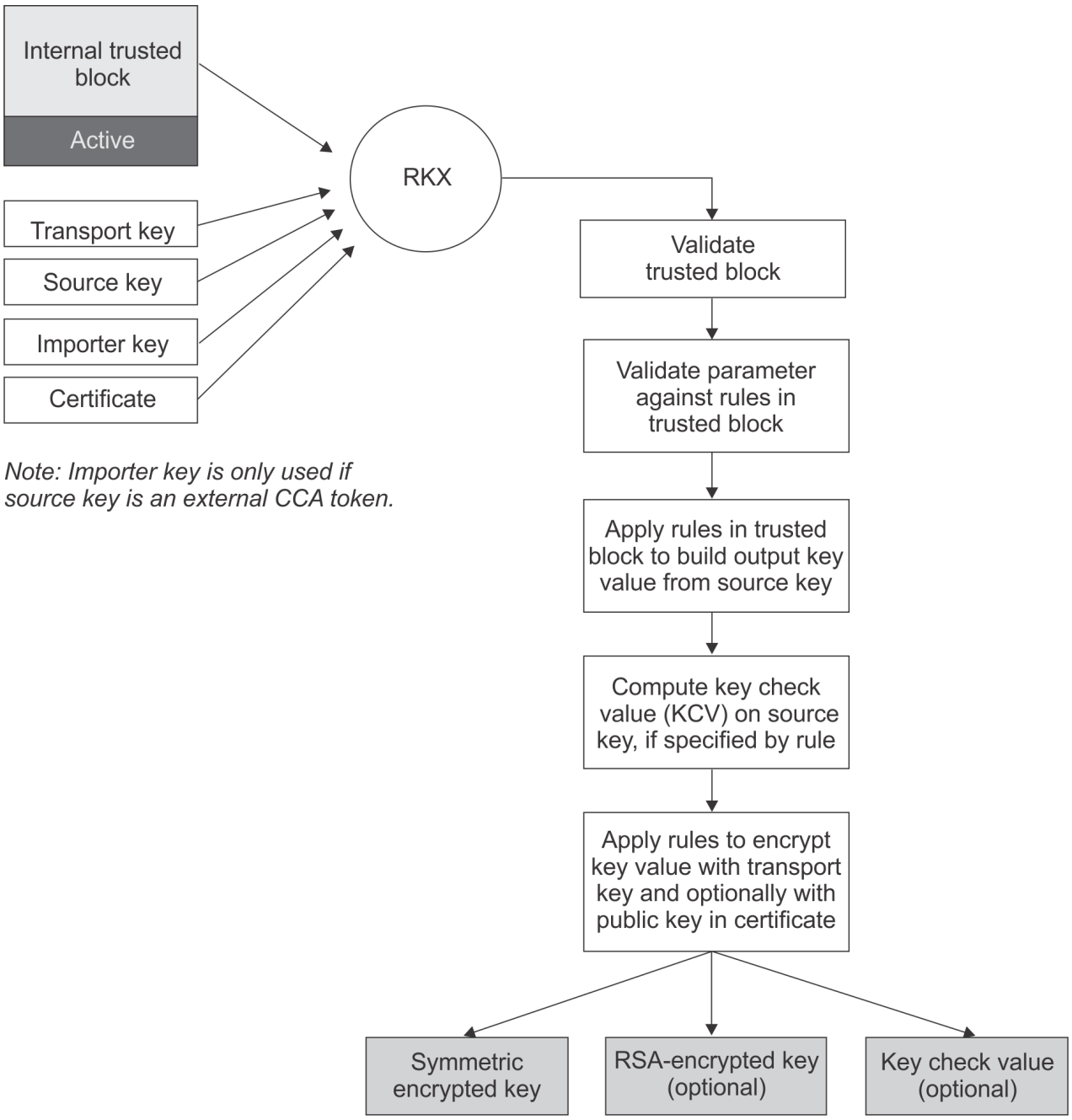


Figure 3. Trusted block creation

A two step process is used to create a trusted block. Trusted blocks are structures that could be abused to circumvent security if an attacker could create them with undesirable settings, and the requirement for two separate and properly authorized people makes it impossible for a single dishonest employee to create such a block. A trusted block cannot be used for any operations until it is in the active state. Any number of trusted blocks can be created in order to meet different needs of application programs.

Exporting keys with Remote_Key_Export

This figure shows the process for using a trusted block in order to export a DES or TDES key. This representation is at a very high level in order to illustrate the basic flow.



Note: Importer key is only used if source key is an external CCA token.

Figure 4. Exporting keys using a trusted block

The Remote Key Export callable service is called with these main parameters:

- A trusted block, in the active state, defines how the export operation is to be processed, including values to be used for things such as variants to apply to the keys.
- The key to be exported, shown previously as the source key. The source key takes one of two forms:
  1. A CCA DES key token
  2. An RKX key token
- A key-encrypting key, shown in the figure as the importer key. This is only used if the source key is an external CCA DES key token, encrypted under a KEK. In this case, the KEK is the key needed to obtain the cleartext value of the source key.
- A transport key, either an exporter KEK or an RKX key token, used to encrypt the key being exported.

- An optional public key certificate which, if included, contains the certified public key for a specific ATM. The certificate is signed with the ATM vendor's private key, and its corresponding public key must be contained in the trusted block so that this certificate can be validated. The public key contained in the certificate can be used to encrypt the exported key.

The processing steps are simple at a high level, but there are many options and significant complexity in the details.

- The trusted block itself is validated. This includes several types of validation.
  - Cryptographic validation using the MAC that is embedded in the block, in which the MAC key is decrypted using the coprocessor's master key, and the MAC is then verified using that key. This verifies the block has not been corrupted or tampered with, and it also verifies that the block is for use with this coprocessor since it will only succeed if the master key is correct.
  - Consistency checking and field validation, in which the validity of the structure itself is checked, and all values are verified to be within defined ranges.
  - Fields in the trusted block are checked to see if all requirements are met for use of this trusted block. One check which is always required is to ensure that the trusted block is in the active state prior to continuing. Another check, which is optional based on the contents of the trusted block, is to ensure the operation is currently allowed by comparing the date of the coprocessor real-time clock to the activation and expiration dates defined in the trusted block.
- Input parameters to the Remote Key Export callable service are validated against rules defined for them within the trusted block. For example:
  - The rule can restrict the length of the key to be exported.
  - The rule can restrict the control vector values for the key to be exported, so only certain key types can be exported with that rule.
- When the export key is decrypted, the rules embedded in the trusted block are then used to modify that key to produce the desired output key value. For example, the trusted block can contain a variant to be exclusive-ORed with the source key prior to when that key is encrypted. Many non-IBM cryptographic systems use variants to provide key separation to restrict a key from improper use.
- A key check value (KCV) can be optionally computed for the source key. If the KCV is computed, the trusted block allows for one of two key check algorithms to be used: (1) encrypting binary zeros with the key, or (2) computing an MDC-2 hash of the key. The KCV is returned as output from the Remote Key Export function.
- The export key, which could possibly be modified with a variant according to the rules in the trusted block, is enciphered with the transport key. The rules can specify that the key be created in one of two formats: (1) a CCA key token, or (2) the new RKX key token, described previously. With proper selection of rule options, the CCA key token can create keys that can be used in non-CCA systems. The key value can be extracted from the CCA key token and as a result, the extracted key value will be contained in a generic encrypted key, with variants and other options as defined in the rule.

  Two optional fields in the trusted block may modify the transport key prior to it being used to encrypt the source key:
  - The trusted block can contain a CCA control vector (CV) to be exclusive-ORed with the transport key prior to it being used to encrypt the export key. This exclusive-OR process is the standard way CCA applies a CV to a key.
  - In addition to the CV described previously, the trusted block can also contain a variant to be exclusive-ORed with the transport key prior to its use.

  If a variant and CV are both present in the trusted block, the variant is applied first, then the CV.
- The export key can optionally be encrypted with the RSA public key identified by the certificate parameter of the Remote Key Export callable service, in addition to encrypting it with the transport key as described previously. These two encrypted versions of the export key are provided as separate outputs of the Remote Key Export callable service. The trusted block allows a choice of encrypting the key in either PKCS1.2 format or PKCSOAEP format.

Generating keys with Remote_Key_Export

This figure shows the process for using a trusted block to generate a new DES or TDES key. This representation is at a very high level in order to illustrate the basic flow.

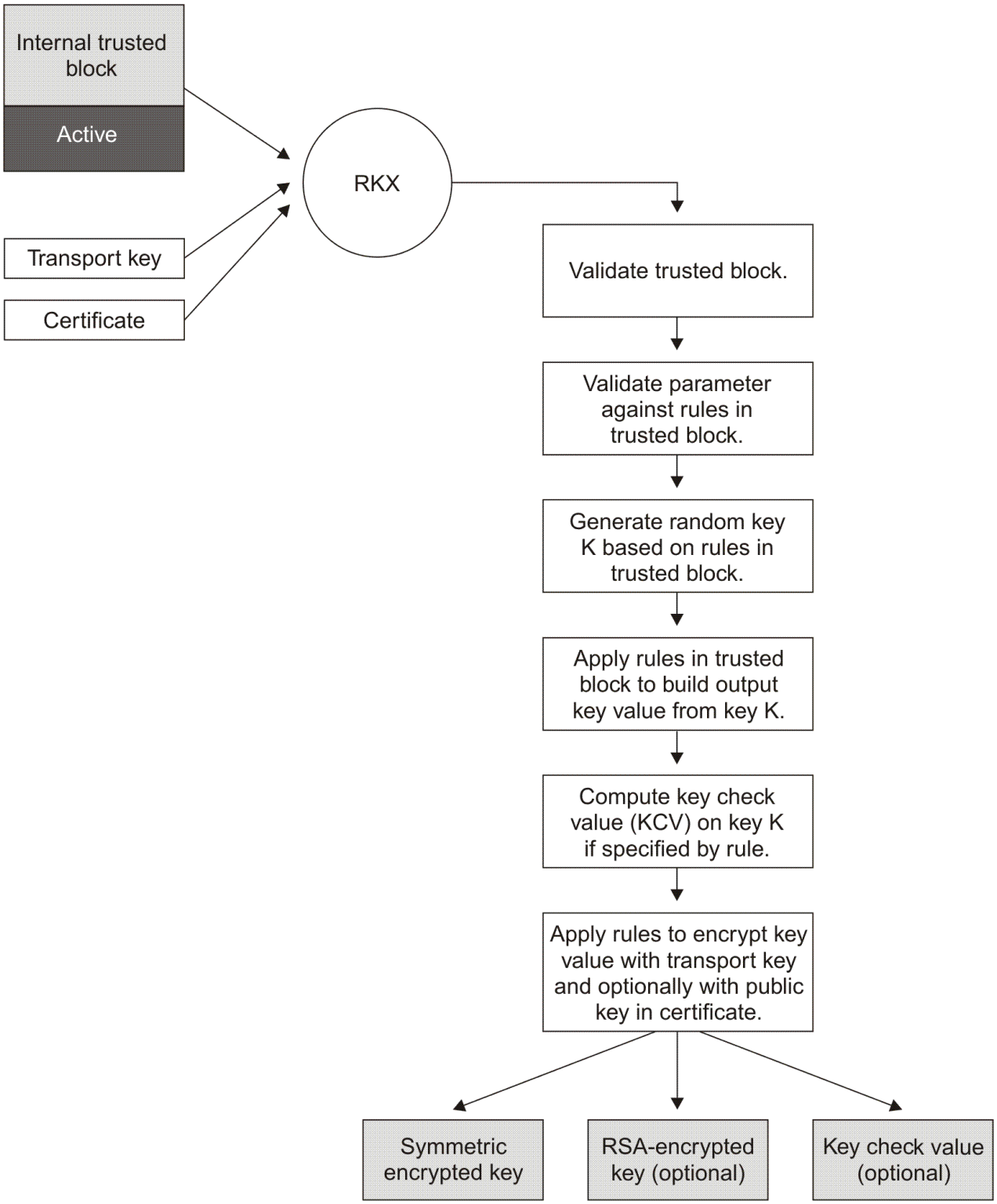


Figure 5. Generating keys using a trusted block

For key generation, the Remote Key Export callable service is called with these main parameters:

- A trusted block, in the internal active state, which defines how the key generation operation is to be processed, including values to be used for things such as variants to apply to the keys. The generated key is encrypted by a variant of the MAC key contained in a trusted block.
- An optional public key certificate which, if included, contains the certified public key for a specific ATM. The certificate is signed with the ATM vendor's private key, and its corresponding public key must be contained in the trusted block so that this certificate can be validated. The public key contained in the certificate can be used to encrypt the generated key.

The processing steps are simple at a high level, but there are many options and significant complexity in the details. Most of the processing steps are the same as those described previously for key export. Therefore, only those processing steps that differ are described here in detail.

- Validation of the trusted block and input parameters is done as described for export previously.
- The DES or TDES key to be returned by the Remote Key Export callable service is randomly generated. The trusted block indicates the length for the generated key.
- The output key value is optionally modified by a variant as described previously for export, and then encrypted in the same way as for export using the Transport key and optionally the public key in the certificate parameter.
- The key check value (KCV) is optionally computed for the generated key using the same method as for an exported key.

Remote key distribution scenario

The new and modified CCA functions for remote key loading are used together to create trusted blocks, and then generate or export keys under the control of those trusted blocks. This figure summarizes the flow of the CCA functions to show how they are used:

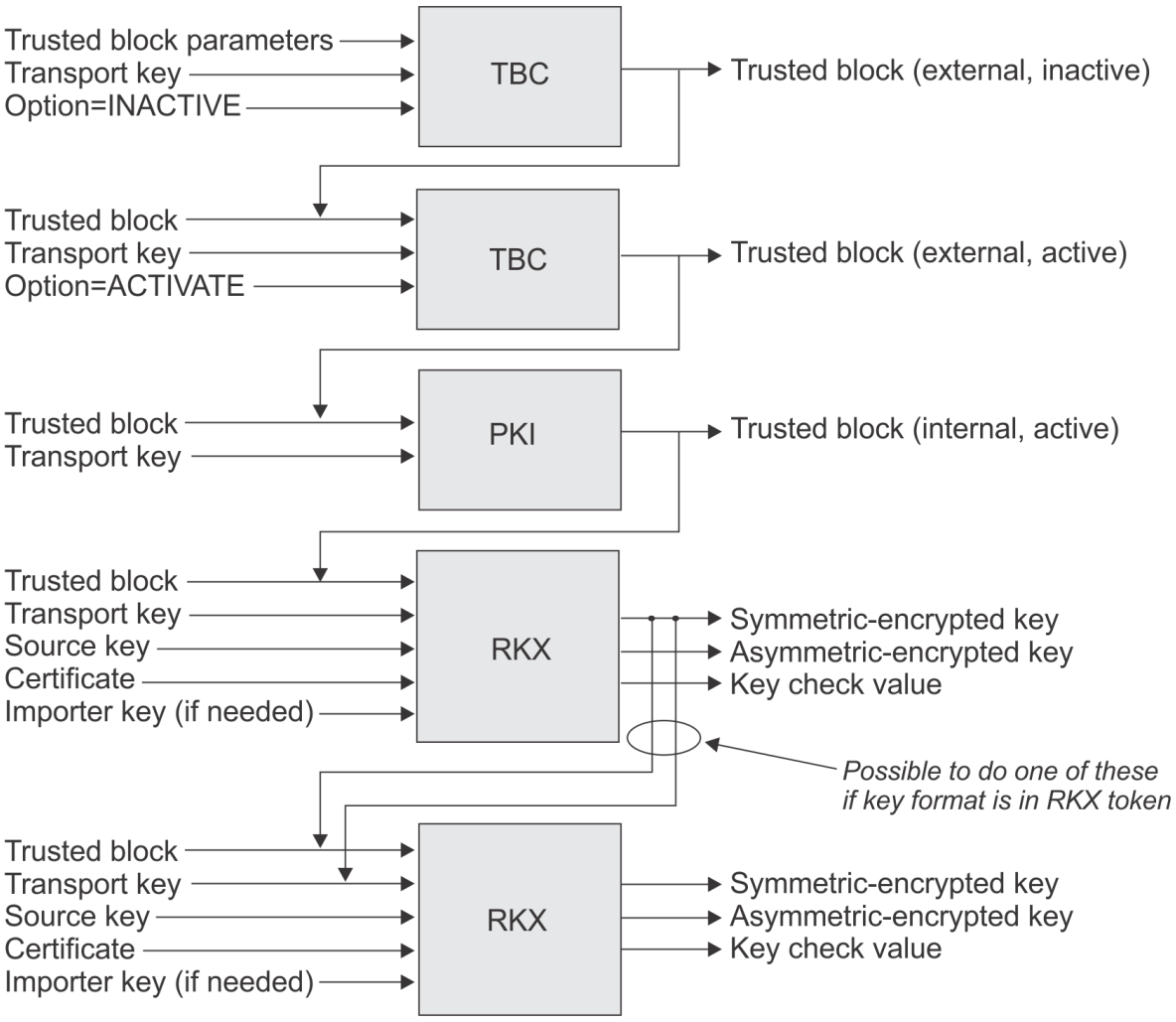


Figure 6. Typical flow of callable services for remote key export

Usage example

The scenario described shows how these functions might be combined in a real-life application to distribute a key to an ATM and keep a copy for local use. Some of the terminology used reflects typical terms used in ATM networks. The example illustrates a fairly complex real-world key distribution scenario, in which these values are produced.

- A TMK (Terminal Master Key), which is the root KEK used by the ATM to exchange other keys, is produced in two forms: (1) encrypted under the ATM public key, so it can be sent to the ATM, and (2) as an RKX key token that will be used in subsequent calls to the Remote Key Export callable service to produce other keys.
- A key-encrypting key KEK1 that is encrypted under the TMK in a form that can be understood by the ATM.
- A PIN-encrypting key PINKEY be used by the ATM to encrypt customer-entered PINs and by the host to verify those PINs. The PINKEY is produced in two forms:
  1. Encrypted under KEK1 in a form that can be understood by the ATM.
  2. As a CCA internal DES key token with the proper PIN-key CV, encrypted under the CCA DES master key and suitable for use with the coprocessor.

It takes seven steps to produce these keys using the Remote Key Export callable service. These steps use a combination of five rules contained in a single trusted block. The rules in this example are referred to as GENERAT1, GENERAT2, EXPORT1, EXPORT2, and EXPORT3.

1. Use the Remote Key Export callable service with rule ID "GENERAT1" to generate a TMK for use with the ATM. The key will be output in two forms:
   a. e_{Pu}(TMK): Encrypted under the ATM public key, supplied in the certificate parameter, CERT.
   b. RKX(TMK): As an RKX key token, suitable for subsequent input to the CSNDRKX callable service.
2. Use the Remote Key Export callable service with rule ID "GENERAT2" to generate a key-encrypting key (KEK1) as an RKX key token, RKX(KEK1).
3. Use the Remote Key Export callable service with rule ID "GENERAT2" to generate a PIN key (PINKEY) as an RKX key token: RKX(PINKEY).
4. Use the Remote Key Export callable service with rule ID "EXPORT1" to export KEK1 encrypted under the TMK as a CCA DES key token using a variant of zeros applied to the TMK. This produces e_{TMK}(KEK1).
5. Use the Remote Key Export callable service with rule ID "EXPORT2" to export PINKEY encrypted under KEK1 as a CCA token using a variant of zeros applied to KEK1. This produces e_{KEK1}(PINKEY).
6. Use the Remote Key Export callable service with rule ID "EXPORT3" to export PINKEY encrypted under KEK2 as an existing CCA key-encrypting key on the local server. This produces e_{KEK2}(PINKEY), with the CCA control vector for a PIN key.
7. Use the Key Import callable service to import the PINKEY produced in step 6 into the local system as an operational key. This produces e_{MK}(PINKEY), a copy of the key encrypted under the local DES master key (MK) and ready for use by CCA PIN API functions.

Remote key distribution benefits

CCA support for remote key loading allows the distribution of initial keys to ATMs and other remote devices securely using public-key techniques, in a flexible way that can support a wide variety of different cryptographic architectures. They also make it easier and more secure to send keys to non-CCA systems when those keys are encrypted with a triple-DES key-encrypting key. These changes make it easier to develop more secure systems.

Diversifying keys

CCA supports diversifying DES and AES symmetric keys. Key-diversification is a technique often used in working with smart cards. In order to secure interactions with a population of cards, a key-generating key is used with some data unique to a card to derive ('diversify') keys for use with that card. The data is often the card serial number or other quantity stored on the card. The data is often public, and therefore, it is very important to handle the key-generating key with a high degree of security or the interactions with the whole population of cards could be placed in jeopardy. CCA supports diversifying a DES key using the Diversified Key Generate callable service and diversifying an AES key using the Diversified Key Generate2 service.

Several methods of diversifying a DES key are supported. They are CLR8-ENC, TDES-ENC, TDES-DEC, SESS-XOR, TDESEMV2, TDESEMV4, and TDES-XOR.

- The CLR8-ENC and TDES-ENC methods triple-encrypt data using the *generating_key* to form the diversified key. The diversified key is then multiply-enciphered by the DES master-key modified by the control vector for the output key. The TDES-DEC method is similar except that the data is triple-decrypted.
- The TDES-ENC, TDES-CBC, and TDES-DEC methods permit the production of either another key-generating key or a final key. Control-vector bits 19 through 22 associated with the key-generating key specify the permissible type of the final key. For more information, see DKYGENKY in “Control-Vector Base Bits” on page 1605. Control-vector bits 12 through 14 associated with the key-generating key specify if the diversified key is a final key or another in a series of key-generating keys. Bits 12 through 14 specify a counter that is decreased by one each time that the diversified key generate service is used to produce another key-generating key. For example, if the key-generating key that you specify has its counter set to B'010', you must specify the control vector for the *generated_key* with a DKYGENKY key type having the counter bits set to B'001' and specify the same final key type in bits 19 through 22. Use of a *generating_key* with bits 12 through 14 set to B'000' results in the creation of the final key.

Therefore, you can control both the number of diversifications required to reach a final key and you can closely control the type of the final key.

- The SESS-XOR method provides a means for modifying an existing DATA, DATAC, MAC, DATAM, MACVER, or DATAMV single-length or double-length key. The provided data is exclusive-OR-ed into the clear value of the key. This form of key diversification is specified by several of the credit card associations.
- The TDESEMV2, TDESEMV4, and TDES-XOR methods also derive a key by encrypting supplied data including a transaction counter value received from an EMV smart card. The processes are described in detail in "Visa, MasterCard, and EMV-related smart card formats and processes" on page 1652. For information on the processing capabilities you can use with EMV smart cards, see "Working with Europay–MasterCard–Visa smart cards" on page 602.

Several methods of diversifying an AES key are supported. They are SESS-ENC, MK-OPTC, and KDFFM-DK.

- The SESS-ENC method of diversifying a key creates a session key by enciphering the 16 bytes of derivation data supplied with the *k*-bit AES key-generating key to produce a *k*-bit AES generated session key using the AES algorithm in ECB mode, where *k* is 128, 192, or 256.
- The MK-OPTC method creates a sequence level 0 key-generating key using the EMV Option C derivation method. This method uses AES in ECB mode to encipher the 16 bytes of derivation data with the *k*-bit diversified key generating key (Issuer Master Key) to produce a *k*-bit generated ICC master key, where *k* = 128, 192, or 256.
- The KDFFM-DK method uses a derivation method based on the NIST KDF in Feedback Mode. Note that this method is specific to the DK PIN methods. This method uses AES CMAC to generate the 16 to 40 bytes of derivation data with the *k*-bit diversified key generating key (banking association specific master key) to produce a *k*-bit generated Bank specific Issuer Master Key, where *k* = 128, 192, or 256.

Callable services for managing the CKDS

ICSF provides services to manage the CKDS. The dynamic CKDS update services allow applications to directly manipulate both the DASD copy and in-storage copy of the current CKDS. The KDS metadata services allow administrators to manage the metadata of records in the CKDS. The coordinated KDS administration service allows the CKDS to be refreshed and the symmetric master keys to be changed and the CKDS reenciphered programmatically.

Note: Applications using the dynamic CKDS update callable services can run concurrently with other operations that affect the CKDS, such as KGUP, CKDS conversion, REFRESH, and dynamic master key change. An operation can fail if it needs exclusive or shared access to the same DASD copy of the CKDS that is held shared or exclusive by another operation. ICSF provides serialization to prevent data loss from attempts at concurrent access, but your installation is responsible for the effective management of concurrent use of competing operations. Consult your system administrator or system programmer for your installation guidelines.

The CKDS Key Record Create2, CKDS Key Record Read2, and CKDS Key Record Write2 callable services must be used for variable-length key tokens. These services also support existing DES and AES tokens.

CKDS Key Record Create callable service (CSNBKRC and CSNEKRC)

This service accepts a key label and creates a null key record in both the DASD copy and in-storage copy of the CKDS. The record contains a key token set to binary zeros and is identified by the key label passed in the call statement. The key label must be unique.

Prior to updating a key record using either the dynamic CKDS update services or KGUP, that record must already exist in the CKDS. You can use either the CKDS key record create service, KGUP, or your key entry hardware to create the initial record in the CKDS.

CKDS Key Record Create2 callable service (CSNBKRC2 and CSNEKRC2)

This service accepts a key label and optionally, a symmetric key token, and creates a key record in both the DASD copy and in-storage copy of the CKDS. The record contains the supplied key token or a null key token and is identified by the key label passed in the call statement. The key label must be unique.

This service must be used with variable-length key tokens. This service supports all symmetric key tokens.

CKDS Key Record Delete callable service (CSNBKRD and CSNEKRD)

This service accepts a unique key label and deletes the associated key record from both the in-storage and DASD copies of the CKDS. This service deletes the entire record, including the key label from the CKDS.

CKDS Key Record Read callable service (CSNBKRR and CSNEKRR)

This service copies an internal key token from the in-storage CKDS to the application storage, where it may be used directly in other cryptographic services. Key labels specified with this service must be unique.

CKDS Key Record Read2 callable service (CSNBKRR2 and CSNEKRR2)

This service copies an internal key token from the in-storage CKDS to the application storage, where it may be used directly in other cryptographic services. Key labels specified with this service must be unique.

This service must be used with variable-length key tokens. This service supports all symmetric key tokens.

CKDS Key Record Write callable service (CSNBKRW and CSNEKRW)

This service accepts an internal key token and a label and writes the key token to the CKDS record identified by the key label. The key label must be unique. Application calls to this service write the key token to both the DASD copy and in-storage copy of the CKDS, so the record must already exist in both copies of the CKDS.

CKDS Key Record Write2 callable service (CSNBKRW2 and CSNEKRW2)

This service accepts an internal key token and a label and writes the key token to the CKDS record identified by the key label. The key label must be unique. Application calls to this service write the key token to both the DASD copy and in-storage copy of the CKDS, so the record must already exist in both copies of the CKDS.

This service must be used with variable-length key tokens. This service supports all symmetric key tokens.

Coordinated KDS Administration callable service (CSFCRC and CSFCRC6)

This service is used to perform the following functions: coordinated CKDS change master key, coordinated CKDS refresh, coordinated PKDS change master key, coordinated PKDS refresh, and coordinated TKDS change master key.

While this service is performing a coordinated change master key function, dynamic KDS update services may continue to run in parallel. During a coordinated refresh operation, dynamic KDS update services may continue to be enabled; however, they will be temporarily suspended internally until the coordinated refresh completes. If this cannot be tolerated, it is recommended to disable dynamic KDS update services when using this service.

In a sysplex environment, this callable service is executed from a single ICSF instance, and the operation is coordinated across all sysplex members sharing the same active KDS. This removes the need for KDS

refresh or KDS change master key functions to be performed locally on every ICSF instance sharing the same active KDS in a sysplex environment.

ICSF Multi-Purpose Service callable service (CSFMPS and CSFMPS6)

This service is used to validate the keys in the active CKDS or PKDS. Use the ICSF multi-purpose callable service prior to a change master key operation as a way to detect keys that may cause a change master key operation to fail.

Key Data Set List callable service (CSFKDSL and CSFKDL6)

This service is used to list the labels of records in the active CKDS and PKDS that match selected metadata and other search criteria. This service is used to list the handles of records in the active TKDS that match selected metadata and other search criteria.

Key Data Set Metadata Read callable service (CSFKDMR and CSFKDMR6)

This service is used to read the metadata of a single record in the active CKDS, PKDS, or TKDS. Multiple metadata fields may be read in one call.

Key Data Set Metadata Write callable service (CSFKDMW and CSFKDMW6)

This service is used to add, delete, or change the metadata of a list of records in the active CKDS, PKDS, or TKDS. Multiple metadata fields may be changed in one call.

Callable Services that support Secure Sockets Layer (SSL)

The Secure Sockets Layer (SSL) protocol, developed by Netscape Development Corporation, provides communications privacy over the Internet. Client/server applications can use the SSL protocol to provide secure communications and prevent eavesdropping, tampering, or message forgery.

ICSF provides callable services that support the RSA-encryption and RSA-decryption of PKCS 1.2-formatted symmetric key data to produce symmetric session keys. These session keys can then be used to establish an SSL session between the sender and receiver.

PKA Decrypt Callable Service (CSNDPKD)

The PKA decrypt callable service uses the corresponding private RSA key to unwrap the RSA-encrypted key and deformat the key value. This service then returns the clear key value to the application.

PKA Encrypt Callable Service (CSNDPKE)

The PKA encrypt callable service encrypts a supplied clear key value under an RSA public key. Currently, the supplied key can be formatted using the PKCS 1.2 or ZERO-PAD methods prior to encryption.

Enciphering and deciphering data

The encipher and decipher callable services protect data off the host. ICSF protects sensitive data from disclosure to people who do not have authority to access it. Using algorithms that make it difficult and expensive for an unauthorized user to derive the original clear data within a practical time period assures privacy.

To protect data, ICSF can use the Data Encryption Standard (DES) algorithm or the Advanced Encryption Standard (AES) algorithm to encipher or decipher data or keys. The DES algorithm is documented in *Federal Information Processing Standard #46*. The AES algorithm is documented in *Federal Information Processing Standard #197*.

These services can be used to protect data:

- Decipher callable service (CSNBDEC, CSNBDEC1, CSNEDEC and CSNEDEC1)

The decipher callable service uses encrypted DES data-encrypting keys to decipher data.

- Encipher callable service (CSNBENC, CSNBENC1, CSNEENC and CSNEENC1)

  The encipher callable service uses encrypted DES data-encrypting keys to encipher data.
- Field Level Decipher (CSNBFLD and CSNEFLD)

  The Field Level Decipher callable service decrypts payment related database fields that have been previously encrypted using the field level encipher callable service.
- Field Level Encipher (CSNBFLE and CSNEFLE)

  The Field Level Encipher callable service encrypts payment related database fields, preserving the format of the fields.
- Symmetric Algorithm Decipher callable service (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)

  The Symmetric Algorithm Decipher callable service uses encrypted AES data-encrypting keys to decipher data.
- Symmetric Algorithm Encipher callable service (CSNBSAE, CSNBSAE1, CSNESAE and CSNESAE1)

  The Symmetric Algorithm Encipher callable service uses encrypted AES data-encrypting keys to encipher data.
- Symmetric Key Decipher callable service (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)

  The Symmetric Key Decipher callable service uses clear and encrypted AES and DES data-encrypting keys to decipher data.
- Symmetric Key Encipher callable service (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)

  The Symmetric Key Encipher callable service uses clear and encrypted AES and DES data-encrypting keys to encipher data.

Encoding and Decoding Data (CSNBECO, CSNEECO, CSNBDCO, and CSNEDCO)

The encode and decode callable services perform functions with clear keys. Encode enciphers 8 bytes of data using the electronic code book (ECB) mode of the DES and a clear key. Decode does the inverse of the encode service. These services are available only on a DES-capable system.

Translating Ciphertext (CSNBCTT2 or CSNBCTT3 and CSNECTT2 or CSNECTT3)

Restriction: These services are only available on the IBM zEnterprise EC12 or later servers.

ICSF provides a ciphertext translate callable service. It deciphers encrypted data (ciphertext) under one encryption key and reenciphers it under another key without having the data appear in the clear outside the cryptographic feature. Such a function is useful in a multiple node network, where sensitive data is passed through multiple nodes prior to reaching its final destination. Different nodes use different keys in the process. For more information about different nodes, see “Using the Cipher Text Translate2 callable service” on page 76.

The translate keys cannot be used for the encipher and decipher callable services.

Managing data integrity and message authentication

To ensure the integrity of transmitted messages and stored data, ICSF provides:

- Message authentication code (MAC).
- Several hashing functions, including modification detection code (MDC), SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512, RIPEMD-160, and MD5.

See Chapter 12, “Using digital signatures,” on page 1109 for an alternate method of message authentication using digital signatures.

The choice of callable service depends on the security requirements of the environment in which you are operating. If you need to ensure the authenticity of the sender and also the integrity of the data, consider message authentication code processing. If you need to ensure the integrity of transmitted data in an environment where it is not possible for the sender and the receiver to share a secret cryptographic key, consider hashing functions, such as the modification detection code process.

Message authentication code processing

The process of verifying the integrity and authenticity of transmitted messages is called *message authentication*. Message authentication code (MAC) processing allows you to verify that a message was not altered or a message was not fraudulently introduced onto the system. You can check that a message you have received is the same one sent by the message originator. The message itself may be in clear or encrypted form. The comparison is performed within the cryptographic feature. Since both the sender and receiver share a secret cryptographic key used in the MAC calculation, the MAC comparison also ensures the authenticity of the message.

In a similar manner, MACs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

ICSF provides support for the use of data-encrypting keys in the MAC generation and verification callable services, and also the use of a MAC generation key in the MAC Verify callable service. This support permits ICSF MAC services to interface more smoothly with non-CCA key distribution system.

HMAC Generate callable service (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)

When a message is sent, an application program can generate an authentication code for it using the HMAC Generate callable service. The callable service computes the message authentication code using FIPS-198 Keyed-Hash Message Authentication Code method.

HMAC Verify callable service (CSNBHMV or CSNBHMV1 and CSNEHMV or CSNEHMV1)

When the receiver gets the message, an application program calls the HMAC Verify callable service. The callable service verifies a MAC by generating another MAC and comparing it with the MAC received with the message. If the two codes are the same, the message sent was the same one received. A return code indicates whether the MACs are the same.

The MAC Verify callable service can use FIPS-198 Keyed-Hash Message Authentication Code method.

MAC Generate callable service (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)

When a message is sent, an application program can generate an authentication code for it using the MAC Generate callable service. The callable service computes the message authentication code using one of these methods:

- Using the ANSI X9.9-1 single key algorithm, a single-length MAC generation key or data-encrypting key, and the message text.
- Using the ANSI X9.19 optional double key algorithm, a double-length or triple-length MAC generation key and the message text.
- Using Europay, MasterCard and Visa (EMV) padding rules with a single-length MAC key or double-length MAC key and the message text.
- Using ISO 16609 algorithm with a double-length or triple-length MAC or a double-length or triple-length DATA key and the message text.

- NIST SP800-38B (2005) procedure and Triple-DES encryption in CBC mode.

ICSF allows a MAC to be the leftmost 32 or 48 bits of the last block of the ciphertext or the entire last block (64 bits) of the ciphertext. The originator of the message sends the message authentication code with the message text.

MAC Generate2 callable service (CSNBMGN2 or CSNBMGN3 and CSNEMGN2 or CSNEMGN3)

When a message is sent, an application program can generate an authentication code for it using the MAC Generate2 callable service.

The callable service computes the message authentication code using FIPS-198 Keyed-Hash Message Authentication Code method for HMAC key or the CMAC (NIST SP 800-38B) algorithm for AES keys.

MAC Verify callable service (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)

When the receiver gets the message, an application program calls the MAC Verify callable service. The callable service verifies a MAC by generating another MAC and comparing it with the MAC received with the message. If the two codes are the same, the message sent was the same one received. A return code indicates whether the MACs are the same.

The MAC Verify callable service can use either of these methods to generate the MAC for authentication:

- The ANSI X9.9-1 single key algorithm, a single-length MAC verification or MAC generation key (or a data-encrypting key), and the message text.
- The ANSI X9.19 optional double key algorithm, a double-length or triple-length MAC verification or MAC generation key and the message text.
- Using Europay, MasterCard and Visa (EMV) padding rules with a single-length MAC key or double-length MAC key and the message text.
- Using ISO 16609 algorithm with a double-length or triple-length MAC or a double-length or triple-length DATA key and the message text.
- NIST SP800-38B (2005) procedure and Triple-DES encryption in CBC mode.

The method used to verify the MAC should correspond with the method used to generate the MAC.

MAC Verify2 callable service (CSNBMVR2 or CSNBMVR3 and CSNEMVR2 or CSNEMVR3)

When the receiver gets the message, an application program calls the MAC Verify2 callable service. The callable service verifies a MAC by generating another MAC and comparing it with the MAC received with the message. If the two codes are the same, the message sent was the same one received. A return code indicates whether the two MACs are the same.

The MAC Verify callable service can use FIPS-198 Keyed-Hash Message Authentication Code method for HMAC key or the CMAC (NIST SP 800-38B) algorithm for AES keys.

Symmetric MAC Generate Callable Service (CSNBSMG, CSNBSMG1, CSNESMG and CSNESMG1)

This service supports generating a MAC using a clear AES key. The algorithms supported are CBC-MAC and XCBC-MAC (AES-XCBC-MAC-96 and AES-XCBC-PRF-128)

Symmetric MAC Verify Callable Service (CSNBSMV, CSNBSMV1, CSNESMV and CSNESMV1)

This service supports verifying a MAC using a clear AES key. The algorithms supported are CBC-MAC and XCBC-MAC (AES-XCBC-MAC-96 and AES-XCBC-PRF-128)

Hashing functions

Hashing functions include one-way hash generation and modification detection code (MDC) processing.

One-Way Hash Generate Callable Service (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1)

This service hashes a supplied message. Supported hashing methods include:

- SHA-1[1]
- SHA-224
- SHA-256
- SHA-384
- SHA-512
- SHA3-224
- SHA3-256
- SHA3-384
- SHA3-512
- SHAKE128
- SHAKE256
- RIPEMD-160
- MD5

MDC Generate callable service (CSNBMDG or CSNBMDG1 and CSNEMDG or CSNEMDG1)

The modification detection code (MDC) provides a form of support for data integrity. The MDC allows you to verify that data was not altered during transmission or while in storage. The originator of the data ensures that the MDC is transmitted with integrity to the intended receiver of the data. For instance, the MDC could be published in a reliable source of public information. When the receiver gets the data, an application program can generate an MDC, and compare it with the original MDC value. If the MDC values are equal, the data is accepted as unaltered. If the MDC values differ the data is assumed to be bogus.

Supported hashing methods through the MDC Generate callable service are:

- MDC-2
- MDC-4
- PADMDC-2
- PADMDC-4

In a similar manner, MDCs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

When data is sent, an application program can generate a modification detection code for it using the MDC Generate callable service. The callable service computes the modification detection code by encrypting the data using a publicly-known cryptographic one-way function. The MDC is a 128-bit value that is easy to compute for specific data, yet it is hard to find data that will result in a given MDC.

Once an MDC has been established for a file, the MDC generate service can be run at any other time on the file. The resulting MDC can then be compared with the previously established MDC to detect deliberate or inadvertent modification.

[1] The Secure Hash Algorithm (SHA) is also called the Secure Hash Standard (SHS), which Federal Information Processing Standard (FIPS) Publication 180 defines.

Managing personal authentication

The process of validating personal identities in a financial transaction system is called *personal authentication*. The personal identification number (PIN) is the basis for verifying the identity of a customer across the financial industry networks. ICSF checks a customer-supplied PIN by verifying it using an algorithm. The financial industry needs functions to generate, translate, and verify PINs. These functions prevent unauthorized disclosures when organizations handle personal identification numbers.

ICSF supports these algorithms for generating and verifying personal identification numbers:

- IBM 3624
- IBM 3624 PIN offset
- IBM German Bank Pool
- VISA PIN validation value
- Interbank

With ICSF, you can translate PIN blocks from one format to another. ICSF supports these formats:

- ANSI X9.8
- ISO formats 0, 1, 2, 3, 4
- VISA formats 1, 2, 3, 4
- IBM 4704 Encrypting PINPAD format
- IBM 3624 formats
- IBM 3621 formats
- ECI formats 1, 2, 3

With the capability to translate personal identification numbers into different PIN block formats, you can use personal identification numbers on different systems.

Verifying credit card data

The Visa International Service Association (VISA) and MasterCard International, Incorporated have specified a cryptographic method to calculate a value that relates to the personal account number (PAN), the card expiration date, and the service code. The VISA card-verification value (CVV) and the MasterCard card-verification code (CVC) can be encoded on either track 1 or track 2 of a magnetic striped card and are used to detect forged cards. Because most online transactions use track-2, the ICSF callable services generate and verify the CVV[2] by the track-2 method.

The VISA CVV generate callable service calculates a 1- to 5-byte value through the DES-encryption of the PAN, the card expiration date, and the service code using two data-encrypting keys or two MAC keys. The VISA CVV service verify callable service calculates the CVV by the same method, compares it to the CVV supplied by the application (which reads the credit card's magnetic stripe) in the *CVV_value*, and issues a return code that indicates whether the card is authentic.

Authentication Parameter Generate (CSNBAPG and CSNEAPG)

This callable service will calculate an authentication parameter (AP) and optionally return it encrypted under an encrypting key.

Clear PIN Encrypt Callable Service (CSNBCPE and CSNECPE)

To format a PIN into a PIN block format and encrypt the results, use the Clear PIN Encrypt callable service. You can also use this service to create an encrypted PIN block for transmission. With the RANDOM keyword, you can have the service generate random PIN numbers. An enhanced PIN security

[2] The VISA CVV and the MasterCard CVC refer to the same value. CVV is used here to mean both CVV and CVC.

mode is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. See “Clear PIN Encrypt (CSNBCPE and CSNECPE)” on page 624 for more information.

Clear PIN Generate Alternate Callable Service (CSNBCPA and CSNECPA)

To generate a clear VISA PIN validation value from an encrypted PIN block, call the clear PIN generate alternate callable service. This service also supports the IBM-PINO algorithm to produce a 3624 offset from a customer selected encrypted PIN.

An enhanced PIN security mode is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “Clear PIN Generate Alternate (CSNBCPA and CSNECPA)” on page 634 for more information.

Clear PIN Generate Callable Service (CSNBPGN and CSNEPGN)

To generate personal identification numbers, call the Clear PIN generate callable service. Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the callable service generates a clear PIN, a PIN verification value, or an offset. The callable service can only execute in special secure mode, which is described in “Special secure mode” on page 10.

CVV Key Combine Callable Service (CSNBCKC and CSNECKC)

This callable service combines 2 single-length CCA internal key tokens into 1 double-length CCA key token containing a CVVKEY-A key type. This combined double-length key satisfies current VISA requirements and eases translation between TR-31 and CCA formats for CVV keys.

Encrypted PIN Generate callable service (CSNBEPG and CSNEEPG)

To generate personal identification numbers, call the Encrypted PIN Generate callable service. Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the callable service generates a PIN and using a PIN block format and the PIN encrypting key, formats and encrypts the PIN. An enhanced PIN security mode is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. See “Encrypted PIN Generate (CSNBEPG and CSNEEPG)” on page 671 for more information.

Encrypted PIN Translate Callable Service (CSNBPTR and CSNEPTR)

To translate a PIN from one PIN-encrypting key to another or from one PIN block format to another or both, call the Encrypted PIN translation callable service. You must identify the input PIN-encrypting key that originally enciphers the PIN. You also need to specify the output PIN-encrypting key that you want the callable service to use to encipher the PIN. If you want to change the PIN block format, specify a different output PIN block format from the input PIN block format. An enhanced PIN security mode is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. The enhanced security mode is also available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “Encrypted PIN Translate (CSNBPTR and CSNEPTR)” on page 678 for more information.

Encrypted PIN Translate2 Callable Service (CSNBPTR2 and CSNEPTR2)

This service has the same functionality as the Encrypted PIN Translate service with additional support for ISO-4 PIN blocks and AES PIN-encrypting keys. See “Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)” on page 685 for more information.

Encrypted PIN Translate Enhanced Callable Service (CSNBPTRE and CSNEPTRE)

To reformat a PIN block where the PAN data is encrypted using format preserving encryption, call the Encrypted PIN translation enhanced callable service. You must identify the input PIN-encrypting key that originally enciphers the PIN, the output PIN-encrypting key that you want the callable service to

use to encipher the PIN, and the key to decipher the PAN. All of these keys may be derived using a key-generating key. The PAN data cannot be changed when reformatting the PIN block. See “Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)” on page 701 for more information.

Encrypted PIN Verify Callable Service (CSNBPVR and CSNEPVR)

To verify a supplied PIN, call the Encrypted PIN verify callable service. You need to specify the supplied enciphered PIN, the PIN-encrypting key that enciphers it, and other relevant data. You must also specify the PIN verification key and PIN verification algorithm. It compares the two personal identification numbers; if they are the same, it verifies the supplied PIN. See Chapter 8, “Financial services,” on page 601 for additional information.

An enhanced PIN security mode is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “Encrypted PIN Verify (CSNBPVR and CSNEPVR)” on page 713 for more information.

Encrypted PIN Verify2 Callable Service (CSNBPVR2 and CSNEPVR2)

To verify a supplied PIN against a reference PIN, call the Encrypted PIN Verify2 callable service. You need to specify the supplied enciphered PIN block, the reference enciphered PIN block, the PIN-encrypting keys that encipher the blocks, and other relevant data. The service compares the two personal identification numbers and if they are the same, it verifies the supplied PIN. IBM 3624, ISO-0, ISO-1, ISO-2, ISO-3, and ISO-4 PIN block formats are supported. See Chapter 8, “Financial services,” on page 601 for additional information.

An enhanced PIN security mode is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “Encrypted PIN Verify2 (CSNBPVR2 and CSNEPVR2)” on page 720 for more information.

PIN Change/Unblock Callable Service (CSNBPCU and CSNEPCU)

To support PIN change algorithms specified in the VISA Integrated Circuit Card Specification, call the PIN change/unblock callable service.

An enhanced PIN security mode is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “PIN Change/Unblock (CSNBPCU and CSNEPCU)” on page 795 for more information.

Recover PIN From Offset (CSNBPFO and CSNEPFO)

This callable service will calculate an encrypted customer-entered PIN from a PIN generating key, account information, and an offset. The generated PIN is returned encrypted under a PIN encrypting key.

Transaction Validation Callable Service (CSNBTRV and CSNETRV)

To support generation and validation of American Express card security codes, call the transaction validation callable service.

Format preserving encryption

Format preserving encryption (FPE) is a method of encryption where the resulting cipher text has the same form as the input clear text. The form of the text can vary according to use and the application. One example is a 16 digit credit card number. After using FPE to encrypt a credit card number, the resulting cipher text is another 16 digit number. In this credit card number example, the output cipher text is limited to numeric digits only.

The FPE services require some knowledge of the input clear text character set in order to create the appropriate output ciphertext.

Format Preserving Algorithms Decipher (CSNBFFXD and CSNEFFXD)

This callable service decrypts payment card data using FFX algorithms.

Format Preserving Algorithms Encipher (CSNBFFXE and CSNEFFXE)

This callable service encrypts payment card data using FFX algorithms.

Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT)

This callable service translates payment card data from encryption under one key to encryption under another key using FFX algorithms.

FPE Decipher (CSNBFPED and CSNEFPED)

This callable service decrypts payment card data using Visa Data Secure Platform (Visa DSP) processing.

FPE Encipher (CSNBFPEE and CSNEFPEE)

This callable service encrypts payment card data using Visa Data Secure Platform (Visa DSP) processing.

FPE Translate (CSNBFPET and CSNEFPET)

This callable service translates payment card data from encryption under one key to encryption under another key using Visa Data Secure Platform (Visa DSP) processing.

EMV simplification services

EMV simplification services are provided to assist in implementing EMV processing. These services provide key management support for generating master keys and deriving session keys. They provide EMV processing and functions for MasterCard, EMV, and Visa standards.

For Visa, either the Cryptogram Version 10 or Cryptogram Version 18 key derivation is used. See Visa specification, Appendix D2. Padding is determined by the Cryptogram Version used.

For MasterCard, M/CHIP 2 key derivation is used. EMV padding rules are used.

For EMV, the session key derivation is used as described in EMV Book 2, Annex A1.3. EMV padding rules are used.

Master keys are:

Application Cryptogram Key (AC)
Used during EMV Transaction Processing (ARQC/ARPC).

Secure Messaging Authentication Key (MAC)
Used to provide integrity for EMV scripting.

Secure Confidentiality Key (ENC)
Used to provide confidentiality for EMV scripts containing PINs.

DATA Key (DATA)
Used to encrypt and decrypt data used in EMV Verification Functions.

Derive ICC Master Key callable service (CSNBDCM and CSNEDCM)

This service generates an ICC master key from an issuer master key. The ICC master keys are needed for ICC personalization, EMV transaction processing, and EMV scripting. Optionally, this service returns the ICC master key as an external token under a key-encrypting key (KEK).

Inputs are:

- Issuer master key (key token or CKDS label).
- PAN and PAN sequence number.

- Key-encrypting key to wrap the generated master key (optional).

Outputs are:

- Internal or external ICC master key token.

See "Derive ICC MK (CSNBDCM and CSNEDCM)" on page 136 for more information.

Derive Session Key callable service (CSNBDSK and CSNEDSK)

This service generates a session key from either the Issuer or ICC master key. Session keys are needed for EMV transaction processing or EMV scripting.

Inputs are:

- Issuer or ICC master key (key token or CKDS label).
- PAN, PAN sequence number, Application Transaction Counter (ATC), and unpredictable number.

Outputs are:

- Internal session key token.

See "Derive Session Key (CSNBDSK and CSNEDSK)" on page 143 for more information.

EMV Scripting callable service (CSNBESC and CSNEESC)

EMV Scripting is a mechanism for sending commands to a payment card. The commands are used to update card parameters including potentially the PIN. Commands may be encrypted for confidentiality or MAC'd for integrity or both.

Scripts are generated by the issuer, or the issuer's agent, when a transaction is received from a payment card. This service receives the script as input, encrypts it, MAC's it, or both, and then returns either the encrypted script, the MAC, or both. The output is intended to be sent back to the payment card along with the response.

This service provides the following functions:

- Scripting with integrity.
- Scripting with confidentiality (for protection of scripts that may or may not contain a PIN).
- Scripting with confidentiality and integrity.
- PIN change/unblock.

Inputs are:

- Issuer MAC and ENC master key or keys (key token or CKDS label).
- PAN, PAN sequence number, script message, Application Transaction Counter (ATC), and unpredictable number.
- PIN Block, PIN Key, and PIN Format (optional).

Outputs are:

- Script message.
- MAC (optional).

See "EMV Scripting Service (CSNBESC and CSNEESC)" on page 646 for more information.

EMV Transaction (ARQC/ARPC) callable service (CSNBEAC and CSNEEAC)

This service provides EMV Authorization Request Cryptogram (ARQC) and Authorization Response Cryptogram (ARPC) transaction processing.

An ARQC is generated by the EMV card upon request from the point of sales terminal to obtain authorization for payment. The ARQC is forwarded across the payment network to the issuer for verification. After the issuer has verified the ARQC, the issuer generates an ARPC (the response).

The ARPC is sent back through the payment network to the point of sales terminal to authorize the transaction.

This service performs the following EMV functions:

- Verification of the Authorization Request Cryptogram (ARQC).
- Generation of the Authorization Response Cryptogram (ARPC).
- Both operations combined: Verify the ARQC and generate the ARPC.

Inputs are:

- Issuer AC master key or keys (key token or CKDS label).
- PAN, PAN sequence number, Cryptogram Information, Application Transaction Counter (ATC), Authorization Response Code (ARC) or Card Status Updates (CSU), ARQC, and unpredictable number.

Output is:

- ARPC.

See "EMV Transaction (ARQC/ARPC) Service (CSNBEAC and CSNEEAC)" on page 657 for more information.

EMV Verification callable service (CSNBEVF and CSNEEVF)

This service provides the following additional functions used by MasterCard:

- Verification of data authentication codes.
- Verification of ICC dynamic numbers.
- Decryption of encrypted counters.

Inputs are:

- Issuer master key (key token or CKDS label).
- PAN, PAN sequence number, data field, Application Transaction Counter (ATC), and unpredictable number.

Outputs are:

- Return and reason code.
- Decrypted counters.

See "EMV Verification Functions (CSNBEVF and CSNEEVF)" on page 665 for more information.

Generate Issuer Master Key callable service (CSNBGIM and CSNEGIM)

This service is intended to help with the initial steps of EMV setup by generating and storing the issuer master keys in the CKDS. Optionally, the keys can be returned as external tokens under a key-encrypting key (KEK) that is shared with the ICC personalization system.

Inputs are:

- CKDS label for the issuer master key.
- Key-encrypting key (KEK) to wrap the generated master key (optional).

Outputs are:

- Issuer master key in the CKDS.
- Internal or external issuer master key token.

See "Generate Issuer MK (CSNBGIM and CSNEGIM)" on page 197 for more information.

ANSI X9.143 (TR-31) key block support

A X9.143 (TR-31) key block is a format defined by the American National Standards Institute (ANSI) to support the interchange of keys in a secure manner with key attributes included in the exchanged data. The TR-31 key block format has a set of defined key attributes that are securely bound to the key so that they can be transported together between any two systems that both understand the TR-31 format. ICSF enables applications to convert a CCA token to a TR-31 key block for export to another party, and to convert an imported TR-31 key block to a CCA token. This enables you to securely exchange keys and their attributes with non-CCA systems.

ICSF enables applications to generate and use operational (internal) TR-31 key blocks. TR-31 Create callable service allows keys and key pairs to be generated for use with ICSF callable services. See Table 9 on page 29 for the list of services that support operational key blocks.

Although there is often a one-to-one correspondence between TR-31 key attributes and the attributes defined by CCA, there are also cases where the correspondence is many-to-one or one-to-many. Because there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the TR-31 Translate callable service and the TR-31 Import callable service provide rule array keywords that enable an application to specify the attributes to attach to the exported or imported key.

The TR-31 key block format defines a header section. The header contains metadata about the key, including its usage attributes. The header can also be extended with optional blocks, which can either have standardized content or proprietary information. Callable services are also provided for retrieving standard header or optional block information from a TR-31 key block without importing the key and for building an optional block.

Support for HMAC keys is introduced by APAR OA58880 for ICSF FMID HCR77D1 and later releases and licensed internal code for IBM z13, IBM z13s, and later servers. Standard ISO:20038 defines a key wrapping method for AES key-encrypting keys and the TR-31 Translate and TR-31 Import services have been updated to support the transport of HMAC keys using this method.

TR-31 Create Callable Service (CSNBT31C and CSNET31C)

The TR-31 Create service can be used to randomly generate AES, DES, and HMAC keys and key pairs in TR-31 blocks or create a skeleton TR-31 key block header, as defined in the ANSI X9.143 standard.

The service is available with the CCA release 8.1 or later licensed internal code on a CEX8 or later adapter on an IBM z16 or later server.

TR-31 Import Callable Service (CSNBT31I and CSNET31I)

The TR-31 Import callable service converts a TR-31 key block to a CCA token. Since there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the caller may need to specify the attributes to attach to the imported key through the rule array.

TR-31 Parse Callable Service (CSNBT31P and CSNET31P)

The TR-31 Parse callable service retrieves standard header information from a TR-31 key block without importing the key. This callable service can be used with the TR-31 Optional Data Read callable service to obtain both the standard header fields and any optional data blocks from the key block.

TR-31 Optional Data Read Callable Service (CSNBT31R and CSNET31R)

A TR-31 key block can hold optional fields which are securely bound to the key block using the integrated MAC. The optional blocks may either contain information defined in the TR-31 standard, or they may contain proprietary data. A separate range of optional block identifiers is reserved for use with proprietary blocks. Applications can call the TR-31 Optional Data Read callable service to obtain lists of the optional block identifiers and optional block lengths, and to obtain the data for a particular optional block. This callable service is often used in conjunction with the TR-31 Parse Callable Service which can be used to determine the number of optional blocks in the TR-31 token.

TR-31 Optional Data Build Callable Service (CSNBT31O and CSNET31O)

The TR-31 Optional Data Build callable service constructs the optional block data structure for a TR-31 key block. It builds the structure by adding one optional block with each call, until your entire set of optional blocks have been added. With each call, the application program provides a single optional block by specifying its ID, its length, and its data. Each subsequent call appends the current optional block to any pre-existing blocks.

TR-31 Translate Callable Service (CSNBT31X and CSNET31X)

The TR-31 Translate callable service can:

1. Translate a CCA token into a TR-31 key block.
2. Translate a TR-31 key block between internal and external formats.
3. Perform a compliance check or compliance tagging of a TR-31 key block.

Because there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the caller may need to specify the attributes to attach to the exported key through the rule array.

Secure messaging

These services will assist applications in encrypting secret information such as clear keys and PIN blocks in a secure message. These services will execute within the secure boundary of the CCA coprocessor.

The Secure Messaging for Keys callable service encrypts a text block, including a clear key value decrypted from an internal or external DES token.

The Secure Messaging for PINs callable service encrypts a text block, including a clear PIN block recovered from an encrypted PIN block.

Trusted Key Entry (TKE) support

The Trusted Key Entry (TKE) workstation is an optional feature. It offers an alternative to clear key entry. You can use the TKE workstation to load master keys, and operational keys in a *secure* way.

You can load keys remotely and for multiple cryptographic coprocessors. The TKE workstation eases the administration for using one cryptographic coprocessor as a production machine and as a test machine at the same time, while maintaining security and reliability.

The TKE workstation can be used for enabling/disabling access control points for callable services executed on cryptographic coprocessors. See Appendix G, “Access control points and callable services,” on page 1661 for additional information.

For complete details about the TKE workstation see *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*.

TKE Version 6.0 or later is required if using a CEX3C.

TKE Version 7.2 or later is required if using a CEX4C.

TKE Version 8.0 or later is required if using a CEX5C.

TKE Version 9.0 or later is required if using a CEX6C.

TKE Version 9.2 or later is required if using a CEX7C.

TKE Version 10.0 or later is required if using a CEX8C.

On IBM z9 EC, IBM z9 BC, IBM z10 EC and IBM z10 BC systems running with May 2004 or higher version of Licensed Internal Code or an IBM z9 EC, IBM z9 BC, IBM z10 EC and IBM z10 BC with MCL 029

Stream J12220 or higher of Licensed Internal Code, you must enable TKE commands for each CCA Crypto Express coprocessor from the Support Element. This is true for new TKE users and those upgrading from TKE V4.*x* or V5.*x* when the new LIC is installed. See *Support Element Operations Guide* and *z/OS Cryptographic Services ICSF TKE Workstation User's Guide* for more information.

Utilities

ICSF provides these utilities.

Character/Nibble Conversion Callable Services (CSNBXBC and CSNBXCB)

The character/nibble conversion callable services are utilities that convert a binary string to a character string and vice versa.

Code Conversion Callable Services (CSNBXEA and CSNBXAE)

The code conversion callable services are utilities that convert EBCDIC data to ASCII data and vice versa.

Cryptographic Usage Statistic (CSFSTAT and CSFSTAT6)

The callable service tracks cryptographic usage external to the ICSF address space.

ICSF Query Algorithm Callable Service (CSFIQA)

The callable service provides information regarding the cryptographic and hash algorithms available.

ICSF Query Facility Callable Service (CSFIQF)

The callable service provides ICSF status information, as well as coprocessor information.

ICSF Query Facility2 Callable Service (CSFIQF2)

The callable service provides status information on the cryptographic environment as currently know by ICSF. This callable service is not SAF protected nor does it call any cryptographic processors.

X9.9 Data Editing Callable Service (CSNB9ED)

The data editing callable service is a utility that edits an ASCII text string according to the editing rules of ANSI X9.9-4.

Typical sequences of ICSF callable services

Sample sequences in which the ICSF callable services might be called are shown in Table 12 on page 73.

Table 12. Combinations of the callable services

Combinations

```
    Combination A (DATA keys only)     Combination B

1. Random number generate          1. Random number generate
2. Clear key import or             2. Secure key import or
   multiple clear key import          multiple secure key import
3. Encipher/decipher               3. Any service
4. Data key export or key export   4. Data key export for DATA keys, or
    (optional step)                   key export in the general case
                                       (optional step)

   Combination C                      Combination D

1. Key generate (OP form only)     1. Key generate (OPEX form)
2. Any service                     2. Any service
3. Key export (optional)

   Combination E                      Combination F

1. Key generate (IM form only)     1. Key generate (IMEX form)
2. Key import                      2. Key import
3. Any service                     3. Any service
4. Key export (optional)

   Combination G                      Combination H

1. Key generate                    1. Key import
2. Key record create               2. Key record create
3. Key record write                3. Key record write
4. Any service (passing label      4. Any service (passing label
   of the key just generated)         of the key just generated)
```

Notes:

1. An example of “any service” is CSNBENC.
2. These combinations exclude services that can be used on their own; for example, key export or encode, or using key generate to generate an exportable key.
3. These combinations do not show key communication, or the transmission of any output from an ICSF callable service.

The key forms are described in “Key Generate (CSNBKGN and CSNEKGN)” on page 211.

Key forms and types used in the Key Generate callable service

The Key Generate callable service is the most complex of all the ICSF callable services. This topic provides examples of the key forms and key types used in the Key Generate callable service.

Generating an operational key

To generate an operational key, choose one of these methods:

For operational keys
Call the Key Generate callable service (CSNBKGN). Table 78 on page 220 and Table 79 on page 221 show the key type and key form combinations for a single key and for a key pair.

For operational keys
Call the Random Number Generate Long callable service (CSNBRNGL) and specify the *form* parameter as RANDOM. For DES keys, specify ODD parity for a random number you intend to use as a key. For AES keys, any random number is permitted. Then pass the generated value to the Multiple Secure Key Import callable service (CSNBSKM) or Secure Key Import2 callable service (CSNBSKI2) with a required key type. The required key type is now in operational form.

This method requires ICSF to be in special secure mode. For more information about special secure mode, see "Special secure mode" on page 10.

For data-encrypting keys
Call the random number generate long callable service (CSNBRNGL) and for DES keys, specify the *form* parameter as ODD. Then pass the generated value to the Clear Key Import callable service (CSNBCKI) or the Multiple Clear Key Import callable service (CSNBCKM). The DATA key type is now in operational form.

You cannot generate a PIN verification (PINVER) key in operational form because the originator of the PIN generation (PINGEN) key generates the PINVER key in exportable form, which is sent to you to be imported.

Generating an importable key

To generate an importable key form, call the Key Generate callable service (CSNBKGN).

If you want a DATA, DECIPHER, ENCIPHER, MAC, PINGEN, DATAM, or DATAC key type in importable form, obtain it directly by generating a single key. If you want any other key type in importable form, request a key pair where either the first or second key type is importable (IM). Discard the generated key form that you do not need.

Generating an exportable key

To generate an exportable key form, call the Key Generate callable service (CSNBKGN).

If you want a DATA, DECIPHER, ENCIPHER, MAC, PINGEN, DATAM, or DATAC key type in exportable form, obtain it directly by generating a single key. If you want any other key type in exportable form, request a key pair where either the first or second key type is exportable (EX). Discard the generated key form that you do not need.

Examples of single-length keys in one form only

```
Key   Key
Form   1

OP   DATA  Encipher or decipher data. Use data key export or key export
           to send encrypted key to another cryptographic partner. Then
           communicate the ciphertext.
OP   MAC   MAC generate. Because no MACVER key exists, there is no
           secure communication of the MAC with another cryptographic
           partner.
IM   DATA  Key Import, and then encipher or decipher. Then key export
           to communicate ciphertext and key with another cryptographic
           partner.
EX   DATA  You can send this key to a cryptographic partner, but you
           can do nothing with it directly. Use it for the key
           distribution service. The partner could then use key import
           to get it in operational form, and use it as in OP DATA
           above.
```

Examples of OPIM single-length, double-length, and triple-length keys in two forms

The first two letters of the key form indicate the form that key type 1 parameter is in, and the second two letters indicate the form that key type 2 parameter is in.

```
Key   Type Type
Form   1    2

OPIM  DATA DATA  Use the OP form in encipher. Use key export with the
                 OP form to communicate ciphertext and key with
                 another cryptographic partner. Use key import at a
                 later time to use encipher or decipher with the same
                 key again.
OPIM  MAC  MAC   Single-length MAC generation key. Use the OP form in
                 MAC generation. You have no corresponding MACVER key,
```

```
                  but you can call the MAC verification service with
                  the MAC key directly. Use the key import callable
                  service and then compute the MAC again using the MAC
                  verification callable service, which compares the MAC
                  it generates with the MAC supplied with the message
                  and issues a return code indicating whether they
                  compare.
```

Examples of OPEX single-length, double-length, and triple-length keys in two forms

```
Key   Type  Type
Form  1      2

OPEX  DATA  DATA    Use the OP form in encipher. Send the EX form and
                    the ciphertext to another cryptographic partner.
OPEX  MAC   MAC     Single-length MAC generation key. Use the OP form in
                    both MAC generation and MAC verification. Send the
                    EX form to a cryptographic partner to be used in the
                    MAC generation or MAC verification services.
OPEX  MAC   MACVER  Single-length MAC generation and MAC verification
                    keys. Use the OP form in MAC generation. Send the EX
                    form to a cryptographic partner where it will be put
                    into key import, and then MAC verification, with the
                    message and MAC that you have also transmitted.
OPEX  PINGEN PINVER Use the OP form in Clear PIN generate. Send the
                    EX form to a cryptographic partner where it is put
                    into key import, and then Encrypted PIN verify,
                    along with an IPINENC key.
OPEX  IMPORTER EXPORTER
                    Use the OP form in key import, key generate,
                    or secure key import. Send the EX form to a
                    cryptographic partner where it is used in key
                    export, data key export, or key generate, or put in
                    the CKDS.
OPEX  EXPORTER IMPORTER
                    Use the OP form in key export, data key export,
                    or key generate. Send the EX form to a cryptographic
                    partner where it is put into the CKDS or used in key
                    import, key generate or secure key import.
```

When you and your partner have the OPEX IMPORTER EXPORTER, OPEX EXPORTER IMPORTER pairs of keys in “Examples of OPEX single-length, double-length, and triple-length keys in two forms” on page 75 installed, you can start key and data exchange.

Examples of IMEX single-length and double-length keys in two forms

```
Key    Type    Type
Form   1        2

IMEX   DATA    DATA      Use the key import callable service to import
                         IM form and use the OP form in encipher. Send
                         the EX form to a cryptographic partner.
IMEX  MAC      MACVER    Use the key import callable service to import
                         the IM form and use the OP form in MAC
                         generate. Send the EX form to a cryptographic
                         partner who can verify the MAC.
IMEX IMPORTER  EXPORTER  Use the key import callable service to import
                         the IM form and send the EX form to a
                         cryptographic partner. This establishes a new
                         IMPORTER/EXPORTER key between you and your
                         partner.
IMEX  PINGEN   PINVER    Use the key import callable service to import
                         the IM form and send the EX form to a
                         cryptographic partner. This establishes a new
                         PINGEN/PINVER key between you and your partner.
```

Examples of EXEX single-length and double-length keys in two forms

For the keys shown in this list, you are providing key distribution services for other nodes in your network, or other cryptographic partners. Neither key type can be used in your installation.

```
Key    Type     Type
Form    1        2

EXEX  DATA     DATA     Send the first EX form to a cryptographic
EXEX  MAC      MACVER   partner with the corresponding IMPORTER and
EXEX  IMPORTER EXPORTER send the second EX form to another
EXEC  OPINENC  IPINENC  cryptographic partner with the corresponding
                        IMPORTER. This exchange establishes a key
                        between two partners.
```

Using the Cipher Text Translate2 callable service

Restriction: The Cipher Text Translate2 callable service is only available on the IBM zEnterprise EC12 and later servers.

This topic describes a scenario using the encipher, Cipher Text Translate2, and decipher callable services with four network nodes: A, B, C, and D. You want to send data from your network node A to a destination node D. You cannot communicate directly with node D, and nodes B and C are situated between you. You do not want nodes B and C to decipher your data.

At node A, you use the Encipher callable service. Node D uses the Decipher callable service.

Node B and C will use the Cipher Text Translate2 callable service. Consider the keys that are needed to support this process:

1. At your node, generate one key in two forms: OPEX CIPHER CIPHERXI
2. Send the exportable CIPHERXI key to node B.
3. Node B and C need to share a key, so generate **a different key** in two forms: EXEX CIPHERXO CIPHERXI.
4. Send the exportable CIPHERXO key to node B.
5. Send the exportable CIPHERXI key to node C.
6. Node C and node D need to share a CIPHERXO key and a CIPHER key. Node D can generate one key in two forms: OPEX CIPHERXO CIPHERXI.
7. Node D sends the exportable CIPHERXO key to node C.

The communication process is shown as:

```
Node:       A              B                    C                D

Callable
Service: Encipher  Ciphertext Translate  Ciphertext Translate   Decipher

Keys:     CIPHER    CIPHERXI  CIPHERXO    CIPHERXI  CIPHERXI     CIPHER

Key Pairs: |____ = ____|         |____ = ____|         |____ = ____|
```

Therefore, you need three keys, each in two different forms. You can generate two of the keys at node A, and node D can generate the third key. Note that the key used in the decipher callable service at node D is **not** the same key used in the encipher callable service at node A.

Summary of callable services

Table 13 on page 77 lists the callable services described in this publication, and their corresponding verbs. The figure also references the topic that describes the callable service.

Table 13. Summary of ICSF callable services		
Service	**Service name**	**Function**
Chapter 5, "Managing symmetric cryptographic keys," on page 115		
CSNBCKI CSNECKI	Clear Key Import	Imports an 8-byte clear DES DATA key, enciphers it under the master key, and places the result into an internal key token. CSNBCKI converts the clear key into operational form as a DATA key.
CSNBCVG CSNECVG	Control Vector Generate	Builds a control vector from keywords specified by the *key_type* and *rule_array* parameters.
CSNBCVT CSNECVT	Control Vector Translate	Changes the control vector used to encipher an external DES key.
CSNBCVE CSNECVE	Cryptographic Variable Encipher	Uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. The plaintext must be a multiple of eight bytes in length.
CSNBDCM CSNEDCM	Derive ICC MK	Generates ICC master keys from issuer master keys. ICC master keys are needed for ICC personalization, EMV transaction processing, and EMV scripting. Optionally, this service returns the ICC master key as an external token wrapped under a key-encrypting key (KEK).
CSNBDDK CSNEDDK	Diversify Directed Key	Generate and derive keys using the direct key diversification key scheme.
CSNBDKG CSNEDKG	Diversified Key Generate	Generates a DES key based upon the key-generating key, the processing method, and the parameter data that is supplied.
CSNBDKG2 CSNEDKG2	Diversified Key Generate2	Generates an AES key based on a function of a key-generating key, the process rule, and data that you supply.
CSNBDKM CSNEDKM	Data Key Import	Imports an encrypted source DES DATA key and creates or updates a target internal key token with the master key enciphered source key.
CSNBDKX CSNEDKX	Data Key Export	Converts a DES DATA key from operational form into exportable form.
CSNBDSK CSNEDSK	Derive Session Key	Derives a session key from either an issuer master key or an ICC master key. The session key can be used for EMV transaction processing or EMV scripting.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNDEDH CSNFEDH	ECC Diffie-Hellman	Creates symmetric key material from a pair of ECC keys using the Elliptic Curve Diffie-Hellman protocol and the static unified model key agreement scheme or “Z” data (the “secret” material output from D-H process).
CSNBGIM CSNEGIM	Generate Issuer MK	Helps with the initial steps of EMV setup by generating and storing the issuer master keys. Optionally, the issuer master keys can be returned as external tokens wrapped under a key-encrypting key (KEK) that is shared with the ICC personalization system.
CSNBKET CSNEKET	Key Encryption Translate	Change the encryption of DES key material in a key token wrapped with ECB (legacy method) and key material wrapped with CBC.
CSNBKEX CSNEKEX	Key Export	Converts any DES key from operational form into exportable form. (However, this service does not export a key that was marked non-exportable when it was imported.)
CSNBKGN CSNEKGN	Key Generate	Generates a 64-bit, 128-bit, or 192-bit odd parity DES key, or a pair of DES keys; and returns them in encrypted forms (operational, exportable, or importable). Generates a 128-bit, 192-bit, or 256-bit AES DATA key and returns them in operational form. CSNBKGN does not produce keys in clear.
CSNBKGN2 CSNEKGN2	Key Generate2	Generates a variable-length HMAC or AES key or a pair of keys; and returns them in encrypted forms (operational, exportable, or importable).
CSNBKIM CSNEKIM	Key Import	Converts any DES key from importable form into operational form.
CSNBKPI CSNEKPI	Key Part Import	Combines the clear key parts of any DES key type and returns the combined key value in an internal key token or an update to the CKDS.
CSNBKPI2 CSNEKPI2	Key Part Import2	Combines the clear key parts of an HMAC or AES key and returns the combined key value in an internal key token or an update to the CKDS.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNBKYT CSNEKYT CSNBKYTX CSNEKYTX	Key Test	Generates or verifies (depending on keywords in the rule array) a secure verification pattern for keys. CSNBKYT and CSNEKYT require the tested key to be in the clear or encrypted under the master key. CSNBKYTX and CSNEKYTX also allow the tested key to be encrypted under a key-encrypting key.
CSNBKYT2 CSNEKYT2	Key Test2	Generates or verifies (depending on keywords in the rule array) a secure verification pattern for keys. CSNBKYT2 and CSNEKYT2 allow the tested key to be in the clear or encrypted under the master key or a key-encrypting key.
CSNBKTB CSNEKTB	Key Token Build	Builds an internal or external token from the supplied parameters. You can use this service to build CCA key tokens for all DES key types ICSF supports.
CSNBKTB2 CSNEKTB2	Key Token Build2	Builds an internal or external key token from the supplied parameters. You can use this service to build CCA key tokens of all the AES and HMAC key types. You can use this callable service to build an internal clear key token for any key type for input to the key test2 callable service. You can use this callable service to build a skeleton token for input to the key generate2 and key part import2 callable services.
CSNBKTR CSNEKTR	Key Translate	Uses one key-encrypting key to decipher a DES input key and then enciphers this key using another key-encrypting key within the secure environment.
CSNBKTR2 CSNEKTR2	Key Translate2	Uses one key-encrypting key to decipher AES and HMAC input keys and then enciphers this key using another key-encrypting key within the secure environment. Converts an AES fixed-length DATA key to a variable-length CIPHER key. Supports applying the compliant tag to an existing token as well as checking if a legacy token can be successfully tagged.
CSNBCKM CSNECKM	Multiple Clear Key Import	Imports a single-, double-, or triple-length clear DES DATA key, enciphers it under the master key, and places the result into an internal key token. CSNBCKM converts the clear key into operational form as a DATA key.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNBSKM CSNESKM	Multiple Secure Key Import	Enciphers a single-, double-, or triple-length clear DES key under the master key or an input importer key, and places the result into an internal or external key token as any key type. Enciphers a clear AES DATA key under the master key. This service executes only in special secure mode.
CSNDPKD CSNFPKD	PKA Decrypt	Uses an RSA private key to decrypt the RSA-encrypted key value and return the clear key value to the application.
CSNDPKE CSNFPKE	PKA Encrypt	Encrypts a supplied clear key value under an RSA public key.
CSNBPEX CSNEPEX	Prohibit Export	Modifies an operational DES key so that it cannot be exported.
CSNBPEXX CSNEPEXX	Prohibit Export Extended	Changes the external token of a DES key in exportable form so that it can be imported at the receiver node but not exported from that node.
CSNBRKA CSNERKA	Restrict Key Attribute	Modifies an AES, HMAC, or DES key so that is cannot be exported. Modifies an AES or HMAC to restrict other attributes of the token.
CSNBRNG CSNERNG CSNBRNGL CSNERNGL	Random Number Generate Random Number Generate Long	Generates an 8-byte random number or a random number with a user-specified length. The output can be specified in three forms of parity: RANDOM, ODD, and EVEN.
CSNDRKX CSNFRKX	Remote Key Export	Generates or exports DES keys for local use and for distribution to an ATM or other remote device. RKX uses a special structure to hold encrypted symmetric keys in a way that binds them to the trusted block and allows sequences of RKX calls to be bound together as if they were an atomic operation.
CSNBSKI CSNESKI	Secure Key Import	Enciphers a clear key under the master key, and places the result into an internal or external key token as any DES key type. This service executes only in special secure mode.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNBSKI2 CSNESKI2	Secure Key Import2	Enciphers a variable-length clear HMAC or AES key under the master key, and places the result into an internal key token as any key type. Enciphers a variable-length clear HMAC or AES key under a key-encrypting key, and places the result into an external key token as any key type. This service executes only in special secure mode.
CSNDSYX CSNFSYX	Symmetric Key Export	Transfers an application-supplied symmetric key from encryption under the host master key to encryption under an application-supplied RSA public key or AES EXPORTER key. The application-supplied key must be an internal key token or the label in the CKDS of a DES DATA, AES DATA, or variable-length symmetric key token.
CSNDSXD CSNFSXD	Symmetric Key Export with Data	Export a symmetric key encrypted using an RSA key, inserted in a PKCS#1 block type 2, with some extra data supplied by the application.
CSNDSYG CSNFSYG	Symmetric Key Generate	Generates a symmetric DES DATA key and returns the key in two forms: enciphered under the DES master key or KEK and under an RSA public key. Generates a symmetric AES DATA key and returns the key in two forms: enciphered under the AES master key and under an RSA public key.
CSNDSYI CSNFSYI	Symmetric Key Import	Imports a symmetric key enciphered under an RSA public key into operational form enciphered under a host master key.
CSNDSYI2 CSNFSYI2	Symmetric Key Import2	Imports a symmetric key enciphered under an RSA public key or AES EXPORTER key into operational form enciphered under a host master key.
CSNDTBC CSNETBC	Trusted Block Create	Creates a trusted block in a two step process. The block will be in external form, encrypted under an IMP-PKA transport key. This means that the MAC key contained within the trusted block will be encrypted under the IMP-PKA key.
CSNBT31X CSNET31X	TR-31 Translate	Converts a CCA token to TR-31 format for export to another party.
CSNBT31I CSNET31I	TR-31 Import	Converts a TR-31 key block to a CCA token.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNBT31P CSNET31P	TR-31 Parse	Retrieves standard header information from a TR-31 key block without importing the key.
CSNBT31R CSNET31R	TR-31 Optional Data Read	Obtains lists of the optional block identifiers and optional block lengths, and obtains the data for a particular optional block.
CSNBT31O CSNET31O	TR-31 Optional Data Build	Constructs the optional block data structure for a TR-31 key block.
CSNBUKD CSNEUKD	Unique Key Derive	Derives a DES key using a base derivation key and derivation data. The following key types can be derived: • CIPHER • ENCIPHER • DECIPHER • MAC • MACVER • IPINENC • OPINENC • DATA token containing a PIN Key
Chapter 6, "Protecting data," on page 469		
CSNBCTT2 CSNBCTT3 CSNECTT2 CSNECTT3	Cipher Text Translate2	Translates the user-supplied ciphertext from one key and enciphers the ciphertext to another key. Supports both AES and DES algorithms. CSNBCTT2 and CSNECTT2 require the ciphertext to reside in the caller's primary address space. CSNBCTT3 and CSNECTT3 allow the ciphertext to reside in the caller's primary address space or in a z/OS data space.
CSNBDEC CSNEDEC CSNBDEC1 CSNEDEC1	Decipher	Deciphers data using the cipher block chaining mode of the DES. (The method depends on the token marking or keyword specification.) The result is called plaintext. CSNBDEC and CSNEDEC require the plaintext and ciphertext to reside in the caller's primary address space. CSNBDEC1 and CSNEDEC1 allow the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNBDCO CSNEDCO	Decode	Decodes an 8-byte string of data using the electronic code book mode of the DES. (This is for DES encryption only.)
CSNBENC CSNEENC CSNBENC1 CSNEENC1	Encipher	Enciphers data using the cipher block chaining mode of the DES. (The method depends on the token marking or keyword specification.) The result is called ciphertext. CSNBENC and CSNEENC require the plaintext and ciphertext to reside in the caller's primary address space. CSNBENC1 and CSNEENC1 allow the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.
CSNBECO CSNEECO	Encode	Encodes an 8-byte string of data using the electronic code book mode of the DES. (This is for DES encryption only.)
CSNBSAD CSNESAD CSNBSAD1 CSNESAD1	Symmetric Algorithm Decipher	Deciphers data using the AES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. CSNBSAD and CSNESAD require the plaintext and ciphertext to reside in the caller's primary address space. CSNBSAD1 and CSNESAD1 allows the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.
CSNBSAE CSNESAE CSNBSAE1 CSNESAE1	Symmetric Algorithm Encipher	Enciphers data using the AES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. CSNBSAE and CSNESAE require the plaintext and ciphertext to reside in the caller's primary address space. CSNBSAE1 and CSNESAE1 allows the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNBSYD CSNBSYD1 CSNESYD CSNESYD1	Symmetric Key Decipher	Deciphers data using the AES or DES algorithm in an address space or a data space using one of the supported modes. Both clear and secure keys are supported. CSNBSYD and CSNESYD require the plaintext and ciphertext to reside in the caller's primary address space. CSNBSYD1 and CSNESYD1 allow the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.
CSNBSYE CSNBSYE1 CSNESYE CSNESYE1	Symmetric Key Encipher	Enciphers data using the AES or DES algorithm in an address space or a data space using one of the supported modes. Both clear and secure keys are supported. CSNBSYE and CSNESYE require the plaintext and ciphertext to reside in the caller's primary address space. CSNBSYE1 and CSNESYE1 allows the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.
Chapter 7, “Verifying data integrity and authenticating messages,” on page 543		
CSNBHMG CSNEHMG CSNBHMG1 CSNEHMG1	HMAC Generate	Generates message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using the FIPS-198 Keyed-Hash Message Authentication Code algorithm. CSNBHMG and CSNEHMG require data to reside in the caller’s primary address space. CSNBHMG1 and CSNEHMG1 allow data to reside in the caller’s primary address space or in a z/OS data space.
CSNBHMV CSNEHMV CSNBHMV1 CSNEHMV1	HMAC Verify	Verifies message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using the FIPS-198 Keyed-Hash Message Authentication Code algorithm. CSNBHMV and CSNEHMV requires data to reside in the caller’s primary address space. CSNBHMV1 and CSNEHMV1 allows data to reside in the caller’s primary address space or in a z/OS data space.

Table 13. Summary of ICSF callable services (continued)

Service	Service name	Function
CSNBMGN CSNEMGN CSNBMGN1 CSNEMGN1	MAC Generate	Generates a 4-, 6-, or 8-byte message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using a DES key and DES based algorithms. CSNBMGN and CSNEMGN require data to reside in the caller's primary address space. CSNBMGN1 and CSNEMGN1 allow data to reside in the caller's primary address space or in a z/OS data space.
CSNBMGN2 CSNEMGN2 CSNBMGN3 CSNEMGN3	MAC Generate2	Generates a keyed hash message authentication code (HMAC) or a ciphered message authentication code (CMAC) for the message string provided as input.
CSNBMVR CSNEMVR CSNBMVR1 CSNEMVR1	MAC Verify	Verifies a 4-, 6-, or 8-byte message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using a DES key and DES based algorithms. CSNBMVR and CSNEMVR require data to reside in the caller's primary address space. CSNBMVR1 and CSNEMVR1 allow data to reside in the caller's primary address space or in a z/OS data space.
CSNBMVR2 CSNEMVR2 CSNBMVR3 CSNEMVR3	MAC Verify2	Verifies a keyed hash message authentication code (HMAC) or a ciphered message authentication code (CMAC) for the message text provided as input.
CSNBMDG CSNEMDG CSNBMDG1 CSNEMDG1	MDC Generate	Generates a 128-bit modification detection code (MDC) for a text string that the application program supplies. CSNBMDG and CSNEMDG require data to reside in the caller's primary address space. CSNBMDG1 and CSNEMDG1 allow data to reside in the caller's primary address space or in a z/OS data space.
CSNBOWH CSNEOWH CSNBOWH1 CSNEOWH1	One Way Hash Generate	Generates a one-way hash on specified text.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNBSMG, CSNESMG CSNBSMG1 CSNESMG1	Symmetric MAC Generate	Use the symmetric MAC generate callable service to generate a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using a clear AES key. CSNBSMG1 allows data to reside in the caller's primary address space or in a z/OS data space.
CSNBSMV, CSNESMV CSNBSMV1 CSNESMV1	Symmetric MAC Verify	Use the symmetric MAC verify callable service to verify a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using a clear AES key. CSNBSMV1 allows data to reside in the caller's primary address space or in a z/OS data space.
Chapter 8, "Financial services," on page 601		
CSNBAPG CSNEAPG	Authentication Parameter Generate	Generate an authentication parameter (AP) and optionally return it encrypted under a supplied encrypting key.
CSNBCPE CSNECPE	Clear PIN Encrypt	Formats a PIN into a PIN block format and encrypts the results.
CSNBPGN CSNEPGN	Clear PIN Generate	Generates a clear personal identification number (PIN), a PIN verification value (PVV), or an offset using one of these algorithms: IBM 3624 (IBM-PIN or IBM-PINO) IBM German Bank Pool (GBP-PIN) VISA PIN validation value (VISA-PVV) Interbank PIN (INBK-PIN) This service executes only in special secure mode.
CSNBCPA CSNECPA	Clear PIN Generate Alternate	Generates a clear VISA PIN validation value (PVV) from an input encrypted PIN block. The PIN block may have been encrypted under either an input or output PIN encrypting key. The IBM-PINO algorithm is supported to produce a 3624 offset from a customer selected encrypted PIN.
CSNBCKC CSNECKC	CVV Key Combine	Combines two single-length CCA internal key tokens into 1 double-length CCA key token containing a CVVKEY-A key type.

Table 13. Summary of ICSF callable services (continued)

Service	Service name	Function
CSNBDCM CSNEDCM	Derive ICC MK	Generates ICC master keys from issuer master keys. ICC master keys are needed for ICC personalization, EMV transaction processing, and EMV scripting. Optionally, this service returns the ICC master key as an external token wrapped under a key-encrypting key (KEK).
CSNBESC CSNEESC	EMV Scripting Service	Mechanism for sending commands to an EMV payment card. The commands are used to update card parameters including potentially the PIN. Commands may be encrypted for confidentiality or MAC'd for integrity or both.
CSNBEAC CSNEEAC	EMV Transaction (ARQC/ARPC) Service	Simplifies EMV Authorization Request Cryptogram (ARQC) and Authorization Response Cryptogram (ARPC) transaction processing.
CSNBEVF CSNEEVF	EMV Verification Functions	Provides additional functions used by MasterCard for their EMV cards in addition to application cryptograms and scripting.
CSNBEPG CSNEEPG	Encrypted PIN Generate	Generates and formats a PIN and encrypts the PIN block.
CSNBPTR CSNEPTR	Encrypted PIN Translate	Reenciphers a PIN block from one PIN-encrypting key to another and, optionally, changes the PIN block format. DUKPT keywords are supported.
CSNBPTR2 CSNEPTR2	Encrypted PIN Translate2	Reenciphers a PIN block from one PIN-encrypting key to another and, optionally, changes the PIN block format. Unique key per transaction key derivation is supported.
CSNBPTRE CSNEPTRE	Encrypted PIN Translate Enhanced	Reformat a PIN block where the PAN data is encrypted using format preserving encryption. Unique key per transaction key derivation is supported.
CSNBPVR CSNEPVR	Encrypted PIN Verify	Verifies a supplied PIN using one of these algorithms: IBM 3624 (IBM-PIN or IBM-PINO) IBM German Bank Pool (GBP-PIN) VISA PIN validation value (VISA-PVV) Interbank PIN (INBK-PIN) DUKPT keywords are supported.
CSNBPVR2 CSNEPVR2	Encrypted PIN Verify2	Verifies a supplied PIN against a reference PIN. DUKPT keywords are supported.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNBFLD CSNEFLD	Field Level Decipher	Decrypts payment related database fields, preserving the format of the fields using the VISA Format Preserving Encryption algorithm.
CSNBFLE CSNEFLE	Field Level Encipher	Encrypts payment related database fields, preserving the format of the fields using the VISA Format Preserving Encryption algorithm.
CSNBFFXD CSNEFFXD	Format Preserving Algorithms Decipher	Decrypts payment related database fields, preserving the format of the fields using NIST FFX algorithms.
CSNBFFXE CSNEFFXE	Format Preserving Algorithms Encipher	Encrypts payment related database fields, preserving the format of the fields using NIST FFX algorithms.
CSNBFFXT CSNEFFXT	Format Preserving Algorithms Translate	Translates payment card data from encryption under one key to encryption under another key using NIST FFX algorithms.
CSNBFPED CSNEFPED	FPE Decipher	Decrypts payment card data using Visa Data Secure Platform (Visa DSP) processing.
CSNBFPEE CSNEFPEE	FPE Encipher	Encrypts payment card data using Visa Data Secure Platform (Visa DSP) processing.
CSNBFPET CSNEFPET	FPE Translate	Translates payment card data from encryption under one key to encryption under another key using Visa Data Secure Platform (Visa DSP) processing.
CSNBPCU CSNEPCU	PIN Change/Unblock	Supports the PIN change algorithms specified in the VISA Integrated Circuit Card Specification.
CSNBPFO CSNEPFO	Recover PIN From Offset	Calculate an encrypted customer-entered PIN from a PIN generating key, account information, and an offset, returnining the PIN properly formatted and encrypted under a PIN encryption key.
CSNBSKY CSNESKY	Secure Messaging For Keys	Encrypts a text block, including a clear key value decrypted from an internal or external DES token.
CSNBSPN CSNESPN	Secure Messaging For PINs	Encrypts a text block, including a clear PIN block recovered from an encrypted PIN block.
CSNDSBC CSNFSBC	SET Block Compose	Composes the RSA-OAEP block and the DES-encrypted block in support of the SET protocol.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNDSBD CSNFSBD	SET Block Decompose	Decomposes the RSA-OAEP block and the DES-encrypted block to provide unencrypted data back to the caller.
CSNBTRV CSNETRV	Transaction Validation	Supports the generation and validation of American Express card security codes.
CSNBCSG CSNECSG	VISA CVV Service Generate	Generates a VISA Card Verification Value (CVV) or a MasterCard Card Verification Code (CVC).
CSNBCSV CSNECSV	VISA CVV Service Verify	Verifies a VISA Card Verification Value (CVV) or a MasterCard Card Verification Code (CVC).
Chapter 9, "Financial services for DK PIN methods," on page 849		
CSNBDCU2 CSNEDCU2	DK PRW Card Number Update2	Generates a PIN reference value (PRW) when a replacement card is being issued.
CSNBDDPG CSNEDDPG	DK Deterministic PIN Generate	Generates a PIN and PIN reference value (PRW) using an AES PIN calculation key.
CSNBDMP CSNEDMP	DK Migrate PIN	Generates the PIN reference value (PRW) for a specified user account.
CSNBDPMT CSNEDPMT	DK PAN Modify in Transaction	Generates a new PIN reference value (PRW) for an existing PIN when a merger has occurred and the account information has changed.
CSNBDPT CSNEDPT	DK PAN Translate	Creates an encrypted PIN block with the same PIN and a different PAN.
CSNBDPC CSNEDPC	DK PIN Change	Allows a customer to change their PIN to a value of their choosing.
CSNBDPV CSNEDPV	DK PIN Verify	Verifies ISO-1 and ISO-4 format PINs.
CSNBDPNU CSNEDPNU	DK PRW Card Number Update	Generates a PIN reference value (PRW) when a replacement card is being issued.
CSNBDPCG CSNEDPCG	DK PRW CMAC Generate	Generates a message authentication code (MAC) over specific values involved in an account number change transaction.
CSNBDRG2 CSNEDRG2	DK Random PIN Generate2	Generates a PIN and a PIN reference value using the random process.
CSNBDRPG CSNEDRPG	DK Random PIN Generate	Generates a PIN and a PIN reference value using the random process.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSNBDRP CSNEDRP	DK Regenerate PRW	Generates a new PIN reference value for a changed account number.
Chapter 14, "Key data set management," on page 1199		
CSNBKRC CSNEKRC	CKDS Key Record Create	Adds a key record containing a key token set to binary zeros to both the in-storage and DASD copies of the CKDS.
CSNBKRC2 CSNEKRC2	CKDS Key Record Create2	Adds a key record containing a key token to both the in-storage and DASD copies of the CKDS.
CSNBKRD CSNEKRD	CKDS Key Record Delete	Deletes a key record from both the in-storage and DASD copies of the CKDS.
CSNBKRR CSNEKRR	CKDS Key Record Read	Copies an internal key token from the in-storage copy of the CKDS to application storage.
CSNBKRR2 CSNEKRR2	CKDS Key Record Read2	Copies an internal key token from the in-storage copy of the CKDS to application storage.
CSNBKRW CSNEKRW	CKDS Key Record Write	Writes an internal key token to the CKDS record specified in the key label parameter. Updates both the in-storage and DASD copies of the CKDS currently in use.
CSNBKRW2 CSNEKRW2	CKDS Key Record Write2	Writes an internal key token to the CKDS record specified in the key label parameter. Updates both the in-storage and DASD copies of the CKDS currently in use.
CSFCRC CSFCRC6	Coordinated KDS Administration	Performs a CKDS refresh or CKDS reencipher and change master key operation while allowing applications to update the CKDS. In a sysplex environment, this callable service performs a coordinated sysplex-wide refresh or change master key operation from a single ICSF instance.
CSFMPS CSFMPS6	ICSF Multi-Purpose Service	Validates the keys in the active CKDS.
CSFKDSL CSFKDSL6	KDS List	Lists the labels matching specified metadata and other search criteria in the active CKDS.
CSFKDMR CSFKDMR6	KDS Metadata Read	Copies specified metadata from the active CKDS record to application storage.
CSFKDMW CSFKDMW6	KDS Metadata Write	Writes specified metadata to a list of CKDS records. Updates both the in-storage and DASD copies of the CKDS that is currently in use.

Table 13. Summary of ICSF callable services (continued)		
Service	**Service name**	**Function**
CSFRRT CSFRRT6	Key Data Set Record Retrieve	Retrieves a KDSR format record from a CKDS, PKDS, or TKDS.
CSFKDU CSFKDU6	Key Data Set Update	Updates a key dataset record.
Chapter 15, "Utilities," on page 1267		
CSNBXBC or CSNBXCB	Character/nibble Conversion	Converts a binary string to a character string or vice versa.
CSNBXEA or CSNBXAE	Code Conversion	Converts EBCDIC data to ASCII data or vice versa.
CSFIQA CSFIQA6	ICSF Query Algorithm	Use this utility to retrieve information about the cryptographic and hash algorithms available. You can control the amount of data that is returned by passing in different *rule_array* keywords.
CSFIQF CSFIQF6	ICSF Query Facility	Provides ICSF status, as well as coprocessor information.
CSFIQF2 CSFIQF26	ICSF Query Facility2	Provide information on the cryptographic environment as currently known by ICSF. This callable service is not SAF protected nor will it call any cryptographic coprocessors.
CSFSTAT CSFSTAT6	Cryptographic Usage Statistic	Tracks cryptographic usage external to the ICSF address space.
CSNB9ED	X9.9 Data Editing	Edits an ASCII text string according to the editing rules of ANSI X9.9–4.
Chapter 16, "Trusted interfaces," on page 1325		
CSFPCI	PCI interface	Query the list of access control points, send/receive requests to coprocessors, upload secure audit records, and obtain cryptographic coprocessor configurations.
CSFWRP CSFWRP6	Key Token Wrap	Unwraps a secure key token from the current master key and then rewraps the key token in the system protection key. This service is intended for operating system code.

Chapter 3. Introducing CCA PKA cryptography and using PKA callable services

The preceding topic focused on symmetric cryptography or secret-key cryptography. This is symmetric—senders and receivers use the same key (which must be exchanged securely in advance) to encipher and decipher data.

Public key cryptography does not require exchanging a secret key. It is asymmetric—the sender and receiver each have a pair of keys, a public key and a different but corresponding private key.

You can use PKA support to exchange symmetric secret keys securely and to compute digital signatures for authenticating messages to users. You can also use public key cryptography in support of secure electronic transactions over open networks, using SET protocols.

PKA key algorithms

Public key cryptography uses a key pair consisting of a public key and a private key. The PKA public key uses one of the following algorithms:

Rivest-Shamir-Adleman (RSA)

The RSA algorithm is the most widely used and accepted of the public key algorithms. It uses three quantities to encrypt and decrypt text: a public exponent (PU), a private exponent (PR), and a modulus (M). Given these three and some cleartext data, the algorithm generates ciphertext as follows:

$$ciphertext = cleartext^{PU} \pmod{M}$$

Similarly, this operation recovers cleartext from ciphertext:

$$cleartext = ciphertext^{PR} \pmod{M}$$

An RSA key consists of an exponent and a modulus. The private exponent must be secret, but the public exponent and modulus need not be secret.

Elliptic Curve Cryptography (ECC)

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. One of the main benefits in comparison with non-ECC cryptography is the same level of security provided by keys of smaller size. ICSF uses ECC for digital signatures and symmetric keys using the Diffie-Hellman key agreement scheme.

CRYSTALS-Dilithium Digital Signature Algorithm (CRDL-DSA)

CRYSTALS-Dilithium is a lattice-based digital signature scheme whose security is based on the hardness of finding short vectors in lattices. The CRYSTALS-Dilithium Digital Signature Algorithm (CRDL-DSA) is a quantum-safe algorithm (QSA) and is a member of the CRYSTALS (Cryptographic Suite for Algebraic Lattices) suite of algorithms. The strength of a CRYSTALS-Dilithium key is represented by the size of its matrix of polynomials. For example, CRYSTALS-Dilithium (6,5) has a matrix size of 6x5. The larger the matrix size, the stronger the key. CRYSTALS-Dilithium keys can only be used for Digital Signature Generation and Verification.

CRYSTALS-Kyber Algorithm

CRYSTALS-Kyber is an IND-CCA2-secure key encapsulation mechanism (KEM), whose security is based on the hardness of solving the learning-with-errors (LWE) problem over module lattices. The CRYSTALS-Kyber is a quantum-safe algorithm (QSA) and is a member of the CRYSTALS (Cryptographic Suite for Algebraic Lattices) suite of algorithms. ICSF currently supports Kyber-1024 Round 2. Kyber-1024 aims at security roughly equivalent to AES-256.

PKA keys

Master keys

PKA master keys protect private keys.

- RSA private key sections (X'02', X'06', and X'08') are protected by the RSA Master Key (RSA-MK). The RSA-MK is a triple-length DES key.
- ECC private key sections, RSA AOPK private key sections (X'30' and X'31'), and QSA private key sections are protected by the ECC Master Key (ECC-MK). The ECC master key is a 256-bit AES key.
- In order for PKA services to function the RSA and/or ECC master keys must be installed. The ICSF administrator installs the master keys on the CCA coprocessors by using either the pass phrase initialization routine, the Clear Master Key Entry panels, or the optional Trusted Key Entry (TKE) workstation.

  Prior to PKA services being enabled on the CCA coprocessor, these conditions must be met:

  – The RSA and/or ECC master keys on the CCA coprocessor must be installed.
  – The PKDS must be initialized with the RSA and/or ECC master keys installed on the CCA coprocessor.

Operational private keys

RSA operational private keys are protected under two layers of DES encryption. They are encrypted under an Object Protection Key (OPK) that in turn is encrypted under the RSA master key. ECC operational private keys are protected under two layers of AES encryption. They are encrypted under an AES OPK that in turn is encrypted under the ECC master key. The OPK is dynamically generated for each private key at import time or when the private key is generated on a CCA coprocessor. ICSF provides a public key data set (PKDS) for the storage of application PKA keys.

Key strength and wrapping of key

Key strength is measured as "bits of security" as described in the documentation of NIST and other organizations. Each individual key will have its "bits of security" computed, then the different key types (AES, DES, ECC, RSA, HMAC) can then have their relative strengths compared on a single scale. When the raw value of a particular key falls between discrete values of the NIST table, the lower value from the table will be used as the "bits of security".

The following tables show some examples of the restrictions due to key strength.

When wrapping an HMAC key with an AES key-encrypting key, the strength of the AES key-encrypting key depends on the attributes of the HMAC key.

Table 14. AES EXPORTER strength required for exporting an HMAC key under an AES EXPORTER

Key-usage field 2 in the HMAC key	**Minimum strength of AES EXPORTER to adequately protect the HMAC key**
SHA-256, SHA-384, SHA-512	256 bits
SHA-224	192 bits
SHA-1	128 bits

Table 15. Minimum RSA modulus length to adequately protect an AES key

Bit length of AES key to be exported	**Minimum strength of RSA wrapping key to adequately protect the AES key**
128	3072

Table 15. Minimum RSA modulus length to adequately protect an AES key (continued)	
Bit length of AES key to be exported	**Minimum strength of RSA wrapping key to adequately protect the AES key**
192	7860
256	15360

Key strength and key wrapping access control points

In order to comply with cryptographic standards, including ANSI X9.24 Part 1 and PCI-HSM, ICSF provides a way to ensure that a key is not wrapped with a key weaker than itself. ICSF provides a set of access control points in the domain role to control the wrapping of keys. ICSF administrators can use these access control points to meet an installation's individual requirements.

There are new and existing access control points that control the wrapping of keys by master and key-encrypting keys. These access control points will either prohibit the wrapping of a key by a key of weaker strength or will return a warning (return code 0, reason code non-zero) when a key is wrapped by a weaker key. All of these ACPs are disabled by default in the domain role.

The processing of callable services will be affected by these access control points. Here is a description of the access control points, the wrapping they control, and the effect on services. These access control points apply to symmetric and asymmetric keys.

When the **Prohibit weak wrapping - Transport keys** access control point is enabled, any service that attempts to wrap a key with a weaker transport key will fail.

When the **Prohibit weak wrapping - Master keys** access control point is enabled, any service that wraps a key under a master key will fail if the master key is weaker than the key being wrapped.

When the **Warn when weak wrap - Transport keys** access control point is enabled, any service that attempts to wrap a key with a weaker transport key will succeed with a warning reason code.

When the **Warn when weak wrap - Master keys** access control point is enabled, any service that attempts to wrap a key with a weaker master key will succeed with a warning reason code.

24-byte DATA keys with a zero control vector can be wrapped with a 16-byte key, the DES master key, or a key-encrypting key, which violates the wrapping requirements. The **Prohibit weak wrapping – Transport keys** and **Prohibit weak wrapping – Master keys** access control points do not cause services to fail for this case. The **Disallow 24-byte DATA wrapped with 16-byte Key** access control point does control this wrapping. When enabled, services will fail. The **Warn when weak wrap – Transport keys** and **Warn when weak wrap – Master keys** access control points will cause the warning to be returned when the access control points are enabled.

When the **TBC – Disallow triple-length MAC key** access control point is enabled, CSNDRKX will fail to import a triple-length MAC key under a double-length key-encrypting key. CSNDTBC will not wrap a triple-length MAC key under a double-length key-encrypting key. The **Prohibit weak wrapping – Transport keys** and **Prohibit weak wrapping – Master keys** access control points do not cause services to fail for this case. The **Warn when weak wrap – Transport keys** and **Warn when weak wrap – Master keys** access control points will cause the warning to be returned when the ACPs are enabled.

If the **Prohibit Weak Wrap** access control point is enabled, RSA private keys may not be wrapped using a weaker DES key-encrypting key. Enabling the **Allow weak DES wrap of RSA** access control points will override this restriction.

RSA private key tokens

The existing RSA private key tokens use a DES object protection key to wrap the private key parts of the key. This wrapping is not compliant for large modulus sizes. New private key sections have been introduced for RSA keys where the object protection key is an AES key. These private key sections are compliant.

PKA callable services

CCA coprocessors provide RSA digital signature functions, key management functions, and DES key distribution functions, PIN, MAC and data encryption functions, and application programming interfaces to these functions through callable services. You can also generate RSA keys on these coprocessors.

ECC support is provided by CCA Cryptographic coprocessors that are a CEX3C or later. Specifically, they provide ECDSA digital signature functions, ECC key management functions, and application programming interfaces to these functions through callable services.

Callable services supporting digital signatures

ICSF provides these services that support digital signatures.

Digital Signature Generate callable service (CSNDDSG and CSNFDSG)

This service generates a digital signature using an RSA, ECC, or CRYSTALS-Dilithium private key. It supports these methods of signature generation:

- ANSI X9.62 (ECC)
- Edwards-curve digital signature algorithm (ECC)
- Koblitz curve secp256k1 (ECC using ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS #1 v1.5 and v2.1 (RSA)
- Padding on the left with zeros (RSA)
- CRYSTALS-Dilithium Digital Signature Algorithm (QSA)

The input text can be hashed using the One-Way Hash Generate callable service, the MDC Generate callable service, or the Digital Signature Generate callable service.

Digital Signature Verify callable service (CSNDDSV and CSNFDSV)

This service verifies a digital signature using an RSA, ECC, or CRYSTALS-Dilithium public key. This service supports these methods of signature generation:

- ANSI X9.62 (ECC)
- Edwards-curve digital signature algorithm (ECC)
- Koblitz curve secp256k1 (ECC using ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS #1 v1.5 and v2.1 (RSA)
- Padding on the left with zeros (RSA)
- CRYSTALS-Dilithium Digital Signature Algorithm (QSA)

The input text can be hashed using the One-Way Hash Generate callable service, the MDC Generate callable service, or the Digital Signature Generate callable service.

Callable services for PKA key management

ICSF provides these services for PKA key management.

PKA Key Generate Callable Service (CSNDPKG and CSNFPKG)

This service generates an RSA or ECC internal or external private key tokens. The internal tokens can be used with services. You can extract the public key token with the PKA Public Key Extract callable service from the private key token.

Input to the PKA key generate callable service is either a skeleton key token created by the PKA key token build callable service or a valid key token.

PKA Key Import Callable Service (CSNDPKI and CSNFPKI)

This service imports a PKA private key, which may be RSA or ECC.

The key token to import can be in the clear or encrypted. The PKA key token build utility creates a clear PKA key token. The PKA key generate callable service generates either a clear or an encrypted PKA key token.

PKA Key Token Build Callable Service (CSNDPKB and CSNFPKB)

The PKA key token build callable service is a utility you can use to create an external PKA key token containing an unenciphered private RSA or ECC key. You can supply this token as input to the PKA key import callable service to obtain an operational internal token containing an enciphered private key. You can also use this service to input a clear unenciphered public ECC or RSA key and return the public key in a token format that other PKA services can use directly.

Use this service to build skeleton key tokens for input to the PKA key generate callable service for creation of RSA or ECC keys.

PKA Key Token Change Callable Service (CSNDKTC and CSNFKTC)

This service changes PKA key tokens (RSA and ECC) or trusted block key tokens, from encipherment under the cryptographic coprocessor's old RSA master key or ECC master key to encipherment under the current cryptographic coprocessor's RSA master key or ECC master key. This callable service only changes private internal tokens. An active CCA coprocessor is required.

PKA Key Translate (CSNDPKT and CSNFPKT)

This service translates a CCA RSA key token to an external smart card key token. An active CCA Crypto Express coprocessor is required.

PKA Public Key Extract Callable Service (CSNDPKX and CSNFPKX)

This service extracts a PKA public key token from a PKA internal (operational) or external (importable) private key token. It performs no cryptographic verification of the PKA private key token.

Callable services to manage the Public Key Data Set (PKDS)

The Public Key Data Set (PKDS) is a repository for ECC and RSA public and private keys and trusted blocks. An application can store keys in the PKDS and refer to them by label when using any of the callable services which accept public key tokens as input. The PKDS update callable services provide support for creating and writing records to the PKDS and reading and deleting records from the PKDS.

The syntax of the PKDS Key Record Create, PKDS Key Record Delete, PKDS Key Record Read, and PKDS Key Record Write services is identical with the same services provided by the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessor programming interface. Key management applications that use these common interface verbs can run on both systems without change.

Coordinated KDS Administration callable service (CSFCRC and CSFCRC6)

This service is used to perform the following functions: coordinated CKDS change master key, coordinated CKDS refresh, coordinated PKDS change master key, coordinated PKDS refresh, and coordinated TKDS change master key.

While this service is performing a coordinated change master key function, dynamic KDS update services may continue to run in parallel. During a coordinated refresh function, dynamic KDS update services may continue to be enabled; however, they will be temporarily suspended internally until the coordinated refresh completes. If this cannot be tolerated, it is recommended to disable dynamic KDS update services when using this service.

In a sysplex environment, this callable service is executed from a single ICSF instance, and the function is coordinated across all sysplex members sharing the same active KDS. This removes the need for KDS refresh or KDS change master key functions to be performed locally on every ICSF instance sharing the same active KDS in a sysplex environment.

ICSF Multi-Purpose Service callable service (CSFMPS and CSFMPS6)

This service is used to validate the keys in the active CKDS or PKDS. Use the ICSF multi-purpose callable service prior to a change master key operation as a way to detect keys that may cause a change master key operation to fail.

Key Data Set List callable service (CSFKDSL and CSFKDL6)

This service is used to list the labels of records in the active CKDS and PKDS that match selected metadata and other search criteria. This service is used to list the handles of records in the active TKDS that match selected metadata and other search criteria.

Key Data Set Metadata Read callable service (CSFKDMR and CSFKDMR6)

This service is used to read the metadata of a single record in the active CKDS, PKDS, or TKDS. Multiple metadata fields may be read in one call.

Key Data Set Metadata Write callable service (CSFKDMW and CSFKDMW6)

This service is used to add, delete, or change the metadata of a list of records in the active CKDS, PKDS, or TKDS. Multiple metadata fields may be changed in one call.

PKDS Key Record Create Callable Service (CSNDKRC and CSNFKRC)

This service accepts an RSA or ECC private key token in either external or internal format, or an RSA or ECC public key token or trusted blocks and writes a new record to the PKDS. An application can create a null token in the PKDS by specifying a token length of zero. The key label must be unique.

PKDS Key Record Delete Callable Service (CSNDKRD and CSNFKRD)

This service deletes a record from the PKDS. An application can specify that the entire record be deleted, or that only the contents of the record be deleted. If only the contents of the record are deleted, the record will still exist in the PKDS but will contain only binary zeros. The key label must be unique.

Note: Retained keys cannot be deleted from the PKDS with this service. See “Retained Key Delete (CSNDRKD and CSNFRKD)” on page 1191 for information on deleting retained keys.

PKDS Key Record Read Callable Service (CSNDKRR and CSNFKRR)

This service reads a record from the PKDS and returns the contents of that record to the caller. The key label must be unique.

PKDS Key Record Write Callable Service (CSNDKRW and CSNFKRW)

This service accepts an RSA or ECC private key token in either external or internal format, or an RSA or ECC public key token or trusted blocks and writes over an existing record in the PKDS. An application can check the PKDS for a null record with the label provided and overwrite this record if it does exist. Alternatively, an application can specify to overwrite a record regardless of the contents of the record.

Note: Retained keys cannot be written to the PKDS with the PKDS Key Record Write service, nor can a retained key record in the PKDS be overwritten with this service.

Callable services for working with retained private keys

Private keys can be generated, retained, and used within the secure boundary of a CCA coprocessor. Retained keys are generated by the PKA Key Generate (CSNDPKG) callable service. The private key values of retained keys never appear in any form outside the secure boundary. All retained keys have an entry in the PKDS that identifies the CCA coprocessor where the retained private key is stored. ICSF provides these callable services to list and delete retained private keys.

Retained Key Delete Callable Service (CSNDRKD and CSNFRKD)

The retained key delete callable service deletes a key that has been retained within a CCA Crypto Express and also deletes the record containing the key token from the PKDS.

Retained Key List Callable Service (CSNDRKL and CSNFRKL)

The retained key list callable service lists the key labels of private keys that are retained within the boundaries of a CCA coprocessor installed on your server.

Clearing the retained keys on a coprocessor

The retained keys on a CCA coprocessor may be cleared. These are the conditions under which the retained key will be lost:

- If the CCA coprocessor detects tampering (the intrusion latch is tripped), ALL installation data is cleared: master keys, retained keys for all domains, as well as roles and profiles.
- If the CCA coprocessor detects tampering (the secure boundary of the card is compromised), it self-destructs and can no longer be used.
- If you issue a command from the TKE workstation to zeroize a domain

  This command zeroizes the data specific to a domain: master keys and retained keys.
- If you issue a command from the Support Element panels to zeroize all domains.

  This command zeroizes ALL installation data: master keys, retained keys and access control roles and profiles.

Callable services for the TR-34

The ANSI TR-34 protocol provides a method to securely distribute symmetric keys using asymmetric techniques.

TR-34 Bind-Begin (CSNDT34B and CSNFT34B)

The TR-34 Bind-Begin service performs the key distribution host side of the bind operations to bind a host to a key receiving device.

TR-34 Bind-Complete (CSNDT34C and CSNFT34C)

The TR-34 Bind-Complete service performs the key receiving device side of the bind operations to complete the bind of the key receiving device to the key distribution host.

TR-34 Key Distribution (CSNDT34D and CSNFT34D)

The TR-34 Key Distribution service performs the key distribution host side of the key transport operations to prepare and send the distributed operational key.

TR-34 Key Receive (CSNDT34R and CSNFT34R)

The TR-34 Key Receive service performs the key receiving device side of the key transport operations to receive the distributed operational key.

Callable services for Secure Electronic Transaction (SET)

SET is an industry-wide open standard for securing bankcard transactions over open networks. The SET protocol addresses the payment phase of a transaction from the individual, to the merchant, to the acquirer (the merchant's current bankcard processor). It can be used to help ensure the privacy and integrity of real time bankcard payments over the Internet. In addition, with SET in place, everyone in the payment process knows who everyone else is. The card holder, the merchant, and the acquirer can be fully authenticated because the core protocol of SET is based on digital certificates. Each participant in the payment transaction holds a certificate that validates his or her identity. The public key infrastructure allows these digital certificates to be exchanged, checked, and validated for every transaction made over the Internet. The mechanics of this operation are transparent to the application.

Under the SET protocol, every online purchase must be accompanied by a digital certificate which identifies the card-holder to the merchant. The buyer's digital certificate serves as an electronic representation of the buyer's credit card but does not actually show the credit card number to the merchant. Once the merchant's SET application authenticates the buyer's identity, it then decrypts the order information, processes the order, and forwards the still-encrypted payment information to the acquirer for processing. The acquirer's SET application authenticates the buyer's credit card information, identifies the merchant, and arranges settlement. With SET, the Internet becomes a safer, more secure environment for the use of payment cards.

ICSF provides these callable services that can be used in developing SET applications that make use of the IBM eServer zSeries cryptographic hardware at the merchant and acquirer payment gateway.

SET Block Compose Callable Service (CSNDSBC and CSNFSBC)

The SET Block Compose callable service performs DES encryption of data, OAEP-formatting through a series of SHA-1 hashing operations, and the RSA-encryption of the Optimal Asymmetric Encryption Padding (OAEP) block.

SET Block Decompose Callable Service (CSNDSBD and CSNFSBD)

The SET Block Decompose callable service decrypts both the RSA-encrypted and the DES-encrypted data.

PKA key tokens

PKA key tokens contain RSA, ECC, or QSA private or public keys. PKA tokens are variable length because they contain either RSA, ECC, or QSA key values, which are variable in length. Consequently, length parameters precede all PKA token parameters. The maximum allowed size is 8000 bytes. PKA key tokens consist of a token header, any required sections, and any optional sections. Optional sections depend on the token type. PKA key tokens can be public or private, and private key tokens can be internal or external. Therefore, there are three basic types of tokens, each of which can contain either RSA, ECC, or QSA information:

- A public key token.
- A private external key token.
- A private internal key token.

Public key tokens contain only the public key. Private key tokens contain the public and private key pair. Table 16 on page 101 summarizes the sections in each type of token.

Table 16. Summary of PKA key token sections

Section	Public External Key Token	Private External Key Token	Private Internal Key Token
Header	X	X	X
RSA, ECC, or QSA private key information		X	X
RSA, ECC, or QSA public key information	X	X	X
Key name (optional, RSA or QSA only)		X	X
Internal information			X

As with DES and AES key tokens, the first byte of a PKA key token contains the token identifier which indicates the type of token.

A first byte of X'1E' indicates an external token with a cleartext public key and optionally a private key that is either in cleartext or enciphered by a transport key-encrypting key. An external key token is in importable key form. It can be sent on the link.

A first byte of X'1F' indicates an internal token with a cleartext public key and a private key that is enciphered by the PKA master key and ready for internal use. An internal key token is in operational key form. A PKA private key token must be in operational form for ICSF to use it. (PKA public key tokens are used directly in the external form.)

Formats for public and private external and internal RSA, ECC, or QSA key tokens begin in "PKA key tokens" on page 1546.

X.509 certificates

An X.509 certificate couples an identity to an RSA or ECC public key. An X.509 certificate can be used in PKA services in lieu of a PKA public key token. An X.509 certificate may only be used in services consistent with its usage attributes.

Note: Only the 31-bit services are listed, but this also applies to the 64-bit equivalent services.

- digitalSignature -- Services: CSNDDSV.
- nonRepudiation -- Services: CSNDDSV.
- keyEncipherment -- Services: CSNDSYG, CSNDSYX, CSNDPKE.
- dataEncipherment -- Services: CSNDPKE.
- keyAgreement -- Services: None.
- keyCertSign -- Services: CSNDDSV.
- cRLSign -- Services: CSNDDSV.
- encipherOnly -- Services: N/A.
- decipherOnly -- Services: N/A.

Certificates may also be used in the ANSI TR-34 services (CSNDT34B, CSNDT34C, CSNDT34D, CSNDT34R). See the service descriptions for information on what usage is required, if any, for each input certificate. If no key usage attribute is present in the certificate, the certificate may be used in any service.

One way to create a certificate with specific key usage attributes is to use the Public Infrastructure Create (CSNDPIC/CSNFPIC) callable service to create a certificate signing request (CSR) with the desired key usage attributes. This CSR can then be signed by a Certificate Authority (CA) to create a certificate with

the desired key usage attributes. The RACDCERT GENCERT command can be used to create a certificate from a CSR.

When using operational certificates that are to be validated against an issuer CA root certificate, the root certificate of the issuer CA must be loaded on all CEX6C and higher coprocessors using the Trusted Key Entry (TKE) workstation before the operational certificate can be used successfully.

PKA key management

You can also generate PKA keys in several ways.

- Using the ICSF PKA key generate callable service.
- Using the 4765 PCIe Cryptographic Coprocessor PKA key generate verb, or a comparable product from another vendor.

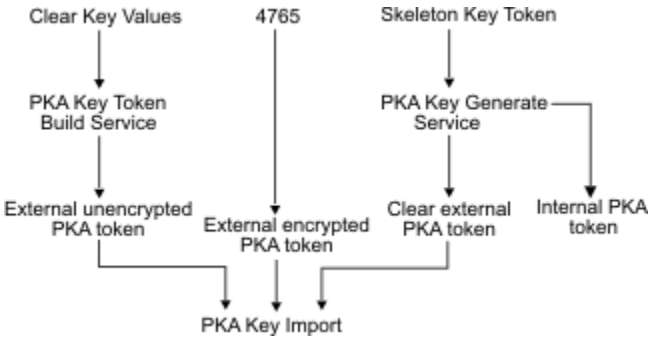


Figure 7. PKA Key Management

With a CCA coprocessor, you can use the ICSF PKA key generate callable service to generate internal and external PKA tokens. You can also generate RSA keys on another system. To input a clear RSA key to ICSF, create the token with the PKA key token build callable service and import it using the PKA key import callable service. To input an encrypted RSA key, generate the key on the Transaction Security System and import it using the PKA key import callable service.

In either case, use the PKA key token build callable service to create a skeleton key token as input (see "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 1147).

The PKA key import callable service uses the clear token from the PKA key token build service or a clear or encrypted token from the Transaction Security System to securely import the key token into operational form for ICSF to use. ICSF does not permit the export of the imported PKA key.

The PKA public key extract callable service builds a public key token from a private key token.

The Public Infrastructure Certificate (CSNDPIC and CSNFPIC) callable service can be used to create a Certificate Signing Request (CSR). This CSR must then be processed by a Certificate Authority (CA) to create an operational certificate for use by the PKA services.

Application RSA or ECC public and private keys tokens can be stored in the public key data set (PKDS), a VSAM data set. X.509 certificates cannot be stored in the PKDS.

Security and integrity of the token

PKA private key tokens may optionally have a 64-byte *private-key name* field. If the *private-key name* exists, ICSF uses RACROUTE REQUEST=AUTH to verify access to the CSFKEYS profile of name *private-key name* prior to using the token in a callable service. For additional security, the processor also validates the entire private key token.

For RSA and QSA keys, the *private-key name* is stored in the PKA private-key name section of the token.

For ECC keys, the *private-key name* is stored in the associated data section of the token.

Key identifier for PKA key token

A *key identifier* for a PKA key token is a variable length (maximum allowed size is 8000 bytes) area that contains one of these:

Key label
Identifies keys that are in the PKDS. Ask your ICSF administrator for the key labels that you can use.

Key token
can be either an internal key token, an external key token, or a null key token. Key tokens are generated by an application (for example, using the PKA key generate callable service), or received from another system that can produce external key tokens.

An **internal key token** can be used only on ICSF, because a PKA master key encrypts the key value. Internal key tokens contain keys in operational form only.

An **external key token** can be exchanged with other systems because a transport key that is shared with the other system encrypts the key value. External key tokens contain keys in either exportable or importable form.

A **null key token** consists of 8 bytes of binary zeros. The PKDS Key Record Create service can be used to write a null token to the PKDS. This PKDS record can subsequently be identified as the target token for the PKA key import or PKA key generate service.

The term *key identifier* is used when a parameter could be one of the previously discussed items and to indicate that different inputs are possible. For example, you may want to specify a specific parameter as either an internal key token or a key label. The key label is, in effect, an indirect reference to a stored internal key token.

Key label

If the first byte of the key identifier is greater than X'40', the field is considered to be holding a **key label**. The contents of a key label are interpreted as a pointer to a public key data set (PKDS) key entry. The key label is an indirect reference to an internal key token.

A key label is specified on callable services with the *key_identifier* parameter as a 64-byte character string, left-justified, and padded on the right with blanks. In most cases, the callable service does not check the syntax of the key label beyond the first byte. One exception is the CKDS key record create callable service which enforces the KGUP rules for key labels unless syntax checking is bypassed by a preprocessing exit.

A key label has this form:

Table 17. Key label format

Offset	Length	Data
00-63	64	Key label name

Key token

A key token is a variable length (maximum allowed size is 8000 bytes) field composed of key value and control information. PKA keys can be either public or private RSA, ECC, CRYSTALS-Dilithium, or CRYSTALS-Kyber keys. Each key token can be either an internal key token (the first byte of the key identifier is X'1F'), an external key token (the first byte of the key identifier is X'1E'), or a null private key token (the first byte of the key identifier is X'00'). For the format of each token type, refer to Appendix B, “Key token formats,” on page 1501.

An internal key token is a token that can be used only on the ICSF system that created it (or another ICSF system with the same PKA master key). It contains a key that is encrypted under the PKA master key.

An application obtains an internal key token by using one of the callable services such as:

- PKA Key Generate.

- PKA Key Import.

The PKA Key Token Change callable service can reencipher private internal tokens from encryption under the old master key (either RSA or ECC) to encryption under the current master key. For debugging information, see Appendix B, “Key token formats,” on page 1501 for the format of an internal key token.

If the first byte of the key identifier is X'1E', the key identifier is interpreted as an external key token. An external PKA key token contains key (possibly encrypted) and control information. By using the external key token, you can exchange keys between systems.

An application obtains the external key token by using one of the callable services such as:

- PKA Public Key Extract.
- PKA Key Token Build.
- PKA Key Generate.

For debugging information, see Appendix B, “Key token formats,” on page 1501 for the format of an external key token.

If the first byte of the key identifier is X'00', the key identifier is interpreted as a null key token. For debugging information, see Appendix B, “Key token formats,” on page 1501 for the format of a null key token.

Compliant-tagged key tokens

A compliant-tagged key token must adhere to the requirements of a compliance mode. A coprocessor in compliance mode must be available to use compliant-tagged key tokens. For more detail, see Appendix H, “Impact of compliance mode on callable services,” on page 1695.

RSA

The compliant-tag is indicated by a flag in the private key section of the key token. For more information about the compliant-tag, see Appendix B, “Key token formats,” on page 1501.

To generate a compliant-tagged key token, the PKA Key Token Build (CSNDPKB/CSNFPKB) service must first be used to build a skeleton token with the compliant-tag flag on. This skeleton token can then be passed to any callable service that generates RSA key tokens and supports compliant-tagged key tokens, for example PKA Key Generate (CSNDPKG/CSNFPKG). A list of services that support compliant-tagged key tokens can be found in Appendix H, “Impact of compliance mode on callable services,” on page 1695.

The PKA Key Import (CSNDPKI/CSNFPKI) service is the exception because it does not support skeleton tokens as input. When importing a PKA key, if the key encrypting key is compliant-tagged, the resulting key is also compliant-tagged. For more information, see the PKA Key Import (CSNDPKI/CSNFPKI) service description.

Summary of the PKA callable services

Table 18 on page 104 lists the PKA callable services and their corresponding verbs. It also references the topic that describes the callable service. Note: PKA services start with CSND*xxx* and have corresponding CSF*xxx* names.

Table 18. Summary of PKA callable services

Service	Service Name	Function
Chapter 8, “Financial services,” on page 601		
CSNDSBC CSNFSBC	SET block compose	Composes the RSA-OAEP block and the DES-encrypted block in support of the SET protocol.

Table 18. Summary of PKA callable services (continued)		
Service	**Service Name**	**Function**
CSNDSBD	SET block decompose	Decomposes the RSA-OAEP block and the DES-encrypted block to provide unencrypted data back to the caller.
Chapter 11, "TR-34 symmetric key management," on page 1059		
CSNDT34B CSNFT34B	TR-34 Bind-Begin	Performs KDH operations to bind host to KRD.
CSNDT34C CSNFT34C	TR-34 Bind-Complete	Performs KRD operations to bind to KDH.
CSNDT34D CSNFT34D	TR-34 Key Distribution	Performs KDH operations to distribute operational symmetric keys.
CSNDT34R CSNFT34R	TR-34 Key Receive	Performs KRD operations to receive operational symmetric keys.
Chapter 12, "Using digital signatures," on page 1109		
CSNDDSG CSNFDSG	Digital signature generate	Generates a digital signature using a PKA private key supporting RSA, ECC, and CRYSTALS-Dilithium algorithms.
CSNDDSV CSNFDSV	Digital signature verify	Verifies a digital signature using a PKA public key supporting RSA, ECC, and CRYSTALS-Dilithium algorithms.
Chapter 13, "Managing PKA cryptographic keys," on page 1131		
CSNDPKG CSNFPKG	PKA key generate	Generate RSA, ECC, CRYSTALS-Dilithium, and CRYSTALS-Kyber private keys.
CSNDPKI CSNFPKI	PKA key import	Imports a PKA key token containing either a clear PKA key or a PKA key enciphered under a limited authority IMP-PKA KEK or an AES transport key.
CSNDPKB CSNFPKB	PKA key token build	Creates an external PKA key token containing a clear private RSA, ECC, CRYSTALS-Dilithium, or CRYSTALS-Kyber key. Using this token as input to the PKA key import callable service returns an operational internal token containing an enciphered private key. Using CSNDPKB on a clear public RSA, ECC, CRYSTALS-Dilithium, or CRYSTALS-Kyber key, returns the public key in a token format that other PKA services can directly use. CSNDPKB can also be used to create a skeleton token for input to the PKA Key Generate service for the generation of an internal RSA, ECC, CRYSTALS-Dilithium, or CRYSTALS-Kyber key token.

Table 18. Summary of PKA callable services (continued)		
Service	**Service Name**	**Function**
CSNDKTC CSNFKTC	PKA key token change	Changes PKA key tokens (RSA, ECC, CRYSTALS-Dilithium, and CRYSTALS-Kyber) or trusted block key tokens, from encipherment under the cryptographic coprocessor's old RSA master key or ECC master key to encipherment under the current cryptographic coprocessor's RSA master key or ECC master key. This callable service only changes private internal tokens.
CSNDPKT CSNFPKT	PKA key translate	Translates a CCA RSA key token to a smart card format or a PKCS #11 object. Converts a CCA RSA key token with a DES OPK to a token with an AES OPK. Converts a CCA PKA key token to a compliant-tagged token. Exports a CCA ECC, CRYSTALS-Dilithium, or CRYSTALS-Kyber token to AESKW external format.
CSNDPKX	PKA public key extract	Extracts a PKA public key token from a supplied PKA internal or external private key token. Performs no cryptographic verification of the PKA private token.
CSNDRKD CSNFRKD	Retained key delete	Deletes a key that has been retained within a CCA Crypto Express coprocessor.
CSNDRKL CSNFRKL	Retained key list	Lists key labels of keys that have been retained within all currently active CCA coprocessors.
Chapter 14, “Key data set management,” on page 1199		
CSFCRC CSFCRC6	Coordinated KDS Administration	Performs a PKDS refresh or PKDS reencipher and change master key operation while allowing applications to update the PKDS. In a sysplex environment, this callable service performs a coordinated sysplex-wide refresh or change master key operation from a single ICSF instance.
CSFMPS CSFMPS6	ICSF Multi-Purpose Service	Validates the keys in the active PKDS.
CSFKDSL CSFKDSL6	KDS List	Lists the labels matching specified metadata and other search criteria in the active PKDS.
CSFKDMR CSFKDMR6	KDS Metadata Read	Copies specified metadata from the active PKDS record to application storage.
CSFKDMW CSFKDMW6	KDS Metadata Write	Writes specified metadata to a list of PKDS records. Updates both the in-storage and DASD copies of the PKDS that is currently in use.
CSNDKRC CSNFKRC	PKDS Key Record Create	Writes a new record to both the in-storage and DASD copies of the PKDS.

Table 18. Summary of PKA callable services (continued)		
Service	**Service Name**	**Function**
CSNDKRD CSNFKRD	PKDS Key Record Delete	Deletes a record from both the in-storage and DASD copies of the PKDS.
CSNDKRR CSNFKRR	PKDS Key Record Read	Copies the contents of a record from the in-storage copy of the PKDS to application storage.
CSNDKRW CSNFKRW	PKDS key record write	Writes over an existing record in both the in-storage and DASD copies of the PKDS.

Chapter 4. Introducing PKCS #11 and using PKCS #11 callable services

The Integrated Cryptographic Service Facility has implemented callable service in support of PKCS #11. A callable service is a routine that receives control using a CALL statement in an application language. Each callable service performs one or more functions, including:

- Initializing and deleting PKCS #11 tokens.
- Creating, reading, updating and deleting PKCS #11 objects.
- Performing cryptographic operations.

Most services do not have hardware requirements. In general, ICSF will perform the operation in hardware or software depending on the environment. See each service for details. All new callable services will be invocable in AMODE(24), AMODE(31), or AMODE(64).

For more information about PKCS #11 and specific hardware requirements for each service or mechanism, see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

PKCS #11 services

PKCS #11 tokens and objects

ICSF provides callable services that support PKCS #11 token and object creation and use. The following table summarizes these callable services. For complete syntax and reference information, refer to Chapter 17, “Using PKCS #11 tokens and objects,” on page 1339.

Table 19. Summary of PKCS #11 callable services

Verb	Service Name	Function
CSFCRC	Coordinated KDS Administration	Performs a TKDS reencipher and change master key function while allowing applications to update the TKDS. In a sysplex environment, this service performs a coordinated sysplex-wide change master key function from a single ICSF instance.
CSFKDSL	KDS List	Lists the object handles matching specified metadata and other search criteria in the active TKDS.
CSFKDMR	KDS Metadata Read	Copies specified metadata from the active TKDS record to application storage.
CSFKDMW	KDS Metadata Write	Writes specified metadata to a list of TKDS records. Updates both the in-storage and DASD copies of the TKDS currently in use.
CSFPDVK	PKCS #11 Derive key	Generate a new secret key object from an existing key object.
CSFPDMK	PKCS #11 Derive Multiple Keys	Generate multiple secret key objects and protocol dependent keying material from an existing secret key object.
CSFPHMG	PKCS #11 Generate MAC	Generate a hashed message authentication code (MAC).

Table 19. Summary of PKCS #11 callable services (continued)		
Verb	**Service Name**	**Function**
CSFPGKP	PKCS #11 Generate Key Pair	Generate an RSA, DSA, Elliptic Curve, or Diffie-Hellman key pair.
CSFPGSK	PKCS #11 Generate Secret Key	Generate a secret key or set of domain parameters.
CSFPGAV	PKCS #11 Get Attribute Value	List the attributes of a PKCS #11 object.
CSFPOWH	PKCS #11 One-Way Hash, Sign, or Verify	Generate a one-way hash on specified text, sign specified text, or verify a signature on specified text.
CSFPPKS	PKCS #11 Private Key Sign	• Decrypt or sign data using an RSA private key using zero-pad or PKCS #1 v1.5 formatting. • Sign data using a DSA private key. • Sign data using an Elliptic Curve private key in combination with DSA.
CSFPPKV	PKCS #11 Public Key Verify	• Encrypt or verify data using an RSA public key using zero-pad or PKCS #1 v1.5 formatting. For encryption, the encrypted data is returned. • Verify a signature using a DSA public key. No data is returned. • Verify a signature using an Elliptic Curve public key in combination with DSA. No data is returned.
CSFPPRF	PKCS #11 Pseudo-Random Function	Generate pseudo-random output of arbitrary length.
CSFPSKD	PKCS #11 Secret Key Decrypt	Decipher data using a clear symmetric key.
CSFPSKE	PKCS #11 Secret Key Encrypt	Encipher data using a clear symmetric key.
CSFPSKR	PKCS #11 Secret Key Reencrypt	Decrypts and re-encrypts data using secure secret keys.
CSFPSAV	PKCS #11 Set Attribute Value	Update the attributes of a PKCS #11 object.
CSFPTRC	PKCS #11 Token Record Create	Initialize or re-initialize a z/OS PKCS #11 token, creates or copies a token object in the token data set and creates or copies a session object for the current PKCS #11 session.
CSFPTRD	PKCS #11 Token Record Delete	Delete a z/OS PKCS #11 token, token object, or session object.
CSFPTRL	PKCS #11 Token Record List	Obtain a list of z/OS PKCS #11 tokens. The caller must have SAF authority to the token. Also obtains a list of token and session objects for a token. Use a search template to restrict the search for specific attributes.
CSFPUWK	PKCS #11 Unwrap Key	Unwrap and create a key object using another key.
CSFPHMV	PKCS #11 Verify MAC	Verify a hash message authentication code (MAC).
CSFPWPK	PKCS #11 Wrap Key	Wrap a key with another key.

PKCS #11 key structure callable services

ICSF provides callable services that offer a fast-path alternative to some of the traditional PKCS #11 callable services. The following table summarizes these callable services. For complete syntax and reference information, refer to Chapter 18, "Using the PKCS #11 key structure callable services," on page 1429.

Table 20. Summary of PKCS #11 callable services that offer a fast-path alternative

Verb	Service Name	Function
CSFPPD2	PKCS #11 Private Key Structure Decrypt	Decrypt data using an RSA private key structure.
CSFPPE2	PKCS #11 Public Key Structure Encrypt	Encrypt data using an RSA public key structure.
CSFPPS2	PKCS #11 Private Key Structure Sign	Sign data using an RSA private key structure.
CSFPPV2	PKCS #11 Public Key Structure Verify	Verify data using an RSA public key structure.

Attribute list

The attributes of an object can be the input and the output of a service. The format of the attributes is shown here and applies to all PKCS #11 callable services. For the token record list service, the search_template has the same format as an attribute list. The lengths in the attribute list and attribute structures are unsigned integers.

An *attribute_list* is a structure in this format:

Number of attributes	Attribute	Attribute	...
2 bytes	4 + 2 + length of *value* bytes	4 + 2 + length of *value* bytes	...

Each attribute is a structure in this format:

Attribute name	Length of value (*n*)	Value
4 bytes	2 bytes	*n* bytes

Handles

A handle is a 44-byte identifier for a token or an object. The format of the handle is as follows:

Name of token or object	Sequence number	ID
32 bytes	8 bytes	4 bytes

The token name in the first 32 bytes of the handle is provided by the PKCS #11 application when the token or object is created. The first character of the name must be alphabetic or a national character (#, $, or @). Each of the remaining characters can be alphanumeric, a national character (#, $, or @), or a period(.)

The sequence number is a hexadecimal number stored as the EBCDIC representation of 8 hexadecimal numbers. The sequence number field in a token is EBCDIC blanks. The token record contains a last-used sequence number field, which is incremented each time an object associated with the token is created. This sequence number value is placed in the handle of the newly-created object.

The ID field is 4 characters. The first character (EBCDIC) identifies the object's category:

S
The handle belongs to a clear session object.

T
The handle belongs to a clear token object.

U
The handle belongs to a clear state object.

X
The handle belongs to an Enterprise PKCS #11 secure session object.

Y
The handle belongs to an Enterprise PKCS #11 secure token object.

Z
The handle belongs to an Enterprise PKCS #11 secure state object.

If the first character is blank, the handle belongs to a token.

The last three characters must be EBCDIC blanks.

Part 2. CCA callable services

This topic introduces DES, AES and PKA callable services.

Chapter 5. Managing symmetric cryptographic keys

This topic describes the callable services that generate and maintain cryptographic keys.

Using ICSF, you can generate keys using either the key generator utility program or the key generate callable service. ICSF provides a number of callable services to assist you in managing and distributing keys.

This topic describes these callable services:

- "Clear Key Import (CSNBCKI and CSNECKI)" on page 116
- "Control Vector Generate (CSNBCVG and CSNECVG)" on page 118
- "Control Vector Translate (CSNBCVT and CSNECVT)" on page 124
- "Cryptographic Variable Encipher (CSNBCVE and CSNECVE)" on page 128
- "Data Key Export (CSNBDKX and CSNEDKX)" on page 130
- "Data Key Import (CSNBDKM and CSNEDKM)" on page 133
- "Derive ICC MK (CSNBDCM and CSNEDCM)" on page 136
- "Derive Session Key (CSNBDSK and CSNEDSK)" on page 143
- "Diversified Key Generate (CSNBDKG and CSNEDKG)" on page 151
- "Diversified Key Generate2 (CSNBDKG2 and CSNEDKG2)" on page 162
- "Diversify Directed Key (CSNBDDK and CSNEDDK)" on page 170
- "ECC Diffie-Hellman (CSNDEDH and CSNFEDH)" on page 182
- "Generate Issuer MK (CSNBGIM and CSNEGIM)" on page 197
- "Key Encryption Translate (CSNBKET and CSNEKET)" on page 204
- "Key Export (CSNBKEX and CSNEKEX)" on page 207
- "Key Generate (CSNBKGN and CSNEKGN)" on page 211
- "Key Generate2 (CSNBKGN2 and CSNEKGN2)" on page 224
- "Key Import (CSNBKIM and CSNEKIM)" on page 237
- "Key Part Import (CSNBKPI and CSNEKPI)" on page 242
- "Key Part Import2 (CSNBKPI2 and CSNEKPI2)" on page 247
- "Key Test (CSNBKYT and CSNEKYT)" on page 252
- "Key Test2 (CSNBKYT2 and CSNEKYT2)" on page 256
- "Key Test Extended (CSNBKYTX and CSNEKYTX)" on page 267
- "Key Token Build (CSNBKTB and CSNEKTB)" on page 271
- "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297
- "Key Translate (CSNBKTR and CSNEKTR)" on page 343
- "Key Translate2 (CSNBKTR2 and CSNEKTR2)" on page 346
- "Multiple Clear Key Import (CSNBCKM and CSNECKM)" on page 355
- "Multiple Secure Key Import (CSNBSKM and CSNESKM)" on page 358
- "PKA Decrypt (CSNDPKD and CSNFPKD)" on page 365
- "PKA Encrypt (CSNDPKE and CSNFPKE)" on page 374
- "Prohibit Export (CSNBPEX and CSNEPEX)" on page 384
- "Prohibit Export Extended (CSNBPEXX and CSNEPEXX)" on page 386
- "Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)" on page 388
- "Remote Key Export (CSNDRKX and CSNFRKX)" on page 393

- "Restrict Key Attribute (CSNBRKA and CSNERKA)" on page 402
- "Secure Key Import (CSNBSKI and CSNESKI)" on page 408
- "Secure Key Import2 (CSNBSKI2 and CSNESKI2)" on page 412
- "Symmetric Key Export (CSNDSYX and CSNFSYX)" on page 417
- "Symmetric Key Export with Data (CSNDSXD and CSNFSXD)" on page 426
- "Symmetric Key Generate (CSNDSYG and CSNFSYG)" on page 430
- "Symmetric Key Import (CSNDSYI and CSNFSYI)" on page 439
- "Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)" on page 444
- "Trusted Block Create (CSNDTBC and CSNFTBC)" on page 450
- "Unique Key Derive (CSNBUKD and CSNEUKD)" on page 454

Clear Key Import (CSNBCKI and CSNECKI)

Use the Clear Key Import callable service to import a clear single-length DES DATA key that is to be used to encipher or decipher data. This callable service can import only DATA keys. Clear Key Import accepts an 8-byte clear DATA key, enciphers it under the master key, and returns the encrypted single-length DES DATA key in operational form in an internal key token.

If the clear key value does not have odd parity in the low-order bit of each byte, the service returns a warning value in the *reason_code* parameter. The callable service does not adjust the parity of the key.

Note: This service has been deprecated. New applications should use the Multiple Clear Key Import service that is described in "Multiple Clear Key Import (CSNBCKM and CSNECKM)" on page 355 instead of this service. The Multiple Clear Key Import service supports 8-byte, 16-byte, and 24-byte DES DATA keys and AES DATA keys.

The callable service name for AMODE(64) invocation is CSNECKI.

Format

```
CALL CSNBCKI(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              clear_key,
              key_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

clear_key

Direction	Type
Input	String

The *clear_key* specifies the 8-byte clear key value to import.

key_identifier

Direction	Type
Output	String

A 64-byte string that is to receive the internal key token. “Key identifier for key token” on page 7 describes the internal key token.

Usage notes

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The **Clear Key Import/Multiple Clear Key Import - DES** access control point controls the function of this service.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 21. Clear Key Import required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Control Vector Generate (CSNBCVG and CSNECVG)

The Control Vector Generate callable service builds a control vector from keywords specified by the *key_type* and *rule_array* parameters.

The callable service name for AMODE(64) is CSNECVG.

Format

```
CALL CSNBCVG(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_type,
             rule_array_count,
             rule_array,
             reserved,
             control_vector )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

key_type

Direction	Type
Input	String

A string variable containing a keyword for the key type. The keyword is 8 bytes in length, left justified, and padded on the right with space characters. It is taken from this list:

- CIPHER
- CIPHERXI
- CIPHERXL
- CIPHERXO
- CVARDEC
- CVARENC
- CVARPINE
- CVARXCVL
- CVARXCVR
- DATA
- DATA-CV (The default DATA control vector will be used.)
- DATAM
- DATAMV
- DECIPHER
- DKYGENKY
- ENCIPHER
- EXPORTER
- IKEYXLAT
- IMPORTER
- IPINENC

- KEYGENKY
- MAC
- MACVER
- OKEYXLAT
- OPINENC
- PINGEN
- PINVER
- SECMSG

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter.

rule_array

Direction	Type
Input	Character String

Keywords that provide control information to the callable service. Each keyword is left justified in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage. “Key Token Build (CSNBKTB and CSNEKTB)” on page 271 illustrates the key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector. The rule array keywords are:

Table 22. Rule array keywords for Control Vector Generate

Keyword	Meaning
Key usage keywords: These keywords allow the key to be use with a callable service, restrict the key to a single algorithm, or allow the key to be used for a specific function.	
Key-encrypting keys	
OPIM	IMPORTER keys that have this attribute can be used in the CSNBKGN service when the key form is OPIM.
IMEX	IMPORTER and EXPORTER keys that have this attribute can be used in the CSNBKGN service when the key form is IMEX.
IMIM	IMPORTER keys that have this attribute can be used in the CSNBKGN service when the key form is IMIM.
IMPORT	IMPORTER keys that have this attribute can be used to import a key in the CSNBKIM service.
OPEX	EXPORTER keys that have this attribute can be used in the CSNBKGN service when the key form is OPEX.
EXEX	EXPORTER keys that have this attribute can be used in the CSNBKGN service when the key form is EXEX.
EXPORT	EXPORTER keys that have this attribute can be used to export a key in the CSNBKEX service.
XLATE	IMPORTER and EXPORTER keys that have this attribute can be used in the CSNBKTR and CSNBKTR2 services.

Table 22. Rule array keywords for Control Vector Generate (continued)	
Keyword	**Meaning**
ANY	This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.
NOT-KEK	This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.
DATA	This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.
LMTD-KEK	This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.
MAC keys	
ANY-MAC	A key with this attribute can be used with any service that uses MAC keys.
CVVKEY-A	Restricts the usage of the key to single-length key-A key or double-length key-A and key-B keys for the CSNBCSG and CSNBCSV services.
CVVKEY-B	Restricts the usage of the key to single-length key-B key for the CSNBCSG and CSNBCSV services.
AMEX-CSC	This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.
ANSIX9.9	This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.
Data operation keys. Note: When the key type is SECMSG, either SMKEY or SMPIN must be specified in the *rule_array*.	
SMKEY	A key with this attribute can be used to encrypt keys in an EMV secure message.
SMPIN	A key with this attribute can be used to encrypt PINs in an EMV secure message.
PIN keys	
NO-SPEC	The key is not restricted to a specific PIN-calculation method.
IBM-PIN	The key can be used with the IBM 3624 PIN-calculation method.
IBM-PINO	The key can be used with the IBM 3624 PIN-calculation method with offset processing.
GBP-PIN	The key can be used with the IBM German Bank Pool PIN-calculation method.
GBP-PINO	The key can be used with the IBM German Bank Pool PIN-calculation method with institution-PIN input or output.
VISA-PVV	The key can be used with the Visa PVV PIN-calculation method.
INBK-PIN	The key can be used with the InterBank PIN-calculation method.
NOOFFSET	Indicates that a PINGEN or PINVER key cannot be used to generate or verify of a PIN when an offset process is requested.
CPINGEN	The key can be used with the CSNBPGN service.
CPINGENA	The key can be used with the CSNBCPA service.
EPINGEN	The key can be used with the CSNBEPG service.
EPINVER	The key can be used with the CSNBPVR service.

Table 22. Rule array keywords for Control Vector Generate (continued)	
Keyword	**Meaning**
CPINENC	The key can be used with the CSNBCPE service.
REFORMAT	The key can be used with the CSNBPTR service in the REFORMAT mode.
TRANSLAT	The key can be used with the CSNBPTR service in the TRANSLATE mode.
EPINGENA	This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.
Key-generating keys. Note: When the key type is KEYGENKY, either CLR8-ENC or UKPT must be specified in the rule array.	
CLR8-ENC	The key can be used to multiply-encrypt 8 bytes of clear data with a generating key.
DALL	The key can be used to generate keys with the following key types: DATA, DATAC, DATAM, DATAMV, DMKEY, DMPIN, EXPORTER, IKEYXLAT, IMPORTER, MAC, MACVER, OKEYXLAT, and PINVER.
DDATA	The key can be used to generate a single-length or double-length DATA or DATAC key.
DEXP	The key can be used to generate an EXPORTER or an OKEYXLAT key.
DIMP	The key can be used to generate an IMPORTER or an IKEYXLAT key.
DMAC	The key can be used to generate a MAC or DATAM key.
DMKEY	The key can be used to generate a SECMSG with a SMKEY secure messaging key for encrypting keys.
DMPIN	The key can be used to generate a SECMSG with a SMPIN secure messaging key for encrypting PINs.
DMV	The key can be used to generate a MACVER or DATAMV key.
DPVR	The key can be used to generate a PINVER key.
DKYL0	A DKYGENKY key with this subtype can be used to generate a key based on the key-usage bits.
DKYL1	A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL0.
DKYL2	A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL1.
DKYL3	A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL2.
DKYL4	A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL3.
DKYL5	A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL4.
DKYL6	A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL5.
DKYL7	A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL6.
UKPT	The key can be used to derive operational keys.
Key management keywords: These keywords are valid with all key types. The keywords are used to allow or disallow key management functions.	

Table 22. Rule array keywords for Control Vector Generate (continued)	
Keyword	**Meaning**
Key lengths (not all lengths are valid for all key types, see Table 116 on page 280 for allowable lengths by key type.)	
MIXED	Indicates that the key can be either a replicated single-length key or a double-length key with two different, random 8-byte values.
SINGLE, KEYLN8	Specifies the key as a single-length key.
DOUBLE, KEYLN16	Specifies the key as a double-length key.
DOUBLE-O	Specifies the key as a double-length key with guaranteed different key values.
TRIPLE	Specifies the key as a triple-length key. For the DATA key type, the control vector is the default DATA control vector.
TRIPLE-O	Specifies the key as a triple-length key with guaranteed different key values. For the DATA key type, the control vector is the default DATA control vector.
Miscellaneous attributes	
COMP-TAG	The key can be used with PCI-HSM compliant applications.
ENH-ONLY	Prohibits the key from being wrapped with the legacy method after it has been wrapped with the enhanced method.
KEY-PART	Specifies the control vector is for a key part.
NO-XPORT	Prohibits the key from being exported by Key_Export or Data_Key_Export.
NOEXCPAC	Prohibits export to CPACF protected key format.
NOCMPTAG	The key cannot be used with PCI-HSM compliant applications.
NOT31XPT	Prohibits the key from being exported by the Key_Export_to_TR31 verb.
T31XPTOK	Permits the key to be exported by the Key_Export_to_TR31 verb.
XPORT-OK	Permits the key to be exported by Key_Export or Data_Key_Export. Also permits the key to be exported by the Key_Export_to_TR31 verb, unless NOT31XPT is enabled.
XPRTCPAC	Allows export to CPACF protected key format.

reserved

Direction	**Type**
Input	String

The *reserved* parameter must be a variable of 8 bytes of X'00'.

control_vector

Direction	**Type**
Output	String

A 16-byte string variable in application storage where the service returns the generated control vector.

Usage notes

See Table 116 on page 280 for an illustration of key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector.

Required hardware

No cryptographic hardware is required by this callable service.

Control Vector Translate (CSNBCVT and CSNECVT)

The Control Vector Translate callable service changes the control vector used to encipher an external key.

See "Changing control vectors with the Control Vector Translate callable service" on page 1613 for additional information about this service.

The callable service name for AMODE(64) invocation is CSNECVT.

Format

```
CALL CSNBCVT(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             KEK_key_identifier,
             source_key_token,
             array_key_left,
             mask_array_left,
             array_key_right,
             mask_array_right,
             rule_array_count,
             rule_array,
             target_key_token )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is defined in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

KEK_key_identifier

Direction	Type
Input/Output	String

The 64-byte string variable containing an internal key token or the key label of an internal key token record containing the key-encrypting key. The control vector in the internal key token must specify the key type of IMPORTER, EXPORTER, IKEYXLAT, or OKEYXLAT.

source_key_token

Direction	Type
Input	String

A 64-byte string variable containing the external key token with the key and control vector to be processed. Triple-length keys are not supported. Tokens wrapped with the WRAPENH3 method are not supported. See Table 24 on page 127 for restrictions on the key length and wrapping method.

array_key_left

Direction	Type
Input/Output	String

A 64-byte string variable containing an internal key token or a key label of an internal key token record that deciphers the left mask array. The internal key token must contain a control vector specifying a CVARXCVL key type.

mask_array_left

Direction	Type
Input	String

A string of seven 8-byte elements containing the mask array enciphered under the left array key.

array_key_right

Direction	Type
Input/Output	String

A 64-byte string variable containing an internal key token or a key label of an internal key token record that deciphers the right mask array. The internal key token must contain a control vector specifying a CVARXCVR key type.

mask_array_right

Direction	Type
Input	String

A string of seven 8-byte elements containing the mask array enciphered under the right array key.

rule_array_count

Direction	Type
Input	Integer

An integer containing the number of elements in the rule array. The value of the *rule_array_count* must be 0, 1, or 2 for this service. If the *rule_array_count* is 0, the default keywords are used.

rule_array

Direction	Type
Input	Character String

The *rule_array* parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The rule_array keywords are:

Table 23. Keywords for Control Vector Translate

Keyword	Meaning
Parity Adjustment Rule (optional)	
ADJUST	Ensures that all target key bytes have odd parity. This is the default.
NOADJUST	Prevents the parity of the target being altered.
Key-portion Rule (optional)	
BOTH	Causes both halves of a 16-byte source key to be processed with the result placed into corresponding halves of the target key. When you use the BOTH keyword, the mask array must be able to validate the translation of both halves.
LEFT	Causes an 8-byte source key, or the left half of a 16-byte source key, to be processed with the result placed into both halves of the target key. This is the default.
RIGHT	Causes the right half of a 16-byte source key to be processed with the result placed into the right half of the target key. The left half is copied unchanged (still enciphered) from the source key.
SINGLE	Causes the left half of the source key to be processed with the result placed into the left half of the target key token. The right half of the target key is unchanged.

target_key_token

Direction	Type
Input/Output	String

A 64-byte string variable containing an external key token with the new control vector. This key token contains the key halves with the new control vector.

The output *target_key_token* will be wrapped in the same manner as the input *source_key_token*.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If *KEK_key_identifier* is a label of an IMPORTER or EXPORTER key, the label must be unique in the CKDS.

Access control point

The **Control Vector Translate** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 24. Control Vector Translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Support for enhanced wrapped DES keys requires the July 2019 or later licensed internal code (LIC).
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Support for enhanced wrapped DES keys requires the July 2019 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Support for enhanced wrapped DES keys requires the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	

Table 24. Control Vector Translate required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Cryptographic Variable Encipher (CSNBCVE and CSNECVE)

The Cryptographic Variable Encipher callable service uses a DES CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. You can use this service to prepare a mask array for the Control Vector Translate service. The plaintext must be a multiple of eight bytes in length.

The callable service name for AMODE(64) invocation is CSNECVE.

Format

```
CALL CSNBCVE(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             c_variable_encrypting_key_identifier,
             text_length,
             plaintext,
             initialization_vector,
             ciphertext )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is defined in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

c_variable_encrypting_key_identifier

Direction	Type
Input/Output	String

The 64-byte string variable containing an internal key or a key label of an internal key token record in the CKDS. The internal key must contain a control vector that specifies a CVARENC key type.

text_length

Direction	Type
Input	Integer

An integer variable containing the length of the plaintext and the returned ciphertext.

plaintext

Direction	Type
Input	String

A string of length 8 to 256 bytes which contains the plaintext. The data must be a multiple of 8 bytes.

initialization_vector

Direction	Type
Input	String

A string variable containing the 8-byte initialization vector that the service uses in encrypting the plaintext.

ciphertext

Direction	Type
Output	String

The field which receives the ciphertext. The length of this field is the same as the length of the plaintext.

Restrictions

- The text length must be a multiple of 8 bytes.
- The maximum length of text that the security server can process is 256 bytes.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control point

The **Cryptographic Variable Encipher** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 25. Cryptographic Variable Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Data Key Export (CSNBDKX and CSNEDKX)

Use the Data Key Export callable service to reencipher a data-encrypting key (key type of DATA only) from encryption under the master key to encryption under an exporter key-encrypting key. The reenciphered key is in a form suitable for export to another system.

The Data Key Export service generates a key token with the same key length as the input token's key.

The callable service name for AMODE(64) invocation is CSNEDKX.

Format

```
CALL CSNBDKX(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
```

```
                source_key_identifier,
                exporter_key_identifier,
                target_key_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

source_key_identifier

Direction	Type
Input/Output	String

A 64-byte string for an internal key token or label that contains a data-encrypting key to be reenciphered. The data-encrypting key is encrypted under the master key.

exporter_key_identifier

Direction	Type
Input/Output	String

The identifier of the *key_encrypting* key to wrap the source key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte DES key token of key type EXPORTER.

For X9.143 keys, the identifier is a variable-length key block of a TDES exporter: key usage K0, algorithm T, and mode of use E.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

target_key_identifier

Direction	Type
Input/Output	String

A 64-byte field that is to receive the external key token, which contains the reenciphered key that has been exported. The reenciphered key can now be exchanged with another cryptographic system.

The output *target_key_identifier* will be wrapped in the same manner as the *source_key_identifier*.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

Restrictions

Current applications will fail if they use an equal key halves exporter to export a key with unequal key halves. You must have access control point 'Data Key Export - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified] and a token is passed as input to be exported, an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the exported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 26. Required access control points for Data Key Export

Access Control Point	Restrictions
Data Key Export	None
Data Key Export - Unrestricted	Key-encrypting key may have equal key halves.

To use a NOCV key-encrypting key with the Data Key Export service, the **NOCV KEK usage for export-related functions** access control point must be enabled in addition to one or both of the access control points listed.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the key-encrypting key is a double-length key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 27. Data Key Export required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Data Key Import (CSNBDKM and CSNEDKM)

Use the Data Key Import callable service to import an encrypted source DES single-length, double-length or triple-length DATA key and create or update a target internal key token with the master key enciphered source key.

The callable service name for AMODE(64) invocation is CSNEDKM.

Format

```
CALL CSNBDKM(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             source_key_token,
             importer_key_identifier,
             target_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

source_key_token

Direction	Type
Input/Output	String

64-byte string variable containing the source key to be imported. The source key must be an external token or null token. The external key token must indicate that a control vector is present; however, the control vector is usually valued at zero. A double-length key that should result in a default DATA control vector must be specified in a version X'01' external key token. Otherwise, both single and double-length keys are presented in a version X'00' key token. For the null token, the service will process this token format as a DATA key encrypted by the importer key and a null (all zero) control vector.

importer_key_identifier

Direction	Type
Input/Output	String

The identifier of the *key_encrypting* key to unwrap the source key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte DES key token of key type IMPORTER.

For X9.143 keys, the identifier is a variable-length key block of a TDES importer: key usage K0, algorithm T, and mode of use D.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

target_key_identifier

Direction	Type
Input/Output	String

A 64-byte string variable containing a null key token or an internal key token. The key token receives the imported key.

If a skeleton key token is provided as input to this parameter, the wrapping method in the skeleton token will be used. Otherwise, the system default key wrapping method will be used to wrap the token.

When the WRAPENH3 method is required, a WRAPENH3 skeleton key token or a WRAPENH3 internal key token must be specified.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

Restrictions

Current applications will fail if they use an equal key halves importer to import a key with unequal key halves. You must have access control point 'Data Key Import - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This service does not adjust the key parity of the source key.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 28. Required access control points for Data Key Import

Access Control Point	Restrictions
Data Key Import	None.
Data Key Import - Unrestricted	Key-encrypting key may have equal key halves.

To use a NOCV key-encrypting key with the Data Key Import service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 29. Data Key Import required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Derive ICC MK (CSNBDCM and CSNEDCM)

The Derive ICC MK callable service generates ICC master keys from issuer master keys. ICC master keys are needed for ICC personalization, EMV transaction processing, and EMV scripting. Optionally, this service returns the ICC master key as an external token wrapped under a key-encrypting key (KEK). Use the TKE workstation to establish the KEK that is optionally used by this service.

The following ICC master keys can be generated:

ICC Master Application Cryptogram Key (AC)
This key is used to generate and verify the ARQC and ARPC.

ICC Master Secure Messaging Authentication Key (MAC)
This key is used to provide integrity for EMV scripting.

ICC Master Secure Messaging Confidentiality Key (ENC)
This key is used to provide confidentiality for EMV scripting.

ICC Master Data Key (DATA)
This key is used for functions that require encryption and decryption of EMV fields.

The callable service name for AMODE(64) invocation is CSNEDCM.

Format

```
CALL CSNBDCM(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             issuer_master_key_identifier_length,
             issuer_master_key_identifier,
             icc_master_key_identifier_length,
             icc_master_key_identifier,
             transport_key_identifier_length,
             transport_key_identifier,
             pan_length,
             pan,
             pan_seq_number,
             reserved1_length,
             reserved1,
             reserved2_length,
             reserved2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 3 and the maximum value is 7.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 30. Rule array keywords for Derive ICC MK

Keyword	Meaning
Algorithm (Required)	
TDES	Specifies the use of Triple-DES.
Key mode (One required). Defines the key derivation mechanism.	
VISA	Use this key mode for Visa Cryptogram Version 10 or Cryptogram Version 18 key derivation.
MC	Use this key mode for MasterCard M/CHIP 2.1 key derivation.
Output key type (One required). See the *issuer_master_key_identifier* and *icc_master_key_identifier* parameters for more information.	
AC	Derives the ICC Master Application Cryptogram Key. This key is used to generate and verify the ARQC and ARPC. When the VISA key mode is specified and the CVN10 cryptogram version is specified or defaulted, the issuer master key must be of a DKYL0 DKYGENKY key and the derived ICC master key will be of type MAC. When the VISA key mode is specified and the CVN18 cryptogram version is specified or the MC key mode is specified, the issuer master key must be a DKYL1 DKYGENKY key and the derived ICC master key will be a DKYL0 DKYGENKY key.
MAC	Derives the ICC Master Secure Messaging Authentication Key. This key is used to provide integrity for EMV scripting. When the VISA key mode is specified, the issuer master key must be of a DKYL0 DKYGENKY key and the derived ICC master key will be of type MAC. Not valid with cryptogram version CVN18. When the MC key mode is specified, the issuer master key must be a DKYL1 DKYGENKY key and the derived ICC master key will be a DKYL0 DKYGENKY key.

Table 30. Rule array keywords for Derive ICC MK (continued)	
Keyword	**Meaning**
ENC	Derives the ICC Master Secure Messaging Confidentiality Key. This key is used to provide confidentiality for EMV scripting. When the MC key mode is specified, the issuer master key must be a DKYL1 DKYGENKY key and the derived ICC master key will be a DKYL0 DKYGENKY key. Not valid with key mode VISA.
DATA	Derives the ICC Master DATA Key. This key is used for functions that require encryption and decryption of EMV fields. When the MC key mode is specified, the issuer master key must be a DKYL1 DKYGENKY key and the derived ICC master key will be a DKYL0 DKYGENKY key. Not valid with key mode VISA.
Key encryption (Optional)	
MASTER	Specifies to return the ICC master key as an internal token encrypted under the master key. This is the default.
XPORT	Specifies to return the ICC master key as external token encrypted under the *transport_key_identifier*.
Control flag (Optional)	
APPANSEQ	Specifies to append the PAN sequence number when the card specific master key is derived. See the descriptions of *pan* and *pan_seq_number*. The default is not to append the PAN sequence number. If PAN length rule PAN-19 is specified, APPANSEQ must also be specified.
Cryptogram version number (Optional. Only valid with key mode VISA.)	
CVN10	Specifies to use the Visa Cryptogram Version 10 processing method. This is the default for key mode VISA.
CVN18	Specifies to use the Visa Cryptogram Version 18 processing method. This is only allowed for output key type AC.
PAN length (Optional. Only valid with Cryptogram version number CVN18.)	
PAN-16	Specifies that only the rightmost 16 digits of the PAN are used to derive the Unique DEA Keys using the method described in Visa ICC Card Specification V1.6, Appendix D.7. This is the default.
PAN-19	Specifies that the rightmost 19 digits of the PAN are used to derive the Unique DEA Keys using the method described in EMV Book 2, Section A1.4.1, Option B.

issuer_master_key_identifier_length

Direction	**Type**
Input	Integer

Specifies the length of the *issuer_master_key_identifier* parameter in bytes. The value must be 64.

issuer_master_key_identifier

Direction	Type
Input/Output	String

A 64-byte DES key identifier (either an internal token or key label) for the issuer master key. The issuer master key is the key from which the ICC master key is derived.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

The key algorithm must be DES and the key type must be DKYGENKY. The value subtype and key usage attributes required are listed in Table 31 on page 140.

Table 31. Derive ICC MK: Key requirements

Master key	VISA CVN10	VISA CVN18	MC
Application Cryptogram Key (AC)	DMAC, DKYL0	DMAC, DKYL1	DMAC, DKYL1
Secure Messaging Authentication Key (MAC)	DMAC, DKYL0	N/A	DMAC, DKYL1
Secure Messaging Confidentiality Key (ENC)	N/A	N/A	DMPIN, DKYL1
Data Key (DATA)	N/A	N/A	DDATA, DKYL1

icc_master_key_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *icc_master_key_identifier* parameter in bytes. The value must be 64.

icc_master_key_identifier

Direction	Type
Output	String

A 64-byte CCA DES key identifier for the ICC master key. The ICC master key is the DES key from which session keys are derived.

On output, this is the derived key token containing the ICC master key. If the XPORT rule is specified, the key token is returned in external format wrapped by the *transport_key_identifier*. Otherwise, it is returned in internal format.

If the *issuer_master_key_identifier* is compliant-tagged, a compliant-tagged token is generated.

The attributes of the generated key (See the output key type rules for a description of key types derived by this service based on the selected key mode):

Table 32. Derive ICC MK: Key type and key usage attributes of the generated keys

Master key	VISA CVN10	VISA CVN18	MC
Application Cryptogram Key (AC)	MAC	DKYGENKY, DMAC, DKYL0	DKYGENKY, DMAC, DKYL0

Table 32. Derive ICC MK: Key type and key usage attributes of the generated keys (continued)			
Master key	**VISA CVN10**	**VISA CVN18**	**MC**
Secure Messaging Authentication Key (MAC)	MAC	N/A	DKYGENKY, DMAC, DKYL0
Secure Messaging Confidentiality Key (ENC)	N/A	N/A	DKYGENKY, DMPIN, DKYL0
Data Key (DATA)	N/A	N/A	DKYGENKY, DDATA, DKYL0

transport_key_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *transport_key_identifier* parameter in bytes. When the XPORT keyword is specified, the value must be 64. Otherwise, the value must be 0.

transport_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the generated keys. This key must be an EXPORTER key type specified as an operational key token or as a key label of an EXPORTER key in key storage. When the *transport_key_identifier_length* is zero, this parameter is ignored.

If the NOCV bit is on in the internal key token containing the transport key, the transport key (not the transport key variant) is used to encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter; or you are passing the transport key in the internal key token with the NOCV bit on and your program is running in supervisor state or key 0-7.

The NOCV bit is shown in Table 608 on page 1504.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

pan_length

Direction	Type
Input	Integer

Length in bytes of the *pan* parameter. The value must be 10.

pan

Direction	Type
Input	String

The 10-byte EMV card's Primary Account Number. The data must be in compressed numeric format and right justified in a 10-byte field, padded to the left with zeroes. For example, PAN 1234567890 must be provided as x'00000000001234567890'.

This data is used in combination with the PAN sequence number to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

pan_seq_number

Direction	Type
Input	String

The 1-byte sequence number of the EMV card's Primary account Number. If the APPANSEQ control flag rule array keyword was specified, this PAN sequence number is used in combination with the PAN to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Cryptographic services used by Derive ICC MK

The following CCA cryptographic services are used by Derive ICC MK:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate
- CSNBKEX - Key Export

The caller does not require authorization to each of these services, only to Derive ICC MK. Additionally, the caller must have the required access control points enabled.

Access control points

The following access control points must be enabled to use Derive ICC MK:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - TDES-XOR
- Diversified Key Generate - TDESEMV2/TDESEMV4

To use a NOCV key-encrypting key, the **NOCV KEK usage for export-related functions** access control must be enabled in addition to the other access control points listed.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 33. Derive ICC MK required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Derive Session Key (CSNBDSK and CSNEDSK)

The Derive Session Key callable service derives a session key from either an issuer master key or an ICC master key. The session key can be used for EMV transaction processing or EMV scripting.

The following session keys can be derived for Visa Cryptogram Version 10 processing method (VISA CVN10):

- Application Cryptogram Session Key (AC) for ARQC and ARPC processing.
- Secure Messaging Authentication Session Key (MAC) for scripting.
- Secure Messaging Confidentiality Session Key (ENC) for scripting.

The following session key can be derived for Visa Cryptogram Version 18 processing method (VISA CVN18):

- Application Cryptogram Session Key (AC) for ARQC and ARPC processing.

The following session keys can be derived for MasterCard M/CHIP 2.1 processing method (MC):

- Application Cryptogram Session Key (AC) from the issuer master key ARQC and ARPC processing.
- ARPC key (AC) from the issuer ARPC master key for ARPC processing.
- Secure Messaging Authentication Session Key (MAC) from either the issuer or ICC master key for scripting.
- Secure Messaging Confidentiality Session Key (ENC) from either the issuer or ICC master key for scripting.
- DATA Session Key (DATA) from either the issuer or ICC master key for encryption and decryption of EMV fields.

The following session keys can be derived for EMV Book 2, Annex A1.3, Visa Cryptogram Version 14, and MasterCard M/CHIP 4.MasterCard M/CHIP 2.1 processing method (EMV):

- Application Cryptogram Session Key (AC) for ARQC and ARPC processing.
- Secure Messaging Authentication Session Key (MAC) for scripting.
- Secure Messaging Confidentiality Session Key (ENC) for scripting.
- DATA Session Key (DATA) for encryption and decryption of EMV fields.

The callable service name for AMODE(64) invocation is CSNEDSK.

Format

```
CALL CSNBDSK(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             master_key_identifier_length,
             master_key_identifier,
             session_key_identifier_length,
             session_key_identifier,
             pan_length,
             pan,
             pan_seq_number,
             atc,
             unpredictable_number_length,
             unpredictable_number,
             reserved1_length,
             reserved1,
             reserved2_length,
             reserved2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 3 and the maximum value is 6.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 34. Rule array keywords for Derive Session Key

Keyword	Meaning
Algorithm (Required)	
TDES	Specifies the use of Triple-DES.
Key mode (One required). Defines the key derivation mechanism. See the output key type for more information.	

Table 34. Rule array keywords for Derive Session Key (continued)	
Keyword	**Meaning**
VISA	Specifies to use either the Visa Cryptogram Version 10 (CVN10) or Cryptogram Version 18 (CVN18) key derivation. • For CVN10, the card's master key is used as the session key (the keys are the same for each session). • For CVN18, the card's master key is used to derive the session key. See Visa specification, Appendix D2. Padding is determined by the Cryptogram version used.
MC	Specifies to use the MasterCard M/CHIP 2.1 key derivation. The ATC and an unpredictable number are encrypted with the card's master key. The card's master key is used when generating the ARPC. Padding is according to EMV.
EMV	Specifies to use the session key derivation as described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.3. Use this key mode for Visa Cryptogram Version 14 and MasterCard M/CHIP 4. EMV padding rules apply.
Output key type (One required). See the *master_key_identifier* description for the requirements for the key-generating key.	
AC	Specifies to derive the Application Cryptogram Session Key (AC). The derived session key will be of the type MAC.
MAC	Specifies to derive the Secure Messaging Authentication Session Key (MAC). The derived session key will be of the type MAC. Not valid with cryptogram version CVN18.
ENC	Specifies to derive the Secure Messaging Confidentiality Session Key (ENC). The derived session key will be of the type SECMSG. Not valid with cryptogram version CVN18.
DATA	Specifies to derive the DATA Session Key (DATA). The derived session key will be of the type DATAC. Not valid with key mode VISA.
Control flag (Optional)	
APPANSEQ	Specifies to append the PAN sequence number when the card specific master key is derived. See the descriptions of *pan* and *pan_seq_number*. The default is not to append the PAN sequence number. If PAN length rule PAN-19 is specified, APPANSEQ must also be specified.
Branch Factor (One optional, valid only with key mode EMV). The branching factor is to be used in EMV session key derivation.	
TDESEMV2	Specifies a branch factor of 2 for a height of 16. This is the default.
TDESEMV4	Specifies a branch factor of 4 for a height of 8.
Cryptogram version number (Optional. Only valid with key mode VISA.)	
CVN10	Specifies to use the Visa Cryptogram Version 10 processing method. This is the default for key mode VISA.

Table 34. Rule array keywords for Derive Session Key (continued)	
Keyword	**Meaning**
CVN18	Specifies to use the Visa Cryptogram Version 18 processing method. This is only allowed for output key type AC.
PAN length (Optional. Only valid with Cryptogram version number CVN18.)	
PAN-16	Specifies that only the rightmost 16 digits of the PAN are used to derive the Unique DEA Keys using the method described in Visa ICC Card Specification V1.6, Appendix D.7. This is the default.
PAN-19	Specifies that the rightmost 19 digits of the PAN are used to derive the Unique DEA Keys using the method described in EMV Book 2, Section A1.4.1, Option B.

master_key_identifier_length

Direction	Type
Input	Integer

Specifies the length of the *master_key_identifier* parameter in bytes. The value must be 64.

master_key_identifier

Direction	Type
Input/Output	String

A 64-byte DES key identifier (either an internal token or key label) for the issuer master key or the ICC master key. This key is the DES key from which card specific keys and session keys are derived.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

The key algorithm must be DES and the key type must be DKYGENKY. The required subtype and key usage attributes for the master key type and key mode combinations are listed in Table 35 on page 147.

Table 35. Derive Session Key: Key requirements				
Master key	**VISA CVN10**	**VISA CVN18**	**MC**	**EMV**
Issuer AC	DMAC, DKYL0	DMAC, DKYL1	DMAC, DKYL1	DMAC, DKYL0
Issuer MAC	DMAC, DKYL0	N/A	DMAC, DKYL1	DMAC, DKYL0
Issuer ENC	DMPIN, DKYL0	N/A	DMPIN, DKYL1	DMPIN, DKYL0
Issuer DATA	N/A	N/A	DDATA, DKYL1	DDATA, DKYL0
Issuer APRC	N/A	N/A	DMAC, DKYL0	N/A
ICC AC	N/A	DMAC, DKYL0	DMAC, DKYL0	N/A
ICC MAC	N/A	N/A	DMAC, DKYL0	N/A
ICC ENC	N/A	N/A	DMPIN, DKYL0	N/A
ICC DATA	N/A	N/A	DDATA, DKYL0	N/A

session_key_identifier_length

Direction	Type
Input/Output	Integer

This parameter specifies the length of the *session_key_identifier* parameter in bytes. The value must be 64.

session_key_identifier

Direction	Type
Output	String

The 64-byte parameter for the derived session key token. The session key can be used for EMV transaction processing or EMV scripting.

If the *master_key_identifier* is compliant-tagged, a compliant-tagged token is generated.

The session key type generated based on the master key and the key mode specified are shown in Table 36 on page 148.

Table 36. Derive Session Key: Attributes of the key generated

Session Key	VISA CVN10	VISA CVN18	MC	EMV
AC	MAC	MAC	MAC	MAC
MAC	MAC	N/A	MAC	MAC
ENC	SECMSG, SMPIN	N/A	SECMSG, SMPIN	SECMSG, SMPIN
DATA	N/A	N/A	DATAC	DATAC
ARPC (AC)	N/A	N/A	MAC	N/A

pan_length

Direction	Type
Input	Integer

Length in bytes of the *pan* parameter. The value must be 10.

pan

Direction	Type
Input	String

The 10-byte EMV card's Primary Account Number. The data must be in compressed numeric format and right justified in a 10-byte field, padded to the left with zeroes. For example, PAN 1234567890 must be provided as x'0000000001234567890'.

This data is used in combination with the PAN sequence number to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

pan_seq_number

Direction	Type
Input	String

The 1-byte sequence number of the EMV card's Primary account Number. If the APPANSEQ control flag rule array keyword was specified, this PAN sequence number is used in combination with the PAN to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

atc

Direction	Type
Input	String

The 2-byte application transaction counter that is used for session key derivation. See the key mode rules for more information on session key derivation.

Note: The first byte is the high-order byte and the second byte is the low order byte.

unpredictable_number_length

Direction	Type
Input	Integer

Specifies the length of the *unpredictable_number* in bytes. The value must be 4 or 8.

unpredictable_number

Direction	Type
Input	String

The 4-byte or 8-byte unpredictable number used in the MasterCard M/Chip 2.1 session key derivation scheme.

For EMV scripting with the MC key mode, this value is expected to be 8 bytes. For EMV transaction processing and verification functions using the MC key mode, this value is expected to be 4 bytes.

The data in this field will not be reformatted by the API before use.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Cryptographic services used by Derive Session Key

The following CCA cryptographic services are used by Derive Session Key:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate

The caller does not require authorization to each of these services, only to Derive Session Key. Additionally, the caller must have the required access control points enabled.

Access control points

The following access control points must be enabled to use Derive Session Key:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - TDES-XOR
- Diversified Key Generate - TDESEMV2/TDESEMV4

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 37. Derive Session Key required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	

Table 37. Derive Session Key required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Diversified Key Generate (CSNBDKG and CSNEDKG)

Use the Diversified Key Generate service to generate a key based on the key-generating key, the processing method, and the parameter supplied. The control vector of the key-generating key also determines the type of target key that can be generated.

To use this service, specify:

- The rule array keyword to select the diversification process.
- The operational key-generating key from which the diversified keys are generated. The control vector associated with this key restricts the use of this key to the key generation process. This control vector also restricts the type of key that can be generated.
- The data and length of data used in the diversification process.
- The generated-key may be an internal token or a skeleton token containing the desired CV of the generated-key. The generated key CV must be one that is permitted by the processing method and the key-generating key. The generated-key will be returned in this parameter.
- A key generation method keyword.

This service generates diversified keys as follows:

- Determines if it can support the process specified in rule array.
- Recovers the key-generating key and checks the key-generating key class and the specified usage of the key-generating key.
- Determines that the control vector in the generated-key token is permissible for the specified processing method.
- Determines that the control vector in the generated-key token is permissible by the control vector of the key-generating key.
- Determines the required data length from the processing method and the generated-key CV. Validates the *data_length*.
- Generates the key appropriate to the specific processing method. Adjusts parity of the key to odd. Creates the internal token and returns the generated diversified key.

The callable service name for AMODE(64) invocation is CSNEDKG.

Format

```
CALL CSNBDKG(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             generating_key_identifier,
             data_length,
             data,
```

```
                data_decrypting_key_identifier,
                generated_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The only valid value is 1, 2, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The processing method is the algorithm used to create the generated key. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 38. Rule Array Keywords for Diversified Key Generate	
Keyword	**Meaning**
Processing Method for generating or updating diversified keys (one, required)	
A28OWFCL	Specifies that 16 bytes of clear data will be processed as described in AS2805.5.4 to create the generated key. The data parameter will be processed by the AusPayNet One Way Function to generate a new key-encrypting key.
A28OWFEC	Specifies that 16 bytes of data encrypted using the *data_decrypting_key_identifier* will be processed as described in AS2805.5.4 to create the generated key. The data parameter will be processed by the AusPayNet One Way Function to generate a new key-encrypting key. The data parameter should contain the data wrapped as follows: data = wrap(PPASN) || wrap(PPASN).
A28XOREC	Specifies that 16 bytes of data encrypted using the *data_decrypting_key_identifier* will be processed as specified in AS2805.5.4 to generate a key-encrypting key.
CLR8-ENC	Specifies that 8 bytes of clear data shall be multiply-encrypted with the generating key. The *generating_key_identifier* must be a KEYGENKY key type with bit 19 of the control vector set to 1. The control vector in *generated_key_identifier* must specify a single-length key. The key type may be DATA, MAC, or MACVER. **Note:** CIPHER class keys are not supported.
SESS-XOR	Modifies an existing DATA, DATAC, MAC, DATAM, MACVER, or DATAMV single-length or double-length key. Specifies the VISA method for session key generation. Data supplied may be 8 or 16 bytes of data depending on whether the *generating_key_identifier* is a single or double length key. The 8 or 16 bytes of data is XORed with the clear value of the *generating_key_identifier*. The *generated_key_identifier* has the same control vector as the *generating_key_identifier*. The *generating_key_identifier* may be DATA/DATAC, MAC/DATAM or MACVER/DATAMV key types.
TDES-DEC	Data supplied may be 8 or 16 bytes of clear data. If the *generated_key_identifier* specifies a single length key, 8 bytes of data is TDES decrypted under the *generating_key_identifier*. If the *generated_key_identifier* specifies a double length key, 16 bytes of data is TDES ECB mode decrypted under the *generating_key_identifier*. No formating of data is done prior to encryption. The *generating_key_identifier* must be a DKYGENKY key type, with appropriate usage bits for the desired generated key.

Table 38. Rule Array Keywords for Diversified Key Generate (continued)

Keyword	Meaning
TDES-ENC	Data supplied may be 8 or 16 bytes of clear data. If the *generated_key_identifier* specifies a single length key, 8 bytes of data is TDES encrypted under the *generating_key_identifier*. If the *generated_key_identifier* specifies a double length key, 16 bytes of data is TDES ECB mode encrypted under the *generating_key_identifier*. No formatting of data is done prior to encryption. The *generating_key_identifier* must be a DKYGENKY key type, with appropriate usage bits for the desired generated key. The *generated_key_identifier* may be a single or double length key with a CV that is permitted by the *generating_key_identifier*.
TDES-CBC	Data supplied must be 16 bytes of clear data. The *generated_key_identifier* must specify a double length key and the 16 bytes of data is TDES-CBC mode encrypted under the *generating_key_identifier*. No formatting of data is done prior to encryption. The *generating_key_identifier* must be a DKYGENKY key type, with appropriate usage bits for the desired generated key. The *generated_key_identifier* must be a double length key with a CV that is permitted by the *generating_key_identifier*.
TDES-XOR	Combines the function of the existing TDES-ENC and SESS-XOR into one step. The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, MAC, MACVER, SMPIN and SMKEY. Key type must be allowed by the generating key control vector.
TDESEMV2	Supports generation of a session key by the EMV 2000 algorithm (This EMV2000 algorithm uses a branch factor of 2). The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, MAC, MACVER, SMPIN and SMKEY. Key type must be allowed by the generating key control vector.
TDESEMV4	Supports generation of a session key by the EMV 2000 algorithm (This EMV2000 algorithm uses a branch factor of 4). The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, MAC, MACVER, SMPIN and SMKEY. Key type must be allowed by the generating key control vector.
Key Wrapping Method (optional)	
USECONFG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword. The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Table 38. Rule Array Keywords for Diversified Key Generate (continued)	
Keyword	**Meaning**
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method with SHA-256 and CMAC authentication code.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.
Translation Control (optional)	
ENH-ONLY	Restrict rewrapping of the *key_identifier* token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method. This is the default when the wrapping method is WRAPENH3.

generating_key_identifier

Direction	Type
Input/Output	String

The identifier of the *key-generating* key. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage.

The requirements for the key are:

Table 39. Key identifier requirements		
Processing method keyword	**CCA key token (64-byte)**	**X9.143 (TR-31) key block (variable-length)**
A28OWFCL	Double-length DES CIPHER key	Key usage D0, algorithm T, and mode of use B.
A28OWFEC	Double-length DES EXPORTER or DES CIPHER key	Either: • key usage D0, algorithm T, and mode of use B. • key usage K0, algorithm T, and mode of use E.
A28XOREC	Double-length DES EXPORTER key	Key usage K0, algorithm T, and mode of use E.
CLR8-ENC	DES KEYGENKY with key usage attribute CLR8-ENC enabled	Key usage B3, algorithm T, and mode of use X. Optional block 'DA' is required.
SESS-XOR	DATA, DATAC, MAC, DATAM, MACVER, or DATAMV single-length or double-length key	Either: • key usage D0, algorithm D or T, and mode of use B, D, or E. • key usage M*, algorithm D or T, and mode of use C, G, or V.

Table 39. Key identifier requirements (continued)

Processing method keyword	CCA key token (64-byte)	X9.143 (TR-31) key block (variable-length)
TDES-DEC TDES-ENC TDES-CBC	DKYGENKY key	Key usage B3, algorithm T, and mode of use X. Optional block 'DA' is required.
TDES-XOR TDESEMV2 TDESEMV4	DKYGENKY key	Key usage B3, algorithm T, and mode of use X. Optional block 'DA' is required.

When the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

data_length

Direction	Type
Input	Integer

The length of the *data* parameter in bytes. The required length depends on the diversification process specified in the rule array and the length of the key identified by the *generated_key_identifier* parameter:

Rule-array keyword	Key length of generated key	Required data length
CLR8-ENC	SINGLE	8
A28OWFCL	DOUBLE	16
A28OWFEC	DOUBLE	16
A28XOREC	DOUBLE	16
TDES-CBC	DOUBLE or null key-token	16
TDES-ENC	DOUBLE or null key-token	16
	SINGLE	8
TDES-DEC	DOUBLE or null key-token	16
	SINGLE	8
TDESEMV2, TDESEMV4	DOUBLE	10, 18, 26, or 34
TDES-XOR	DOUBLE	10 or 18
SESS-XOR	DOUBLE	16
	SINGLE	8

data

Direction	Type
Input	String

Data input to the diversified key or session key generation process. Data depends on the processing method and the *generated_key_identifier*.

For TDESEMV4 or TDESEMV2, the data is either 18 bytes (36 digits) or 34 bytes 68 digits) of data comprised of:

- 16 bytes (32 digits) of card specific data used to create the card specific intermediate key (UDK) as per the TDES-ENC method. This will typically be the PAN and PAN Sequence number as per the EMV specifications
- 2 bytes (4 digits) of ATC (Application Transaction Count)
- (optional) 16 bytes (32 digits) of IV (Initial Value) used in the EMV

data_decrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the value supplied in the data parameter. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

When the processing method rule is A28OWFEC or A28XOREC, this parameter contains a CCA key token or X9.143 key block. Otherwise, this parameter must contain a 64-byte null token.

For CCA keys, the identifier is a 64-byte DES key token of key type CIPHER or DECIPHER. The key must be double-length.

For X9.143 keys, the identifier is a variable-length key block of a DES data-encrypting key: key usage D0, algorithm T, and mode of use B or D.

When the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

generated_key_identifier

Direction	Type
Input/Output	String

The key to be generated. The key will be in a fixed-length operational key token or variable-length TR-31 key block. The length of a key token will be 64 and the maximum length of a key block will be 9992.

Because there is no length field for this parameter, if a TR-31 key block will be the output, the caller **must** pass a buffer of at least 9992 bytes and then parse the TR-31 key block to get the length. It is recommended that TR-31 Parse (CSNBT31P and CSNET31P) be used for this parsing.

On input, specify a null token or an internal token or a skeleton token containing the control vector of the key to be generated or a skeleton key block with the attributes of the key to be generated. On output, this parameter contains the generated key.

Table 40. Input requirements for the key identifier

Processing method keyword	CCA key token (64-byte token)	X9.143 (TR-31) key block (variable-length block)
A28OWFCL	Internal token or skeleton token of a DES CIPHER, DES MAC with sub-type ANY-MAC, or DES IPINENC key	Internal block or skeleton block of a data-encryption key (key usage D0, algorithm T, and mode of use B), MAC key (key usage M*, algorithm T, and mode of use C, G, or V), or PIN-encrypting key (key usage P0, algorithm T, and mode of use D).
A28OWFEC	Internal token or skeleton token of a DES EXPORTER or DES CIPHER key. The supplied token must match the key supplied in the *generating_key_identifier* parameter.	Internal block or skeleton block of a key-encrypting key (key usage K0, algorithm T, and mode of use E) or data-encryption key (key usage D0, algorithm T, and mode of use B). The supplied block must match the key supplied in the *generating_key_identifier* parameter.
A28XOREC	Internal token or skeleton token of a DES EXPORTER key	Internal block or skeleton block of a key-encrypting key (key usage K0, algorithm T, and mode of use E).
CLR8-ENC	Internal token or skeleton token of a DES CIPHER, DATA, DECIPHER, ENCIPHER, MAC or MACVER key	Internal block or skeleton block of a data-encryption key (key usage D0, algorithm T, and mode of use B, D, or E) or MAC key (key usage M1 or M3, algorithm T, and mode of use C, G, V).
SESS-XOR	Null token	Null token or block.

Table 40. Input requirements for the key identifier (continued)

Processing method keyword	CCA key token (64-byte token)	X9.143 (TR-31) key block (variable-length block)
TDES-DEC TDES-ENC TDES-CBC	Null token or an internal token or skeleton token of a DES DATA, EXPORTER, IMPORTER, IKEYXLAT, OKEYXLAT, MAC, MACVER, PINVER, SECMSG, or DKYGENKY key	Null token or block or an internal block or skeleton block of a: • Data-encryption key (key usage D0, algorithm T, and mode of use B, D, or E). • MAC key (key usage M*, algorithm T, and mode of use C, G, V). • Key-encrypting key (key usage K*, algorithm T, and mode of use D or E). • PIN verification key (key usage V*, algorithm T, and mode of use C, G, or V). • Secure messaging key (key usage F1 or F2, algorithm T, and mode of use X). • Key-derivation key (key usage B3, algorithm T, and mode of use X).
TDES-XOR TDESEMV2 TDESEMV4	Internal token or skeleton token of a DES DATA, MAC, MACVER or SECMSG key	Internal block or skeleton block of a: • Data-encryption key (key usage D0, algorithm T, and mode of use B). • MAC key (key usage M*, algorithm T, and mode of use C, G, V). • Secure messaging key (key usage F1 or F2, algorithm T, and mode of use X).

To generate a compliant-tagged key token, a compliant-tagged skeleton token must be supplied. To generate a compliant-tagged key block, the IBM optional block “10” with the Compliance Tag attribute enabled must be included with the skeleton key block.

For CCA key tokens:

- When the WRAPENH3 method is selected, a skeleton key token is required. A secure internal key token wrapped with the WRAPENH3 method obfuscates the key length.
- The output *generated_key_token* will use the default wrapping method unless a rule array keyword overriding the default is specified.
- The DES key wrapping methods available are described in “CCA key wrapping” on page 20.

Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Refer to Appendix C, “Changing control vectors with the CVT callable service,” on page 1603 for information on the control vector bits for the DKG key generating key.

For Session key algorithm (EMV Smartcard specific), a master derivation key (MDK) can be used in two ways:

- To calculate the Card Specific Key (or UDK) in the personalization process, call this service with the TDES-ENC or TDES-CBC method using an output token that has been primed with the CV of the final session key, for instance, if the MDK is a DMPIN, the token should have the CV of an SMPIN key; DMAC (a double length MAC); DDATA (a double length DATA key), and so on.

  The result would then be exported in the personalization file. This key is not usable in this form for any other calculations.
- To use the session key, call this service with the TDESEMV4 method. Provide, for input, the same card data that was used to create the UDK as well as the ATC and optionally the IV value. This is the key that will be used in EMV related Smartcard processing.

  This same processing applies to those API's the generate the session key on your behalf, like CSNBPCU.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the generated key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 41. Required access control points for Diversified Key Generate

Rule array keyword	Access control point
A28XOREC	Diversified Key Generate - A28XOREC
A28OWFCL	Diversified Key Generate - A28OWFCL
A28OWFEC	Diversified Key Generate - A28OWFEC
CLR8-ENC	Diversified Key Generate - CLR8-ENC
SESS-XOR	Diversified Key Generate - SESS-XOR
TDES-DEC	Diversified Key Generate - TDES-DEC
TDES-ENC	Diversified Key Generate - TDES-ENC
TDES-CBC	Diversified Key Generate - TDES-CBC
TDES-XOR	Diversified Key Generate - TDES-XOR
TDESEMV2 or TDESEMV4	Diversified Key Generate - TDESEMV2/TDESEMV4

When the key wrapping method keyword specifies a wrapping method that is not the default method, the **Diversified Key Generate - Allow wrapping override keywords** access control must be enabled.

When a key-generating key of key type DKYGENKY is specified with control vector bits (19 - 22) of B'1111', the **Diversified Key Generate - DKYGENKY - DALL** access control point must also be enabled in the domain role.

When using the TDES-ENC or TDES-DEC modes, you can specifically enable generation of a single-length key or a double-length key with equal key-halves by enabling the **Diversified Key Generate - Single length or same halves** access control point.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 42. Diversified Key Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC). Rule array keywords A28OWFEC, A28OWFCL, and A28XOREC are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC). Rule array keywords A28OWFEC, A28OWFCL, and A28XOREC are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC). Rule array keywords A28OWFEC, A28OWFCL, and A28XOREC are not supported. X9.143 key blocks are not supported.

Table 42. Diversified Key Generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC). Rule array keywords A28OWFEC, A28OWFCL, and A28XOREC are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC). Rule array keywords A28OWFEC, A28OWFCL, and A28XOREC are not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC). Rule array keywords A28OWFEC, A28OWFCL, and A28XOREC require the CCA release 7.4 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	Rule array keywords A28OWFEC, A28OWFCL, and A28XOREC are not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Diversified Key Generate2 (CSNBDKG2 and CSNEDKG2)

The Diversified Key Generate2 callable service generates an AES key based on a function of a key-generating key, the process rule, and data that you supply.

To use this service, specify:

- The rule array keyword to select the diversification process.
- The operational AES key-generating key from which the diversified keys are generated.

For a key-generating key with a key-derivation sequence level of 1 or 2:
The type of key created will be a DKYGENKY key with a sequence level one lower than the key-generating key with the same key usage fields.

For a key-generating key with a key-derivation sequence level of 0:
Key usage field 1 determines the type of key that is generated and restricts the use of this key to the key-diversification process. If the generating key has related key usage fields 3 through field 6 defined, these key usage attributes are used to control the permitted key usage attributes for the key to be generated.

Note: Key usage field 2 of the generating DKYGENKY key contains a flag in its high-order byte. This flag byte determines how key usage fields 3 and beyond (called the related generated key usage fields) are used to control the values of the key usage fields of the generated key:

- When the type of key to diversify is D-ALL, the flag is undefined because there are no key usage restrictions on the generated key. The generating key has no related generated key usage fields.
- When the type of key to diversify is not D-ALL and the flag byte has KUF-MBE usage, the key usage fields of the key to be generated must be equal to the related generated key usage fields that start with key usage field 3 of the generating key.
- When the type of key to diversify is not D-ALL and the flag byte has KUF-MBP usage, the key usage fields of the key to be generated must be permissible. In other words, a key to be diversified is only permitted to have a level of usage less than or equal to the related key usage fields (key usage fields starting with key usage field 3). One exception is that the UDX-only setting of the generated key always must be equal to the UDX-ONLY setting of the generating key.

- The diversification data and length of data used in the diversification process.
- The variable-length AES symmetric-key generated token with a suitable key type and key usage fields for receiving the diversified key, or a null key token if the type of key to diversify supports default key usage and a default key is desired.

The callable service name for AMODE(64) invocation is CSNEDKG2.

Format

```
CALL CSNBDKG2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             generating_key_identifier_length,
             generating_key_identifier,
             derivation_data_length,
             derivation_data,
             input_initial_vector_length,
             input_initial_vector,
             reserved2_length,
             reserved2,
             generated_key_identifier1_length,
             generated_key_identifier1,
             generated_key_identifier2_length,
             generated_key_identifier2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1, 2, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 43. Rule array keywords for Diversified Key Generate2

Keyword	Meaning
Diversification Process (required)	
KDFFM-DK	Specifies to use the DK version of key derivation function in feedback mode. This method uses AES CMAC to encipher the derivation data with the *k*-bit diversified key generating key (banking association specific master key) to produce a *k*-bit generated bank specific Issuer Master Key, where *k* = 128, 192, or 256.
MK-OPTC	Specifies to use the EMV master key derivation option C specified in *EMV Integrated Circuit Card Specifications for Payments Systems*. This method uses AES in ECB mode to encipher the 16 bytes of derivation data with the *k*-bit diversified key generating key (Issuer Master Key) to produce a *k*-bit generated ICC master key, where *k* = 128, 192, or 256.

Table 43. Rule array keywords for Diversified Key Generate2 (continued)	
Keyword	**Meaning**
SESS-ENC	Specifies to use the EMV common session key derivation option specified in *EMV Integrated Circuit Card Specifications for Payments Systems*. This method uses AES in ECB mode to encipher the 16 bytes of derivation data with the *k*-bit diversified key generating key (ICC master key) to produce a *k*-bit generated key (ICC session key), where *k* = 128, 192, or 256.
Bit length of generated key (one, optional). Valid only with the KDFFM-DK keyword. Default is to use the bit length of the generating key as the bit length of the generated key.	
KLEN128	Specifies the bit length of the generated key to be 128.
KLEN192	Specifies the bit length of the generated key to be 192, allowed if and only if the bit length of the generating key is greater than or equal to 192.
KLEN256	Specifies the bit length of the generated key to be 256, allowed if and only if the bit length of the generating key is 256.
IV usage (one, optional). Valid only with process keyword KDFFM-DK.	
DEFLT-IV	Specifies to use the DK default initial vector value as the IV in the derivation function. This is the default value.
USE-IV	Specifies to use the value specified in the *input_initial_vector* parameter as the IV in the derivation function.

generating_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *generating_key_identifier* parameter. If the *generating_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 9992.

generating_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-generating key. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA keys:

- The key algorithm of this key must be AES and the key type must be DKYGENKY. The key usage field indicates the key type of the generated key. The key length determines the length of the generated key.
- If SESS-ENC is specified, the clear length of the generated key is equal to the clear length of the generating key. If SESS-ENC is specified, the key-derivation sequence level must be set to DKYL0 in the key usage field 2.

For X9.143 (TR-31) keys, the key usage must B3, the DA optional block is required, the algorithm must be A, and the mode of use is X. Optional block "DA" is required.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

derivation_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *derivation_data* parameter. If SESS-ENC or MK-OPTC is specified, the value must be 16.

When the process rule KDFFM-DK is specified, the value must be between 1 to 2048 inclusive for CCA release 6.7, 7.4, and later. Otherwise, the value must be 16 to 40 inclusive.

derivation_data

Direction	Type
Input	String

The derivation data to be used in the key generation process. This data is often referred to as the diversification data. For SESS-ENC, the derivation data is 16-bytes long.

Note that if SESS-ENC is specified and the length of the key generating key is 192 bits or 256 bits, the data is manipulated in conformance with the EMV Common Session Key Derivation Option.

input_initial_vector_length

Direction	Type
Input	Integer

Length in bytes of the *input_initial_vector* parameter. For CCA releases 6.7, 7.4 and later and when the KDFFM-DK process rule and the USE-IV keywords are specified, the value must be between 0 or 16 inclusive. Otherwise, the value must be 0.

input_initial_vector

Direction	Type
Input	String

For the KDFFM-DK process rule, the 16-byte initial vector value for the algorithm When a value is not provided, the default value, 0x52525252525252522525252525252525, will be used. When the USE-IV keyword is specified and the *input_initial_vector_length* is 0, the initial value will be hexadecimal zero.

When the *input_initial_vector_length* is zero, this field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

generated_key_identifier1_length

Direction	Type
Input/Output	Integer

On input, the length of the buffer for the *generated_key_identifier1* parameter in bytes. The maximum value is 9992 bytes.

On output, the parameter holds the actual length of the *generated_key_identifier1* parameter.

generated_key_identifier1

Direction	Type
Input/Output	String

The buffer for the variable-length generated key token.

On input, the buffer contains a null token or a valid internal 64-byte skeleton CCA key token or internal 16-byte skeleton TR-31 key block containing the desired key-usage fields and key-management fields you want to generate. The key token must be left justified in the buffer.

The generating key (*generating_key_identifier* parameter) determines whether on input the *generated_key_identifier1* parameter can identify a null key token or a skeleton key token.

When the *generating_key_identifier* is compliant-tagged, a compliant-tagged key token will be created.

When a skeleton key token is passed as input and the *generating_key_identifier* is compliant-tagged, the skeleton token must have the compliant-tagged flag on. When a skeleton key block is passed as input and the *generating_key_identifier* is compliant-tagged, the skeleton key block must include the IBM optional block "10" with the Compliance Tag attribute enabled.

Table 44. Summary of input generating key tokens, input generated key tokens, and output generated key tokens

Input generating key token	Input generated key token	Output generated key token
DKYL0, type of key to diversify D-ALL or key block with multiple entries in the DA optional block	Skeleton key token or block required.	Key type same as skeleton, diversified key final.
DKYL0, type of key to diversify not D-ALL or key block with a single entry in the DA optional block	Null or skeleton key token or block allowed.	Key type determined by input generated key token type of key to diversify. If null key token on input, the output key token will have attributes based on the related generated key usage fields of the input generating key token. Otherwise, the output key token will have attributes of input skeleton key token.
DKYL1, any type of key to diversify	Null key token required.	Same as input generating key token except DKYL0 and with new level of diversified key.
DKYL2, any type of key to diversify	Null key token required.	Same as input generating key token except DKYL1 and with new level of diversified key.

Notes:

1. If the supplied generated key-token or block contains a key, the key value and length are ignored and overwritten.

2. If the *generating_key_identifier1* parameter identifies a DKYGENKY key token with a key-derivation sequence level of DKYL0 and it does not have a type of key to diversify of D-ALL, the key type must match what the generating key indicates can be created in the key generating key usage field at offset 45.
3. The key usage fields in the generated key must meet the requirements (KUF 'must be equal' or 'must be permitted') of the corresponding key usage fields in the generating key unless D-ALL is specified in the generating key. A flag bit in the DKYGENKY key-usage field 2 determines whether the key-usage field level of control is KUF-MBE or KUF-MBP.
4. If authorized by access control, D-ALL permits the derivation of several different keys.

On output, the buffer contains the generated key token.

generated_key_identifier2_length

Direction	Type
Input/Output	Integer

Length in bytes of the *generated_key_identifier2* parameter. The value must be 0.

generated_key_identifier2

Direction	Type
Input/Output	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the generated key.

Access control points

The following table shows the access control points in the domain role that control the function of this service:

Table 45. Required access control points for Diversified Key Generate2

Rule array keyword	Access control point
KDFFM-DK	Diversified Key Generate2 - KDFFM-DK
MK-OPTC	Diversified Key Generate2 - MK-OPTC
SESS-ENC	Diversified Key Generate2 - SESS-ENC

To use the KLEN192 and KLEN256 keywords, the **Diversified Key Generate2 - Allow length option with KDFFM-DK** access control point must be enabled.

If the key-generating key key-usage fields indicate that all key types may be derived, the **Diversified Key Generate2 – DALL** access control point must be enabled in the domain role.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 46. Diversified Key Generate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords KDFFM-DK, MK-OPTC, KLEN128, KLEN192, and KLEN256 require the June 2015 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The *input_initial_vector_length* parameter value must be 0. The *derivation_data_length* must not exceed 40. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Keywords KDFFM-DK, MK-OPTC, KLEN128, KLEN192, and KLEN256 require the June 2015 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The *input_initial_vector_length* parameter value must be 0. The *derivation_data_length* must not exceed 40. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The *input_initial_vector_length* and *derivation_data_length* parameter support requires the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The *input_initial_vector_length* parameter value must be 0. The *derivation_data_length* must not exceed 40. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	The *input_initial_vector_length* and *derivation_data_length* parameter support requires the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	The *input_initial_vector_length* and *derivation_data_length* parameter support requires the CCA release 7.4 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 46. Diversified Key Generate2 required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Diversify Directed Key (CSNBDDK and CSNEDDK)

The Diversify Directed Key callable service is used to selectively generate and derive a pair of associated keys in connection with a directed key diversification key scheme. The objective of the concept is to generate and derive a key pair with different key usages from one key diversification key (KDK). Key direction comes into play in that one of the keys is generated and is used for one direction (for example, encryption, MAC generate, and so forth), while the other key is derived and will have usage associated with a different direction (for example, decryption, MAC verification, and so forth). This callable service provides an option to perform the generate or derive operation.

A structure called a key type vector, which is always used as the initialization vector for the diversification process, is passed in as input and is used to determine what and how the key is produced by this callable service.

The key generated by this callable service is used as a session key. The intention in this context is that the keys of a generated and derived key pair are one-time keys. The key management fields of the output key will indicate that the key cannot be exported.

The callable service name for AMODE(64) invocation is CSNEDDK.

Format

```
CALL CSNBDDK(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              kdk_key_identifier_length,
              kdk_key_identifier,
              key_type_vector_length,
              key_type_vector,
              additional_derivation_data_length,
              additional_derivation_data,
              random_data_length,
              random_data,
              output_key_identifier_length,
              output_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Output	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 2.

rule_array

Direction	Type
Input	Character

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 47. Keywords for Diversify Directed Key

Keyword	Meaning
Diversification Process (One required)	
KDFFM	Specifies to use the Key Derivation Function (KDF) in Feedback Mode (NIST SP 800-108) to generate key. The key type vector is used as the IV for this process.
Function (one required)	
DERIVE	Specifies to derive the passive diversified key of a pair of directed keys.
GENERATE	Specifies to generate the active diversified key of a pair of directed keys.

kdk_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *kdk_key_identifier* parameter. If the *kdk_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 725.

kdk_key_identifier

Direction	Type
Input/Output	String

The identifier of the key diversification key used to derive keys. The key identifier is an operational token or the key label of an operational token in key storage.

The key algorithm of this key must be AES and the key type must be KDKGENKY. The key usage fields indicate the type of key to diversify and if the key is to be derived for entity A or entity B.

Note: When the GENERATE function is specified and the generating key has usage of KDKTYPEA, the associated DERIVE function must have usage of KDKTYPEB. Likewise, when the GENERATE function is specified and this key has usage of KDKTYPEB, the associated DERIVE function must have usage of KDKTYPEA.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

key_type_vector_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *key_type_vector* parameter. The value must be 16.

key_type_vector

Direction	Type
Input	String

The 16-byte *key_type_vector* specifies the rules for the calculation of the key value to be generated or derived and contains information needed to restrict the usage of the key to be generated or derived. The format of the structure is as follows:

Offset	Length	Description
0	2	Version number X'0000'.

Offset	Length	Description
2	2	Type of key to be derived or generated. **Value** **Meaning** **X'0000'** MAC. **X'0001'** Data encryption (cipher). **X'0003'** PIN encryption. **X'0004'** Key wrapping. All other values are reserved and undefined.
4	2	Key algorithm. **Value** **Meaning** **X'0002'** AES. All other values are reserved and undefined.
6	2	Length of the key to be derived or generated in bits. **Value** **Meaning** **X'0800'** 2048 (for example, AES-256).

Offset	Length	Description
8	2	**Key usage restriction 1** of the key to be derived or generated, based on the key type field (value at offset 2): **For MAC key type** (Value at offset 2 = X'0000'). **Value** **Meaning** **X'0001'** Key can derive or generate a CMAC mode key only. All other values are reserved and undefined. **For data encryption (cipher) key type** (Value at offset 2 = X'0001'). **Value** **Meaning** **X'0002'** Key can derive or generate a CBC mode key only. All other values are reserved and undefined. **For PIN encryption key type** (Value at offset 2 = X'0003'). **Value** **Meaning** **X'0000' or X'0002'** Key can derive or generate an ISO-4 format key only. See the note for KTVs for this key type. All other values are reserved and undefined. **For Key wrap key type** (Value at offset 2 = X'0004'). **Value** **Meaning** **X'0001'** Key can derive or generate a VARDRV-D key only. All other values are reserved and undefined. **For all other key types not listed above:** **Value** **Meaning** **X'0000'** No key usage restriction 1. All other values are reserved and undefined.

Offset	Length	Description
10	2	**Key usage restriction 2** of the key to be derived or generated, depending on the key type (offset 2) and key usage restriction 1 (offset 8): **MAC key type and HMAC mode** (Value at offset 2 = X'0000' and offset 8 = X'0001'). **Value** **Meaning** **X'0002'** SHA-256. **X'0003'** SHA-384. **X'0004'** SHA-512. All other values are reserved and undefined. **Key-wrap key type and VARDRV-D mode** (Value at offset 2 = X'0004' and offset 8 = X'0001'). **Value** **Meaning** **X'0100'** Maximum key length of the protected keys is 2048 bits. All other values are reserved and undefined. **All other values** at offset 2 and offset 8. **Value** **Meaning** **X'0000'** Undefined. All other values are reserved and undefined. **For all other key types and key usage restriction 1 combinations not listed above:** **Value** **Meaning** **X'0000'** No key usage restriction 2. All other values are reserved and undefined.
12	3	Reserved, must be binary zero.

Offset	Length	Description
15	1	**Key direction variant indicator** Diversifies the key to be derived or generated depending on the permitted use of direction. The HSM has to restrict the usage of the key depending on this value and the type of entity (A or B) which is an additional parameter in the process of deriving or generating the key. This value affects a key usage attribute of the key to be derived or generated. **Value** **Meaning** **X'00'** A ↔ B (undirected use of key). **X'01'** A → B (A active, B passive use of key). **X'10'** A ← B or equivalent (A passive, B active use of key). **X'FF'** System is to determine key direction from entity usage of the KDKGENKY and rule array keywords. **KDK-A + GENERATE rule array keyword** Direction set to X'01' (A → B). **KDK-B + GENERATE rule array keyword** Direction set to X'10' (A ← B). **KDK-A + DERIVE rule array keyword** Direction set to X'10' (A ← B). **KDK-B + DERIVE rule array keyword** Direction set to X'01' (A → B). All other values are reserved and undefined.

The following tables define the valid KTV supported.

For each of the four key types defined at KTV offset 2 (MAC, data encryption, PIN encryption, and key wrapping), there are two KTVs defined, with the only difference between them being the key direction variant indicator (KTV offset 15). Either entity Type A is active and Type B is passive (A→B), or Type B is active and Type A is passive (A←B). See Table 48 on page 176 for additional information.

Table 48. Summary of KTV tables

A→B or A←B	MAC generate/ verify	Data encrypt/ decrypt	PIN encrypt/ decrypt	Key wrap/unwrap
A→B (A active)	KTVM1 Table 49 on page 177	KTVC1 Table 51 on page 177	KTVP1 Table 55 on page 178	KTVW1 Table 57 on page 179
A←B (B active)	KTVM2 Table 50 on page 177	KTVC2 Table 52 on page 177	KTVP2 Table 56 on page 178	KTVW2 Table 58 on page 179

Table 49. KTV for MAC generate/verify, Type A active and Type B passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	MAC	AES	AES-256	CMAC	'else'	-	A→B
00	00 00	00 02	01 00	00 01	00 00	00 00 00	01
KTVM1 = X'00 00 00 00 00 02 01 00 00 01 00 00 00 00 00 01'							

Table 50. KTV for MAC generate/verify, Type B active and Type A passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	MAC	AES	AES-256	CMAC	'else'	-	A←B
00	00 00	00 02	01 00	00 01	00 00	00 00 00	10
KTVM2 = X'00 00 00 00 00 02 01 00 00 01 00 00 00 00 00 10'							

Table 51. KTV for data encryption (cipher), Type A active and Type B passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	Cipher	AES	AES-256	CBC	'else'	-	A→B
00	00 01	00 02	01 00	00 02	00 00	00 00 00	01
KTVC1 = X'00 00 00 01 00 02 01 00 00 02 00 00 00 00 00 01'							

Table 52. KTV for data encryption (cipher), Type B active and Type A passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	Cipher	AES	AES-256	CBC	'else'	-	A←B
00	00 01	00 02	01 00	00 02	00 00	00 00 00	10
KTVC2 = X'00 00 00 01 00 02 01 00 00 02 00 00 00 00 00 10'							

For PIN encryption key type, the key usage indicator 1 (offset 8) for ISO-4 format can have the value of '0000' or '0002' for a pair of KTVs. The caller is not allowed to mix pairs of KTVs because the KTV is used as the IV in the key creating process. This is the responsibility of the caller of the service.

KTV pair for PIN encryption key type with key usage indicator 1 with '0000' value.

Table 53. KTV for PIN encryption, Type A active and Type B passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	PIN-Enc	AES	AES-256	ISO-4	'else'	-	A→B
00	00 03	00 02	01 00	00 00	00 00	00 00 00	01
KTVP1 = X'00 00 00 03 00 02 01 00 00 00 00 00 00 00 00 01'							

Table 54. KTV for PIN encryption, Type B active and Type A passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	PIN-Enc	AES	AES-256	ISO-4	'else'	-	A←B
00	00 03	00 02	01 00	00 00	00 00	00 00 00	10
KTVP2 = X'00 00 00 03 00 02 01 00 00 02 00 00 00 00 00 10'							

KTV pair for PIN encryption key type with key usage indicator 1 with '0002' value.

Table 55. KTV for PIN encryption, Type A active and Type B passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	PIN-Enc	AES	AES-256	ISO-4	'else'	-	A→B
00	00 03	00 02	01 00	00 02	00 00	00 00 00	01
KTVP1 = X'00 00 00 03 00 02 01 00 00 02 00 00 00 00 00 01'							

Table 56. KTV for PIN encryption, Type B active and Type A passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	PIN-Enc	AES	AES-256	ISO-4	'else'	-	A←B
00	00 03	00 02	01 00	00 02	00 00	00 00 00	10
KTVP2 = X'00 00 00 03 00 02 01 00 00 02 00 00 00 00 00 10'							

For Table 57 on page 179, entity Type A must use an AES EXPORTER key with usage of EXPTT31D, while entity Type B must use an IMPORTER key with usage of IMPTT31D.

Key wrapping with key block protection (ISO TC 68/SC 2 Nxxxx, 2016-08-17, ISO DIS 20038):

Table 57. KTV for key wrapping, Type A active and Type B passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	Key wrap	AES	AES-256	VARDRV-D	Maximum bit length of the protected keys	-	A→B
00	00 04	00 02	01 00	00 01	01 00	00 00 00	01
KTVW1 = X'00 00 00 04 00 02 01 00 00 01 01 00 00 00 00 01'							

For Table 58 on page 179, entity Type B must use an AES EXPORTER key with usage of EXPTT31D, while entity Type A must use an IMPORTER key with usage of IMPTT31D.

Table 58. KTV for key wrapping, Type B active and Type A passive

Version (offset 0)	Key type indicator (offset 2)	Algorithm indicator (offset 4)	Key length (offset 6)	Key usage indicator 1 (offset 8)	Key usage restriction 2 (offset 10)	RFU (offset 12)	Key direction variant indicator (offset 15)
0	Key wrap	AES	AES-256	VARDRV-D	Maximum bit length of the protected keys	-	A←B
00	00 04	00 02	01 00	00 01	01 00	00 00 00	10
KTVW2 = X'00 00 00 04 00 02 01 00 00 01 01 00 00 00 00 10'							

additional_derivation_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *additional_derivation_data* parameter. The value must be between 0 and 2032 inclusive for CCA release 6.7, 7.4, and later. Otherwise, the value must be 0 and 24 inclusive.

The sum of the *additional_derivation_data_length* and the *random_data_length* cannot exceed 2048.

additional_derivation_data

Direction	Type
Input	String

Data to be used in the key generation or key derivation process.

The additional derivation data concatenated with the random data cannot exceed 2048 for CCA releases 6.7, 7.4, and later. Otherwise, the random data cannot exceed 40 bytes.

random_data_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *random_data* parameter. The value must be between 16 and 40 inclusive. The sum of the *additional_derivation_data_length* and the *random_data_length* cannot exceed 2048 for CCA releases 6.7, 7.4, and later. Otherwise, the random data cannot exceed 40 bytes.

When keyword GENERATE is specified in the rule array, this is an input and an output parameter. On input, this value specifies the number of bytes of data to use as the random data portion of the diversification data used in diversifying the first key of a key pair. On output, the returned value indicates the number of bytes of data actually returned in the *random_data* variable.

When keyword DERIVE is specified in the rule array, this is an input only parameter. On input, this value specifies the number of bytes of data to use as the random data portion of the diversification data used in diversifying the second key of a key pair. To produce the desired results, this value must be the same length returned by a previous associated GENERATE function call.

random_data

Direction	Type
Input/Output	String

The random data used in the diversification process.

When the GENERATE function is specified, on input, this variable is ignored, and on output, this variable contains the random data created and used to diversify the first output key of a key pair.

When the DERIVE function is specified, on input, this variable must contain the random data previously created during a previous GENERATE function and is used to diversify the second output key of a key pair.

Note: For a given pair of output keys, the DERIVE function must provide the same random data and *additional_derivation_data* value as the GENERATE function used.

The additional derivation data concatenated with the random data cannot exceed 2048 for CCA releases 6.7, 7.4, and later. Otherwise, the random data cannot exceed 40 bytes.

output_key_identifier_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the buffer for the *output_key_identifier* parameter. On input, the value is the size of the buffer. The maximum length is 725. On output, the value is the length of the key token returned in the *output_key_identifier* parameter.

output_key_identifier

Direction	Type
Input/Output	String

The buffer to receive the generated key. The key attributes are identified by the *key_type_vector* parameter.

On input, the buffer must contain a variable-length symmetric null key token (see Table 625 on page 1537), left justified in the buffer. The rest of the buffer is copied, but not checked.

When the *kdk_key_identifier* is compliant-tagged, a compliant-tagged key token will be created.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **Diversify Directed Key** access control point in the domain role controls the function of this service.

The access controls for rule array keywords are listed in the table:

Diversification process rule-array keyword	Function rule-array keyword	Access control
KDFFM	DERIVE	Diversify Directed Key - allow KDFFM DERIVE
	GENERATE	Diversify Directed Key - allow KDFFM GENERATE

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 59. Diversify Directed Key required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	This service requires the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The *additional_derivation_data_length* must not exceed 24.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	This service requires the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The *additional_derivation_data_length* must not exceed 24.
	Crypto Express6 CCA Coprocessor	This service requires the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The *additional_derivation_data_length* parameter support requires the CCA release 6.7 or later licensed internal code (LIC).

Table 59. Diversify Directed Key required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The *additional_derivation_data_length* must not exceed 24.
	Crypto Express6 CCA Coprocessor	The *additional_derivation_data_length* parameter support requires the CCA release 6.7 or later licensed internal code (LIC).
	Crypto Express7 CCA Coprocessor	The *additional_derivation_data_length* parameter support requires the CCA release 7.4 or later licensed internal code (LIC).
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

ECC Diffie-Hellman (CSNDEDH and CSNFEDH)

Use the ECC Diffie-Hellman callable service to create:

- Symmetric key material from a pair of ECC keys using the Elliptic Curve Diffie-Hellman protocol and the static unified model key agreement scheme.
- "Z" - The "secret" material output from D-H process.
- Symmetric key material from a Hybrid Quantum Safe Algorithm (QSA) Key Exchange Scheme involving a CRYSTALS-KYBER encrypted value or an AES encrypted value and a pair of ECC keys using the Elliptic Curve Diffie-Hellman protocol.

Output may be one of the following forms:

- Internal CCA Token (DES or AES): AES keys are in the "Variable-length Symmetric Key Token" format. DES keys are in the "DES Internal Key Token" format.
- External CCA Token (DES or AES): AES keys are in the "Variable-length Symmetric Key Token" format. DES keys are in the "DES External Key Token" format.
- Internal TR-31 key block: DES or AES.
- External TR-31 key block: DES or AES.
- "Z" - The "secret" material output from D-H process.

The callable service name for AMODE(64) invocation is CSNFEDH.

Format

```
CALL CSNDEDH(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              private_key_identifier_length,
              private_key_identifier,
              private_KEK_key_identifier_length,
```

```
                private_KEK_key_identifier,
                public_key_identifier_length,
                public_key_identifier,
                hybrid_key_identifier_length,
                hybrid_key_identifier,
                party_identifier_length,
                party_identifier,
                key_bit_length,
                initialization_vector_length,
                initialization_vector,
                hybrid_ciphertext_length,
                hybrid_ciphertext,
                reserved3_length,
                reserved3,
                reserved4_length,
                reserved4,
                reserved5_length,
                reserved5,
                output_KEK_key_identifier_length,
                output_KEK_key_identifier,
                output_key_identifier_length,
                output_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be from 1 to 8 inclusive.

rule_array

Direction	Type
Input	String

The *rule_array* parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The rule_array keywords are:

Table 60. Keywords for ECC Diffie-Hellman

Keyword	Meaning
Scheme (one, optional)	
ECDH	Specifies to follow the Elliptic Curve Diffie Hellman Key Agreement Scheme. This is the default.
QSA-ECDH	Specifies to follow the Hybrid QSA Key Exchange Scheme. The *hybrid_key_identifier* parameter must contain the key used to decrypt the value in the *hybrid_ciphertext* parameter.
Key agreement (one required)	
DERIV01	Use input skeleton key-token and derive one element of any key pair. Denotes ANSI X9.63 protocol static unified model key-agreement scheme (see NIST SP800-56A). Initiator and responder must have a sufficient level of trust such that they each derive only one element of any key pair. The DERIV01 rule is designed for CCA to CCA interaction.
DERIV02	Use input skeleton key-token and derive one element of any key pair. Denotes key derivation function ANSI-X9.63-KDF (see Section 5.6.3 of ANSI X9.63-2011). Initiator and responder must have a sufficient level of trust such that they each derive only one element of any key pair.
PASSTHRU	Skip Key derivation step and return raw “Z" material.
Hybrid Scheme Encrypted Value Key Type (one required for QSA-ECDH, when hybrid key identifier is present)	
IHKEYKYB	The hybrid key identifier is a CRYSTALS-Kyber private key token.
IHKEYAES	The hybrid key identifier is an AES key token.
Transport Key Type (one optional if output KEK key identifier is present)	
OKEK-DES	The output KEK key identifier is a "DES" KEK token.
OKEK-AES	The output KEK key identifier is a "AES" KEK token.
Output Key Type (one optional if output key identifier is present)	
KEY-DES	The output key identifier is a "DES" skeleton token.
KEY-AES	The output key identifier is an "AES" skeleton token.
Hash type (one optional, only valid with DERIV02)	
SHA-224	Specifies the use of the SHA-224 method.
SHA-256	Specifies the use of the SHA-256 method. This is the default.
SHA-384	Specifies the use of the SHA-384 method.

Table 60. Keywords for ECC Diffie-Hellman (continued)	
Keyword	**Meaning**
SHA-512	Specifies the use of the SHA-512 method.
***Key Wrapping Method (optional)*. Valid for DES CCA keys only.**	
USECONFG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method with SHA-256. This is the default for triple-length keys.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method with SHA-256 and CMAC authentication code.
WRAP-ECB	Specifies that the original wrapping method is to be used.
***Translation Control (optional)*. Valid for DES CCA keys only.**	
ENH-ONLY	Restrict rewrapping of the *key_identifier* token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method. This is the default when the wrapping method is WRAPENH2 or WRAPENH3.

private_key_identifier_length

Direction	Type
Input	Integer

The length of the *private_key_identifier* parameter in bytes. If the *private_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 3500.

private_key_identifier

Direction	Type
Input	String

The *private_key_identifier* must contain an internal or an external token or a label of an internal or external ECC key. The ECC key token must contain a public-private key pair. When the key agreement keyword is DERIV01, a clear key will be accepted.

The ECC curve type and size must be the same as the type (Prime, Brainpool, or Koblitz) and size of the ECC key-token specified by the public key identifier parameter. The key-usage flag byte (offset 50 in the private-key section) of the ECC key-token identified by the private key identifier parameter must permit key establishment (either KEY-MGMT or KM-ONLY).

For keyword DERIV02, the key identifier must contain a key-derivation section, type X'23' (see key-derivation in Section 4.2 and Table 14 of ANSI X9.63-2011).

private_KEK_key_identifier_length

Direction	Type
Input	Integer

The length of the *private_KEK_key_identifier* in bytes. If the *private_KEK_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 9992. If the *private_key_identifier* contains an internal ECC token, this value must be a zero.

private_KEK_key_identifier

Direction	Type
Input	String

The key-encrypting key to unwrap the ECC private key token. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

When *private_KEK_key_identifier_length* is zero, this parameter is ignored.

For CCA keys, the identifier is a variable-length AES key token of key type EXPORTER or IMPORTER with key management attributes enable to allow the key to wrap an AES key.

For X9.143 (TR-31) keys, the identifier is a variable-length AES key block of a key-encrypting key: key usage K0, algorithm A, and the mode of use D or E.

If the token or key block supplied was encrypted under the old master key, the token or key block is returned encrypted under the current master key.

public_key_identifier_length

Direction	Type
Input	Integer

The length of the *public_key_identifier* in bytes. If the *public_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 3500.

public_key_identifier

Direction	Type
Input	String

The *public_key_identifier* parameter must contain an ECC public token or the label of an ECC Public token. The *public_key_identifier* specifies the other party's ECC public key which is enabled for key management functions. If the *public_key_identifier* identifies a token containing a public-private key pair, no attempt to decrypt the private part will be made.

hybrid_key_identifier_length

Direction	Type
Input	Integer

The length of the *hybrid_key_identifier* in bytes.

When the rule array keyword is neither IHKEYKYB nor IHKEYAES, the value must be zero.

When the *hybrid_key_identifier* contains a label, the value must be 64.

When *hybrid_key_identifier* contains an AES CIPHER token, the value must be actual length of the token and 9992.

When *hybrid_key_identifier* contains a CRYSTALS-Kyber private token, the value must be between the actual length of the token and 8000.

hybrid_key_identifier

Direction	Type
Input/Output	String

The identifier of the key used to decrypt the *hybrid_ciphertext* parameter. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

When *hybrid_key_identifier_length* is zero, this parameter is ignored.

When IHKEYKYB is specified, the *hybrid_key_identifier* parameter must contain a CRYSTALS-Kyber private key. The private key must have the U-DATENC capability.

When IHKEYAES is specified, the *hybrid_key_identifier* parameter must contain an AES CIPHER key.

- For CCA keys, the identifier is a variable-length AES key token of key type CIPHER with DECRYPT capability and the encryption mode attribute CBC.
- For X9.143 (TR-31) keys, the identifier is a variable-length AES key block of a key-encrypting key: key usage D0, algorithm A, and mode of use D or B.

If the token or key block supplied was encrypted under the old master key, the token or key block is returned encrypted under the current master key.

party_identifier_length

Direction	Type
Input/Output	Integer

The length of the *party_identifier* parameter in bytes. For the DERIV01 keyword, the value must be between 8 and 64, inclusive. For the DERIV02 keyword, the value must be between 0 and 256, inclusive. When the PASSTHRU rule array keyword is specified, the value must be 0 and the *party_identifier* parameter is ignored.

party_identifier

Direction	Type
Input/Output	String

The *party_identifier* parameter contains the entity identifier information. This information should contain the both entities data according to NIST SP800-56A Section 5.8 when the DERIV01 rule array keyword is specified. For DERIV02, this information should contain the optional shared data according to Section 5.6.3 of ANSI X9.63-2011.

key_bit_length

Direction	Type
Input/Output	Integer

The key bit length parameter contains the number of bits of key material to derive and place in the provided key token. The value must be 0 if the PASSTHRU rule array keyword was specified. Otherwise, it must be 64 - 2048.

initialization_vector_length

Direction	Type
Input	Integer

When IHKEYAES is passed, this parameter contains the length of the *initialization_vector* in bytes. For IHKEYAES, the value must be 16 bytes.

When IHKEYAES is not passed, this parameter must be zero.

initialization_vector

Direction	Type
Input	String

When IHKEYAES is passed, this parameter contains the 16-byte *initialization_vector* that will be used to decrypt the hybrid_ciphertext.

When *initialization_vector_length* is zero, this parameter is ignored.

hybrid_ciphertext_length

Direction	Type
Input	Integer

When IHKEYKYB or IHKEYAES are passed, this parameter contains the length of *hybrid_ciphertext* in bytes. For IHKEYAES, the value must be 32 bytes. For IHKEYKYB, the value must be 1568 bytes.

When neither IHKEYKYB nor IHKEYAES are passed, this parameter must be zero.

hybrid_ciphertext

Direction	Type
Input	String

When IHKEYKYB or IHKEYAES is passed, the *hybrid_ciphertext* parameter must contain an encrypted value that will be deciphered with the *hybrid_key_identifier* and used in the Hybrid QSA Key Exchange Scheme.

When *hybrid_ciphertext_length* is zero, this parameter is ignored.

reserved3_length

Direction	Type
Input/Output	Integer

The *reserved3_length* parameter must be zero.

reserved3

Direction	Type
Input/Output	String

This parameter is ignored.

reserved4_length

Direction	Type
Input/Output	Integer

The *reserved4_length* parameter must be zero.

reserved4

Direction	Type
Input/Output	String

This parameter is ignored.

reserved5_length

Direction	Type
Input/Output	Integer

The *reserved5_length* parameter must be zero.

reserved5

Direction	Type
Input/Output	String

This parameter is ignored.

output_KEK_key_identifier_length

Direction	Type
Input	Integer

The length of the *output_KEK_key_identifier* parameter in bytes.

The *output_KEK_key_identifier_length* must be zero when:

- The *output_key_identifier* will contain an internal token, or
- The PASSTHRU rule array keyword was specified.

When the *output_KEK_key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

output_KEK_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the output key identifier. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

When the *output_KEK_key_identifier_length* is zero, this parameter is ignored.

For CCA keys, this is a variable-length key token containing a DES or AES key-encrypting key.

- For DES keys, the key is of type IMPORTER or EXPORTER with the IMPORT/EXPORT key usage attribute enabled in the control vector.
- For AES keys, the key is of type IMPORTER or EXPORTER with the IMPORT/EXPORT bit set in key usage field 1 and the wrap class derivation bit set in key usage field 4.

For X9.143 (TR-31) keys, this is a variable-length key block containing a TDES or AES key-encrypting key: key usage K0 or K1, algorithm T or A, and mode of use D or E.

If the token or key block supplied was encrypted under the old master key, the token or key block is returned encrypted under the current master key.

Note: A CCA key token can be wrapped by a CCA key-encrypting key or a TR-31 key-encrypting key. A TR-31 key block can be wrapped by a CCA key-encrypting key or a TR-31 key-encrypting key.

output_key_identifier_length

Direction	Type
Input/Output	Integer

The length of the *output_key_identifier* parameter in bytes. The service checks the field to ensure it is at least equal to the size of the token to return. On return from this service, this field is updated with the exact length of the key token created. The maximum allowed value is 9992 bytes.

output_key_identifier

Direction	Type
Input/Output	String

On input, the *output_key_identifier* must contain a skeleton key token or key block header (DERIV01 or DERIV02) or a null token (PASSTHRU).

For X9.143 (TR-31) key block headers, the header may contain: key usage K0, K1, or D0 algorithm D, T, or A, and mode of use D, E, or B.

On output, the *output_key_identifier* will contain:

- An internal or an external key token containing the generated symmetric key material.
- "Z" data (in the clear) if the PASSTHRU rule array keyword was specified.

If this variable specifies an external DES key token then the output KEK key identifier must identify a DES key-encrypting key token. If this specifies an external key token other than a DES key token then the output KEK key identifier must identify an AES key-encrypting key token.

Restrictions

The NIST security strength requirements will be enforced, with respect to ECC Curve type (input) and derived key length.

Only the following key types will be generated, skeleton key tokens of any other type will fail.

- DES: (Legacy DES token)
  - CIPHER
  - CIPHERXI
  - CIPHERXL
  - CIPHERXO
  - DECIPHER
  - ENCIPHER
  - IMPORTER
  - EXPORTER
  - IMP-PKA
- AES
  - DATA (Legacy AES token)
  - CIPHER (Variable-length symmetric key-token)
  - IMPORTER (Variable-length symmetric key-token)
  - EXPORTER (Variable-length symmetric key-token)

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the valid key bit lengths and the minimum curve size required for each of the supported output key types.

Table 61. Valid key bit lengths and minimum curve size required for the supported output key types.

Output Key ID type	Valid Key Bit Lengths	Minimum Curve Required
DES	64	P160
	128	P160

Table 61. Valid key bit lengths and minimum curve size required for the supported output key types. (continued)

Output Key ID type	Valid Key Bit Lengths	Minimum Curve Required
AES	128	P256
	192	P384
	256	P512

If the output key-encrypting key identifier is a weaker key than the key being generated, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key or the key-encrypting key is a double-length key.

Concatenation strings used for each derivation service

Table 62 on page 191 describes the concatenation string used for derivation service DERIV01.

Note: All integers are in Big-Endian format.

Table 62. CSNDEDH concatenation string format for DERIV01

Offset (bytes)	Length (bytes)	Value	Comments
0	4	Initialized to X'00000001'	Counter (four-byte) unsigned integer.
4	*xx*	Z	A shared secret bit string or octet string.
Fields added when QSA-ECDH is chosen			
4 + *xx*	*tt*	T *plaintext* decrypted from the *hybrid_ciphertext* parameter.	*plaintext* used for the Hybrid QSA Key Exchange Scheme.
4 + *xx* + *tt*	1	Value: **X'03'** DES **X'04'** AES	Algorithm identifier.
5 + *xx* + *tt*	1	Passed *party_info_length* variable.	Party information length passed by caller, converted to a one-byte unsigned integer.
6 + *xx* + *tt*	*party_info_length*	String identified by *party_info* parameter.	Party information passed by the caller.
6 + *xx* + *tt* + *party_info_length*	2	Supplied public information length, *zz*.	Two-byte unsigned integer specifying length of supplied public information.

Table 62. CSNDEDH concatenation string format for DERIV01 (continued)

Offset (bytes)	Length (bytes)	Value	Comments
6 + *xx* + *tt* + *party_info_length*	2	Supplied public information length, *zz*.	Two-byte unsigned integer specifying length of supplied public information.
8 + *xx* + *tt* + *party_info_length*	*zz*	Supplied public information.	Token data extracted from the skeleton key token identified by the *output_key_identifier* parameter.

Table 63 on page 192 describes the concatenation string used for derivation service DERIV02.

Note: All integers are in Big-Endian format.

Table 63. CSNDEDH concatenation string format for DERIV02

Offset (bytes)	Length (bytes)	Value	Comments
0	*xx*	Z	A shared secret bit string or octet string.
Fields added when QSA-ECDH is chosen			
xx	*tt*	T *plaintext* decrypted from the *hybrid_ciphertext* parameter.	32 byte *plaintext* decrypted from the *hybrid_ciphertext* parameter; length not an explicit field in concatenation string.
xx + *tt*	4	Initialized to X'00000001'.	Counter (four-byte) unsigned integer.
4 + *xx* + *tt*	*yy*	String identified by *party_info* parameter.	Party information passed by the caller; length not an explicit field in concatenation string.

Creating a Hybrid Quantum Safe Algorithm (QSA) Key Exchange Scheme

With CCA release 8.0, it is possible to build a Hybrid Quantum Safe (QSA) Key Exchange Scheme using CCA. The CCA services available support a Hybrid QSA Key Exchange Scheme where no data is exposed outside of the Crypto Express Adapter that is used as input to the final key derivation.

However, the Hybrid QSA Key Exchange Scheme is not a complete protocol so:

- Authentication of the public keys used in the Hybrid QSA Key Exchange Scheme is the responsibility of the host.
  - CRYSTALS-Kyber keys do not participate in PKIs at this time. The 'kyb-cert-A' certificate for a CRYSTALS-Kyber public key identified below is in recognition that certificate formats will be needed for the authentication part of a protocol.
  - For the ECC public keys, the CCA internal PKI may be used for authentication if the trust anchor has been installed to the adapter.
- A full protocol should include a Key Check Value calculated over the shared-key created by one person (for example, Alice) so that another person (for example, Bob) can verify the creation of an agreed shared-key.

The Hybrid QSA Key Exchange Scheme involves two participants (for example, Alice and Bob) and involves two CCA services: PKA Encrypt (CSNDPKE) and ECC Diffie-Hellman (CSNDEDH).

Step 1: The first person (Alice) creates the keys:

```
Kyb-priv-A, Kyb-pub-A: CRYSTALS-Kyber(1024) key pair
EC-priv-A, EC-pub-A: ECC key pair for key agreement
Kyb-cert-A, EC-cert-A: authenticated forms of Kyb-pub-A and EC-pub-A
```

and then sends the Kyb-cert-A and EC-cert-A keys to the second person (Bob).

Step 2: The second person (Bob) receives and validates the Kyb-cert-A and EC-cert-A keys from Alice.

1. After validation, Bob creates these keys:

   ```
   AES-ciph-B: AES-CIPHER key in a CCA key token
   EC-priv-B, EC-pub-B: ECC key pair for key agreement
   EC-cert-B: authenticated form of EC-pub-B
   Kyb-pub-A CCA public key token, with public key pulled from Kyb-cert-A
   ```

   Notes:

   - AES-ciph-B should be as strong as the derived shared-key.
   - AES-ciph-B should allow encrypt and decrypt because it is used on the same node.
2. Bob creates the **shared-key** derivation input using the CSNDPKE service. Bob calls the CSNDPKE service with the RANDOM keyword, AES-ciph-B, Kyb-pub-A, AES encryp-tion IV. The CSNDPKE service:
   - Generates a random 32B value: rand-32.
   - AES-CBC encrypts rand-32 using key AES-ciph-B and the AES encryption IV, returning [AES-ciph-B(rand-32)] in *keyvalue*.
   - CRYSTALS-Kyber encrypts rand-32 with Kyb-pub-A returning [Kyb-pub-A(rand-32)] in the *PKA_enciphered_keyvalue* parameter.
3. Bob completes the **shared-key** derivation, using the CSNDEDH service. Bob calls the CSNDEDH service with a derivation keyword, desired key length, [AES-ciph-B(rand-32)], AES-ciph-B, AES encryption IV, EC-priv-B, EC-cert-A, output skeleton token. The CSNDEDH service:
   - Decrypts rand-32 using the key AES-ciph-B and the AES encryption IV.
   - Uses EC-priv-B and EC-cert-A with ECDH to generate the Z value.
   - Passes Z and rand-32 to the key derivation function indicated by the derivation keyword, rand-32 is the *salt* or OtherData. The **shared-key** of the requested length is derived.
   - Places the **shared-key** in the output skeleton token provided, encrypts the key value.
   - Returns the final CCA **shared-key** token.
4. Bob stores the **shared-key**.
5. Bob sends EC-cert-B, [Kyb-pub-A(rand-32)] to Alice.

Step 3: Alice receives and validates EC-cert-B, [Kyb-pub-A(rand-32)].

Alice completes the **shared-key** derivation, using the CSNDEDH service.

Alice calls the CSNDEDH service with a derivation keyword, desired key length, [Kyb-pub-A (rand-32)], Kyb-priv-A, EC-priv-A, EC-cert-B, output skeleton token. The CSNDEDH service:

- Decrypts rand-32 using Kyb-priv-A.
- Uses EC-priv-A and EC-cert-B with ECDH to generate the Z value.
- Passes Z and rand-32 to the key derivation function indicated by the derivation keyword, rand-32 is the *salt* or OtherData. The **shared-key** of the requested length is derived.
- Places the **shared-key** in the output skeleton token provided, encrypts the key value.
- Returns the final CCA **shared-key** token.

Alice stores the **shared-key** .

The **shared-key** is now established at both Alice and Bob.

The role of the CSNDEDH service in this scheme is to complete the shared-key derivation for Alice or Bob and return the shared-key in a CCA key token.

Change to the key derivation in the CSNDEDH service:

- For DERIV01 and DERIV02, the change is the same: NIST SP 800-56C Rev 2 has defined Z' = Z || T, where T is a hybrid addition. The decrypted *hybrid_ciphertext* is concatenated to the end of the normal Z in the CSNDEDH concatenation string.
- This is accomplished in one call to CSNDEDH as follows:
- Bob's call to CSNDEDH:

  Inputs:

  - derivation keyword,
  - desired key length,
  - [AES-ciph-B(rand-32)], : output from CSNDPKE, random 32 byte value encrypted by AES-ciph-B
  - AES-ciph-B : AES-cipher key CCA token for Bob
  - EC-priv-B, : Bob's private ECC key
  - EC-cert-A, : Alice's public key
  - output skeleton token

  Outputs:
  CCA **shared-key** token
- Alice's call to CSNDEDH:

  Inputs:

  - derivation keyword,
  - desired key length,
  - [Kyb-pub-A(rand-32)], : output from CSNDPKE, random 32 byte value encrypted by Kyb-pub-A
  - Kyb-priv-A : CRYSTALS-Kyber private key CCA token for Alice
  - EC-priv-A, : Alice's private ECC key
  - EC-cert-B, : Bob's public key
  - output skeleton token

  Outputs:
  CCA **shared-key** token

Access control points

The ECC Diffie-Hellman callable service requires the **ECC Diffie-Hellman** access control point to be enabled in the domain role.

Specifying the PASSTHRU rule array keyword requires that the **ECC Diffie-Hellman - Allow PASSTHRU** access control point be enabled in the domain role.

Specifying the DERIV02 rule array keyword requires that the **ECC Diffie-Hellman - Allow DERIV02** access control point be enabled in the domain role.

Specifying the QSA-ECDH rule array keyword requires that the **ECC Diffie-Hellman - Allow Hybrid QSA Scheme** access control point be enabled in the domain role.

If the *output_key_identifier* parameter references a DES key token and the key wrapping method keyword specifies a wrapping method that is not the default method, then the **ECC Diffie-Hellman - Allow key wrap override** access control point must be enabled in the domain role.

Each Elliptic Curve type supported has its own access control point. The access control point must be enabled to use the curve type and strength.

- ECC Diffie-Hellman - Allow Prime Curve 192
- ECC Diffie-Hellman - Allow Prime Curve 224
- ECC Diffie-Hellman - Allow Prime Curve 256
- ECC Diffie-Hellman - Allow Prime Curve 384
- ECC Diffie-Hellman - Allow Prime Curve 521
- ECC Diffie-Hellman - Allow BP Curve 160
- ECC Diffie-Hellman - Allow BP Curve 192
- ECC Diffie-Hellman - Allow BP Curve 224
- ECC Diffie-Hellman - Allow BP Curve 256
- ECC Diffie-Hellman - Allow BP Curve 320
- ECC Diffie-Hellman - Allow BP Curve 384
- ECC Diffie-Hellman - Allow BP Curve 512
- ECC Diffie-Hellman - Allow Koblitz curve 256

To prevent a weaker key from being used to generate a stronger key, enable the **ECC Diffie-Hellman – Prohibit weak key generate** access control point in the domain role.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 64. ECC Diffie-Hellman required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	The DERIV02, SHA-224, SHA-256, SHA-384, and SHA-512 keywords requires the March 2016 or later licensed internal code (LIC). Triple-length DES keys require the July 2019 or later licensed internal code (LIC). ECC Koblitz curve secp256k1 is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). CRYSTALS-Kyber private keys and rules QSA-ECDH and ECDH are not supported. X9.143 key blocks are not supported.

Table 64. ECC Diffie-Hellman required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	The DERIV02, SHA-224, SHA-256, SHA-384, and SHA-512 keywords requires the March 2016 or later licensed internal code (LIC). Triple-length DES keys require the December 2018 or later licensed internal code (LIC). ECC Koblitz curve secp256k1 is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). CRYSTALS-Kyber private keys and rules QSA-ECDH and ECDH are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). ECC Koblitz curve secp256k1 is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). CRYSTALS-Kyber private keys and rules QSA-ECDH and ECDH are not supported. X9.143 key blocks are not supported.

Table 64. ECC Diffie-Hellman required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	ECC Koblitz curve secp256k1 is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). CRYSTALS-Kyber private keys and rules QSA-ECDH and ECDH are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). CRYSTALS-Kyber private keys and rules QSA-ECDH and ECDH are not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	ECC Koblitz curve secp256k1 requires the September 2020 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). CRYSTALS-Kyber private keys and rules QSA-ECDH and ECDH are not supported. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	CRYSTALS-Kyber private keys and rules QSA-ECDH and ECDH are not supported. X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	CRYSTALS-Kyber private keys and rules QSA-ECDH and ECDH require CCA release 8.0 or later licensed internal code (LIC). X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Generate Issuer MK (CSNBGIM and CSNEGIM)

The Generate Issuer MK callable service helps with the initial steps of EMV setup by generating and storing the issuer master keys. Optionally, the issuer master keys can be returned as external tokens wrapped under a key-encrypting key (KEK) that is shared with the ICC personalization system. Use the TKE workstation to establish the KEK that is optionally used by this service.

Use the Generate Issuer MK service to generate the issuer master keys for use in EMV processing. The master keys are double-length DES key of key type DKYGENKY. The subtype and key usage attributes depends on the type of master key and key mode.

The generated key is stored in the CKDS using the label supplied and returned to the caller in an internal token. The label must not exist in the CKDS. Optionally, the key can be returned as external tokens wrapped under a key-encrypting key.

The following master keys can be generated:

Issuer Master Application Cryptogram Key (AC)
For VISA and EMV, this key is used for deriving session keys for ARQC verification and ARPC generation.

For MasterCard, two keys are generated:

- The issuer master key used for deriving session keys for ARQC verification.
- The ARPC master key used for deriving session keys for ARPC generation.

Issuer Master Secure Messaging Authentication Key (MAC)
This key is used for deriving session keys to provide integrity for EMV scripting.

Issuer Master Secure Messaging Confidentiality Key (ENC)
This key is used for deriving session keys to provide confidentiality for EMV scripting.

Issuer Master Data Key (DATA)
This key is used for deriving session keys for functions that require encryption and decryption of EMV fields for EMV and MasterCard.

The callable service name for AMODE(64) invocation is CSNEGIM.

Format

```
CALL CSNBGIM(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             issuer_master_key_identifier_length,
             issuer_master_key_identifier,
             issuer_ARPC_master_key_identifier_length,
             issuer_ARPC_master_key_identifier,
             transport_key_identifier_length,
             transport_key_identifier,
             reserved1_length,
             reserved1,
             reserved2_length,
             reserved2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 3 and the maximum value is 6.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 65. Rule array keywords for Generate Issuer MK

Keyword	Meaning
Algorithm (Required)	
TDES	Specifies the use of Triple-DES.
Key mode (One required)	
VISA	Use this key mode for Visa Cryptogram Version 10 or Cryptogram Version 18.
MC	Use this key mode for MasterCard M/CHIP 2.1.
EMV	Use this key mode for EMV book 2, Annex A1.3, Visa Cryptogram Version 14, and MasterCard M/CHIP 4.
Output key type (One required). See the *issuer_master_key_identifier* and *issuer_ARPC_master_key_identifier* parameters for the attributes of the generated keys.	
AC	Generates the Issuer Master Application Cryptogram Key. This key is used for deriving session keys for ARQC verification and ARPC generation.
MAC	Generates the Issuer Master Secure Messaging Authentication Key. This key is used to provide integrity for EMV scripting. Not valid with cryptogram version CVN18.

Table 65. Rule array keywords for Generate Issuer MK (continued)	
Keyword	**Meaning**
ENC	Generates the Issuer Master Secure Messaging Confidentiality Key. This key is used to provide confidentiality for EMV scripting. Not valid with cryptogram version CVN18.
DATA	Generates the Issuer Master Data Key. This key is used for functions that require encryption and decryption of EMV fields. Not valid with the VISA key mode.
Key encryption (Optional)	
MASTER	Specifies to return the ICC Master Key as an internal token encrypted under the master key. This is the default.
XPORT	Specifies to return the issuer master key or issuer master keys as external tokens wrapped under the *transport_key_identifier*.
Compliance (Optional)	
COMP-TAG	Generate a compliant-tagged key.
NOCMPTAG	Do not generate a compliant-tagged key. This is the default.
Cryptogram version number (Optional. Only valid with key mode VISA.)	
CVN10	Specifies the Visa Cryptogram Version 10 processing method. This is the default for key mode VISA.
CVN18	Specifies the Visa Cryptogram Version 18 processing method. This is only allowed for output key type AC.

issuer_master_key_identifier_length

Direction	**Type**
Input	Integer

Specifies the length of the *issuer_master_key_identifier* parameter in bytes. The value must be 64.

issuer_master_key_identifier

Direction	**Type**
Input/Output	String

The key identifier of the master key being generated.

On input, this is a 64-character label of the record to be added to the CKDS. The supplied CKDS label must not already exist. This service will not overwrite existing CKDS records.

On output, the generated key is stored in the CKDS and is returned in a key token in this parameter. When the XPORT keyword is specified, the token is returned in external format wrapped under the key-encrypting key supplied in the *transport_key_identifier* parameter. Otherwise, it is returned in internal format.

All master keys generated are double-length DES keys of key type DKYGENKY. The subtype and key usage attributes for the master keys are listed in Table 66 on page 201.

Table 66. Generate Issuer MK: Attributes of the generated key				
Master key	**VISA CVN10**	**VISA CVN18**	**MC**	**EMV**
Application Cryptogram Key (AC)	DMAC, DKYL0	DMAC, DKYL1	DMAC, DKYL1	DMAC, DKYL0
Secure Messaging Authentication Key (MAC)	DMAC, DKYL0	N/A	DMAC, DKYL1	DMAC, DKYL0
Secure Messaging Confidentiality Key (ENC)	DMPIN, DKYL0	N/A	DMPIN, DKYL1	DMPIN, DKYL0
Data Key (DATA)	N/A	N/A	DDATA, DKYL1	DDATA, DKYL0

issuer_ARPC_master_key_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *issuer_ARPC_master_key_identifier* parameter in bytes. When the key mode keyword is MC and the output key type keyword is AC, the value must be 64. Otherwise, the value must be zero.

issuer_ARPC_master_key_identifier

Direction	Type
Input/Output	String

The key identifier of the master key being generated. If the *issuer_ARPC_master_key_identifier* parameter is zero, this parameter is ignored.

On input, this is a 64-character label of the record to be added to the CKDS. The supplied CKDS label must not already exist. This service will not overwrite existing CKDS records.

On output, the generated key is stored in the CKDS and is returned in a key token in this parameter. When the XPORT keyword is specified, the token is returned in external format wrapped under the key-encrypting key supplied in the *transport_key_identifier* parameter. Otherwise, it is returned in internal format.

The attributes of the generated key:

- Key algorithm is DES.
- Key type is DKYGENKY.
- Subtype is DKYL0.
- Key usage is DMAC.

transport_key_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *transport_key_identifier* parameter in bytes. When the XPORT keyword is specified, the value must be 64. Otherwise, the value must be 0.

transport_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the generated keys. The key identifier is an operational token or the key label of an operational token in key storage. When the *transport_key_identifier_length* is zero, this parameter is ignored.

The key identifier must be a DES IMPORTER or EXPORTER. If an EXPORTER key is supplied, it may be a NOCV EXPORTER. If an IMPORTER is supplied and the MasterCard AC master keys are being generated, it cannot be an NOCV IMPORTER. A NOCV IMPORTER may be used for the generation of other master keys.

If the NOCV bit is on in the internal key token containing the transport key, the transport key (not the transport key variant) is used to encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter; or you are passing the transport key in the internal key token with the NOCV bit on and your program is running in supervisor state or key 0-7.

The NOCV bit is shown in Table 608 on page 1504.

If the COMP-TAG keyword is specified, the transport key must be compliant-tagged.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Cryptographic services used by Generate Issuer MK

The following CCA cryptographic services are used by Generate Issuer MK:

- CSNBKEX - Key Export
- CSNBKGN - Key Generate
- CSNBKIM - Key Import
- CSNBKRC2 - CKDS Key Record Create2
- CSNBKTB - Key Token Build

The caller does not require authorization to each of these services, only to Generate Issuer MK. Additionally, the caller must have the required access control points enabled.

Access control points

The following access control points must be enabled to use Generate Issuer MK:

- Key Export
- Key Generate - OP
- Key Import

To use a NOCV key-encrypting key EXPORTER, the **NOCV KEK usage for export-related functions** access control must be enabled in addition to the other access control points listed.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 67. Generate Issuer MK required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords COMP-TAG and NOCMPTAG are not supported. Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Keywords COMP-TAG and NOCMPTAG are not supported. Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).

Table 67. Generate Issuer MK required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Key Encryption Translate (CSNBKET and CSNEKET)

Use the Key Encryption Translate service to change the method of encryption of DES key material. The input key can be a double-length external DES CCA DATA key token, or a double-length CBC-encrypted key. The return key is encrypted using CBC encryption or CCA (ECB) encryption. The CCA DATA key must be double-length and have an all-zero control vector. The CBC-encrypted key is treated as a 16-byte or 24-byte string encrypted with an all-zero initialization vector.

Note: This service does not handle keys wrapped using the enhanced method.

The callable service name for AMODE(64) invocation is CSNEKET.

Format

```
CALL CSNBKET(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             kek_key_identifier_length,
             kek_key_identifier,
             key_in_length,
             key_in,
             key_out_length,
             key_out)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

The *rule_array* keywords are:

Table 68. Keywords for Key Encryption Translate

Keyword	Meaning
Key translation method (one required)	
CBCTOECB	Specifies decryption of a 16-byte or 24-byte string and CCA encryption of the resulting clear-key value as an external CCA DATA key.
ECBTOCBC	Specifies decryption of a CCA DATA key and the CBC encryption of the resulting clear key.

kek_key_identifier_length

Direction	Type
Input	Integer

The length of the *kek_key_identifier* parameter in bytes. The value must be 64.

kek_key_identifier

Direction	Type
Input/Output	String

The key-encrypting key used to decrypt the input key and to encrypt the output key. This is an internal token or the 64-byte label of a key in the CKDS. For the CBCTOECB keyword, the key must be a DES IMPORTER key. For the ECBTOCBC keyword, the key must be a DES EXPORTER key.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

key_in_length

Direction	Type
Input	Integer

The length of the *key_in* parameter in bytes. The value must be 16 or 24 for a **CBCTOECB** translation or 64 for an **ECBTOCBC** translation.

key_in

Direction	Type
Input	String

The key material or key token to be translated. This is either a CCA external key token or a 16-byte or 24-byte CBC-encrypted key.

key_out_length

Direction	Type
Input/Output	Integer

The length of the *key_out* parameter in bytes. On input, set value must be:

- At least 24 for the ECBTOCBC translation.
- At least 64 for the CBCTOECB translation.

Upon successful completion, the parameter is set to the length of the value returned in the *key_out* parameter.

key_out

Direction	Type
Output	String

The translated key material or key token.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The following access control points in the domain role control the function of this service:

Table 69. Required access control points for Key Encryption Translate	
Rule array keyword	**Access control point**
CBCTOECB	Key Encryption Translate – CBC to ECB
ECBTOCBC	Key Encryption Translate – ECB to CBC

Note: These access controls are not enabled in the domain role until the March 2016 or later licensed internal code (LIC). A TKE workstation is required to enable them.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 70. Key Encryption Translate required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC).
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Key Export (CSNBKEX and CSNEKEX)

Use the key export callable service to export a DES operational fixed-length key token, wrapped under the DES master, into an external fixed-length DES key token wrapped by a DES key-encrypting key. The external key can be exported to another system.

The callable service name for AMODE(64) invocation is CSNEKEX.

Format

```
CALL CSNBKEX(
             return_code,
             reason_code,
             exit_data_length,
```

```
                exit_data,
                key_type,
                source_key_identifier,
                exporter_key_identifier,
                target_key_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_type

Direction	Type
Input	Character Integer

The parameter is an 8-byte field that contains either a key type value or the keyword TOKEN. The keyword is left-justified and padded on the right with blanks.

If the key type is TOKEN, ICSF determines the key type from the control vector (CV) field in the internal key token provided in the *source_key_identifier* parameter.

Key type values for the Key Export callable service are: CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, DATA, DATAC, DATAM, DATAMV, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER.

source_key_identifier

Direction	Type
Input	String

A 64-byte string of the internal key token that contains the key to be reenciphered. This parameter must identify an internal key token in application storage, or a label of an existing key in the cryptographic key data set.

If you supply TOKEN for the *key_type* parameter, ICSF looks at the control vector in the internal key token and determines the key type from this information. If you supply TOKEN for the *key_type* parameter and supply a label for this parameter, the label must be unique in the cryptographic key data set.

exporter_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to wrap the source key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA key tokens, the identifier is a 64-byte DES key token of key type EXPORTER. When the NOCV bit is on in the internal key token containing the transport key, the transport key (not the transport key variant) is used to encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter; or you are passing the transport key in the internal key token with the NOCV bit on and your program is running in supervisor state or key 0-7.

For X9.143 keys, the identifier is a variable-length key block of a TDES key-encrypting key usage K0, algorithm T, and mode of use E.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

target_key_identifier

Direction	Type
Output	String

The 64-byte field for the external key token that contains the reenciphered key. The reenciphered key can be exchanged with another cryptographic system.

The output *target_key_identifier* will be wrapped in the same manner as the *source_key_identifier*.

The DES key wrapping methods available are described in “CCA key wrapping” on page 20.

Restrictions

Current applications will fail if they use an equal key halves exporter to export a key with unequal key halves. You must have access control point 'Key Export - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

For key export, you can use these combinations of parameters:

- A valid key type in the *key_type* parameter and an internal key token in the *source_key_identifier* parameter. The key type must be equivalent to the control vector specified in the internal key token.
- A *key_type* parameter of TOKEN and an internal key token in the *source_key_identifier* parameter. The *source_key_identifier* can be a label with TOKEN only if the labelname is unique on the CKDS. The key type is extracted from the control vector contained in the internal key token.
- A valid key type in the *key_type* parameter, and a label in the *source_key_identifier* parameter.

If internal key tokens are supplied in the *source_key_identifier* or *exporter_key_identifier* parameters, the key in one or both tokens can be reenciphered. This occurs if the master key was changed since the internal key token was last used. The return and reason codes that indicate this do *not* indicate which key was reenciphered. Therefore, assume both keys have been reenciphered.

For key types CIPHERXI, CIPHERXL, and CIPHERXO, the key-encrypting key in the exporter_key_identifier parameter must have a control vector with the key halves guaranteed unique flag on in the key form bits. An existing key-encrypting key can have its control vector updated using the restrict key attribute callable service.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified] and a token is passed as input to be exported, an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the exported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 71. Required access control points for Key Export

Access Control Point	Restrictions
Key Export	None
Key Export - Unrestricted	Key-encrypting key may have equal key halves

To use a NOCV key-encrypting key with the key export service, the **NOCV KEK usage for export-related functions** access control point must be enabled in addition to one or both of the access control points listed.

If the output key-encrypting key identifier is a weaker key than the key being exported, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys access** control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the key-encrypting key is a double-length key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 72. Key export required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in output DES CIPHER key tokens requires a CEX6C with the P41458.002 or later MCL. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The CPACF export bit (XPRTCPAC) in the output key token is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Key Generate (CSNBKGN and CSNEKGN)

Use the key generate callable service to generate either one or two odd parity fixed-length DES keys in CCA key tokens of any type. The keys can be single-length (8 bytes), double-length (16 bytes), or triple-length (24 bytes). The callable service does not produce keys in clear form and all keys are returned

in encrypted form. When two keys are generated, each key has the same clear value, although this clear value is not exposed outside the secure cryptographic feature.

Use the key generate callable service to generate a fixed-length CCA AES key of DATA type. The callable service does not produce AES keys in clear form and all AES keys are returned in encrypted form. Only one AES key is generated.

The callable service name for AMODE (64) invocation is CSNEKGN.

Format

```
CALL CSNBKGN(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_form,
             key_length,
             key_type_1,
             key_type_2,
             KEK_key_identifier_1,
             KEK_key_identifier_2,
             generated_key_identifier_1,
             generated_key_identifier_2 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_form

Direction	Type
Input	Character String

A 4-byte keyword that defines the type of key or keys you want to generate. This parameter also specifies if each key should be returned for either operational, importable, or exportable use. The keyword must be in a 4-byte field, left-justified, and padded with blanks. See Table 73 on page 213 for the values and their meanings.

The first two characters refer to *key_type_1*. The next two characters refer to *key_type_2*.

If the *key_form* is OP, EX or IM, the KEK_*key_identifier_2*, *key_type_2*, and *generated_key_identifier_2* should be set to NULL.

Table 73. Key Form values for the Key Generate callable service

Keyword	Meaning
EX	One key that can be sent to another system.
EXEX	A key pair; both keys to be sent elsewhere, possibly for exporting to two different systems. The key pair has the same clear value.
IM	One key that can be locally imported. The key can be imported onto this system to make it operational at another time.
IMEX	A key pair to be imported; one key to be imported locally and one key to be sent elsewhere. Both keys have the same clear value.
IMIM	A key pair to be imported; both keys to be imported locally at another time.
OP	One operational key. The key is returned to the caller in the key token format. Specify the OP key form when generating AES keys.
OPEX	A key pair; one key that is operational and one key to be sent from this system. Both keys have the same clear value.
OPIM	A key pair; one key that is operational and one key to be imported to the local system. Both keys have the same clear value. On the other system, the external key token can be imported to make it operational.
OPOP	A key pair; normally with different control vector values.

The key forms are defined as follows:

Operational (OP)
The key value is enciphered under a master key. The result is placed into an internal key token. The key is then operational at the local system.

Importable (IM)
The key value is enciphered under an importer key-encrypting key. The result is placed into an external key token.

Exportable (EX)
The key value is enciphered under an exporter key-encrypting key. The result is placed into an external key token. The key can then be transported or exported to another system and imported there for use. This key form cannot be used by any ICSF callable service.

The keys are placed into tokens that the *generated_key_identifier_1* and *generated_key_identifier_2* parameters identify.

Valid key type combinations depend on the key form. See Table 79 on page 221 for valid key combinations.

key_length

Direction	Type
Input	Character String

An 8-byte value that defines the length of the key. The keyword must be left-justified and padded on the right with blanks. You must supply one of the key length values in the *key_length* parameter.

Table 74. Key Length values for the Key Generate callable service

Value	Description	Algorithm
SINGLE or KEYLN8	Single-length (8-byte) key.	DES
SINGLE-R	Double length (16-byte) key. The two key halves will be the same. This makes the key effectively a single length key.	DES
DOUBLE or KEYLN16	Double-length (16-byte) key. The two key halves are not guaranteed to be unique.	DES
DOUBLE-O	Double-length (16-byte) key. Each of the two key halves will be unique (not the same value).	DES
KEYLN24	DATA key type only. The control vector in the output token will be all zeros.	DES
TRIPLE	Triple-length (24-byte) key. The key values are not guaranteed to be unique.	DES
TRIPLE-O	Triple-length (24-byte) key. The key values will be unique.	DES
KEYLN16	128-bit key.	AES
KEYLN24	192-bit key.	AES
KEYLN32	256-bit key.	AES

DES Keys: Double-length (16-byte) keys have an 8-byte left half and an 8-byte right half. Both halves can have identical clear values or not. If you want the same value to be used in both key halves (refered to as replicated key values), specify *key_length* as SINGLE, SINGLE-R or KEYLN8. If you want different values to be the basis of each key half, specify *key_length* as DOUBLE, DOUBLE-O or KEYLN16.

Triple-length (24-byte) keys have three 8-byte key parts. To generate a triple-length DATA key with three different values to be the basis of each key part, specify *key_length* as TRIPLE or TRIPLE-O.

For the DATA key type,

- Keyword KEYLN24 generates a triple-length key in a version 1 key token with a zero control vector.
- Keywords TRIPLE and TRIPLE-O generate a triple-length key in a version 0 key token with a DATA control vector.

Use SINGLE/SINGLE-R if you want to create a DES transport key that you would use to exchange DATA keys with a PCF system.

AES Keys: AES only allows KEYLN16, KEYLN24, KEYLN32. To generate a 128-bit AES key, specify key_length as KEYLN16. For 192-bit AES keys specify key_length as KEYLN24. A 256-bit AES key requires a key_length of KEYLN32. All AES keys are DATA keys.

This table shows the valid key lengths for each key type supported by DES keys. An **X** indicates that a key length is permitted for a key type. A **D** indicates the default key length.

Table 75. Key lengths for DES keys

Key Type	Single - KEYLN8	Single-R	Double - KEYLN16	DOUBLE-O	KEYLN24	TRIPLE	TRIPLE-O
CIPHERXI CIPHERXL CIPHERXO				X			
CVARDEC* CVARENC* CVARPINE* CVARXCVL* CVARXCVR*	X						
DATA	D		X		X	X	X
DATAC* DATAM DATAMV		X	X				
DKYGENKY* KEYGENKY*		X	D				
EXPORTER IMPORTER		X	D	X		X	X
IKEYXLAT OKEYXLAT		X	D	X			
IPINENC OPINENC PINGEN PINVER		X	D	X		X	X
MAC MACVER CIPHER DECIPHER ENCIPHER	D	X	X	X		X	X

This table shows the valid key lengths for each key type supported by AES keys. An X indicates that a key length is permitted for that key type.

Table 76. Key lengths for AES keys			
Key Type	**KEYLEN16**	**KEYLEN24**	**KEYLEN32**
AESTOKEN **AESDATA**	X X	X X	X X

key_type_1

Direction	Type
Input	Character String

Use the *key_type_1* parameter for the first, or only key, that you want generated. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form.

The 8-byte keyword for the *key_type_1* parameter can be one of the following:

- AESDATA, AESTOKEN, CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, DATA, DATAC, DATAM, DATAMV, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER
- or the keyword TOKEN

If *key_type_1* is TOKEN, ICSF examines the control vector (CV) field in the *generated_key_identifier_1* parameter to derive the key type. When *key_type_1* is TOKEN, ICSF does not check for the length of the key for DATA keys. Instead, ICSF uses the *key_length* parameter to determine the length of the key.

If *key_type_1* is AESDATA or AESTOKEN, the key generated will be an AES key of type DATA. When *key_type_1* is AESTOKEN, ICSF uses the key_length parameter to determine the length of the key.

See Table 78 on page 220 and Table 79 on page 221 for valid key type and key form combinations.

key_type_2

Direction	Type
Input	Character String

Use the *key_type_2* parameter for a key pair, which is shown in Table 79 on page 221. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form. *key_type_2* is only used when DES keys are generated.

The 8-byte keyword for the key_type_2 parameter can be one of the following:

- CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, DATA, DATAC, DATAM, DATAMV, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER
- or the keyword TOKEN

If *key_type_2* is TOKEN, ICSF examines the control vector (CV) field in the *generated_key_identifier_2* parameter to derive the key type. When *key_type_2* is TOKEN, ICSF does not check for the length of the key for DATA keys. Instead, ICSF uses the *key_length* parameter to determine the length of the key.

If only one key is to be generated, *key_type_2* and *KEK_key_identifier_2* are ignored.

See Table 78 on page 220 and Table 79 on page 221 for valid key type and key form combinations.

KEK_key_identifier_1

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to wrap the generated key 1 when the key form indicates an external key is generated. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage. When an operational key is being generated for *generated_key_1* (OP), the parameter is ignored. In this case, it is recommended that the parameters are initialized to 64-bytes of X'00'.

For key labels, this is a 64-byte string, left-justified and padded with blanks.

For CCA key tokens, this is a 64-byte key token of a DES key-encrypting key.

For X9.143 (TR-31) key blocks, this is a variable-length key block of a DES key-encrypting key.

Table 77 on page 217 shows the attributes of the key-encrypting keys for the key forms. For CCA key tokens, the attributes must be enabled in the control vector.

Table 77. Key attributes for key-encrypting keys for CSNBKGN

Key form keyword	***KEK_key_identifier_1***		***KEK_key_identifier_2***	
	CCA key	**X9.143 (TR-31) key**	**CCA key**	**X9.143 (TR-31) key**
OP, OPOP	64-byte null key token		64-byte null key token	
EX	DES EXPORTER with attributes EXEX and EXPORT	Key usage K0 Algorithm T Mode of use E	64-byte null key token	
IM	DES IMPORTER with attributes IMEX and IMPORT	Key usage K0 Algorithm T Mode of use D	64-byte null key token	
OPEX	64-byte null key token		DES EXPORTER with attributes OPEX and EXPORT	Key usage K0 Algorithm T Mode of use E
OPIM	64-byte null key token		DES IMPORTER with attributes OPIM and IMPORT	Key usage K0 Algorithm T Mode of use D
EXEX	DES EXPORTER with attributes EXEX and EXPORT	Key usage K0 Algorithm T Mode of use E	DES EXPORTER with attributes EXEX and EXPORT	Key usage K0 Algorithm T Mode of use E
IMEX	DES IMPORTER with attributes IMEX and IMPORT	Key usage K0 Algorithm T Mode of use D	DES IMPORTER with attributes IMEX and EXPORT	Key usage K0 Algorithm T Mode of use E
IMIM	DES IMPORTER with attributes IMIM and IMPORT	Key usage K0 Algorithm T Mode of use D	DES IMPORTER with attributes IMIM and IMPORT	Key usage K0 Algorithm T Mode of use D

For CCA keys, when the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter; or you are passing the key-encrypting key in the internal key token with the NOCV bit on and your program is running in supervisor state or key 0-7.

The NOCV bit is shown in Table 608 on page 1504.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

KEK_key_identifier_2

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to wrap the generated key 2 when the key form indicates an external key is generated. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage. When an operational key is being generated for generated_key_2 (OP), the parameter is ignored. In this case, it is recommended that the parameters are initialized to 64-bytes of X'00'.

For key labels, this is a 64-byte string, left-justified and padded with blanks.

For CCA key tokens, this is a 64-byte key token of a DES key-encrypting key.

For X9.143 (TR-31) key blocks, this is a variable-length key block of a DES key-encrypting key.

See Table 77 on page 217 for the attributes of the key-encrypting keys for the key forms.

For CCA keys, when the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter; or you are passing the key-encrypting key in the internal key token with the NOCV bit on and your program is running in supervisor state or in key 0-7.

The NOCV bit is shown in Table 608 on page 1504.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

generated_key_identifier_1

Direction	Type
Input/Output	String

On input, this parameter contains a 64-byte key token.

- When the *key_type_1* parameter is TOKEN, a key token that ICSF will use to determine the key type and key length:
  - A skeleton token of the key type and length to be generated.
  - An internal or external token of the key type and length to be generated.
- Otherwise, a null token.

On output, this parameter receives the generated key token:

- Internal DES or AES key token for an operational key form, or

- External DES key tokens containing a key enciphered under the *KEK_key_identifier_1* parameter.

To mark an operational IMPORTER or EXPORTER key as a NOCV KEK, the following is required:

- The *key_type_1* parameter must be IMPORTER or EXPORTER.
- The key form is OPEX.
- *generated_key_identifier_1* parameter must contain a valid DES internal token of the same type with the NOCV KEK flag enabled.

Or

- The *key_type_1* parameter must be TOKEN.
- The key form is OPEX.
- *generated_key_identifier_1* parameter must contain a skeleton internal token of an IMPORTER or EXPORTER with the NOCV KEK flag enabled. Any wrapping method can be specified.

To generate a compliant-tagged DES key token, a compliant-tagged skeleton token must be supplied.

The output *generated_key_identifier_1* will use the default wrapping method unless a skeleton token is supplied as input. If a skeleton token is supplied as input, the wrapping method in the skeleton token will be used.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

When the WRAPENH3 method is selected, a skeleton key token is required. A secure internal key token wrapped with the WRAPENH3 method obfuscates the key length.

When *key_type_1* parameter is AESDATA, the *generated_key_identifier_1* is ignored. In this case, it is recommended that the parameter be initialized to 64-bytes of X'00'. If you specify a *key_type_1* of AESTOKEN, the *generated_key_identifier_1* parameter must be an internal AES key token or a clear AES key token. Information in this token can be used to determine the key type:

- The *key_type_1* parameter overrides the type in the token.
- The *key_length* parameter overrides the length value in the generated key token.

generated_key_identifier_2

Direction	Type
Input/Output	String

On input, this parameter contains a 64-byte key token.

- When the *key_type_2* parameter is TOKEN, a key token that ICSF will use to determine the key type and key length:
  - A skeleton token of the key type and length to be generated.
  - An internal or external token of the key type and length to be generated.
- Otherwise, a null token.

On output, this parameter receives the generated key token:

- Internal DES key token for an operational key form, or
- External DES key tokens containing a key enciphered under the *KEK_key_identifier_2* parameter.

To generate a compliant-tagged DES key token, a compliant-tagged skeleton token must be supplied.

The output *generated_key_identifier_2* will use the default wrapping method unless a skeleton token is supplied as input. If a skeleton token is supplied as input, the wrapping method in the skeleton token will be used.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

When the WRAPENH3 method is selected, a skeleton key token is required. A secure internal key token wrapped with the WRAPENH3 method obfuscates the key length.

Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This service cannot be used to generate SECMSG keys. SECMSG keys must be derived using CSNBDKG.

For key types CIPHERXI, CIPHERXL, and CIPHERXO, the key-encrypting keys in the *KEK_key_identifier_1* and *KEK_key_identifier_2* parameters must have a control vector with the key halves guaranteed unique flag on in the key form bits. An existing key-encrypting key can have its control vector updated using the restrict key attribute callable service.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the generated key.

Usage notes - Key type and key form combinations

Table 78 on page 220 shows the valid key type and key form combinations for a single DES or AES key. Key types marked with an "*" must be requested through the specification of a proper control vector in a key token and through the use of the TOKEN keyword.

Note: Not all keytypes are valid on all hardware.

Table 78. Key Generate Valid Key Types and Key Forms for a Single Key

Key Type 1	Key Type 2	OP	IM	EX
AESDATA	Not applicable	X		
AESTOKEN	Not applicable	X		
CIPHER	Not applicable	X	X	X
DATA	Not applicable	X	X	X
DATAC*	Not applicable	X	X	X
DATAM	Not applicable	X	X	X
DKYGENKY*	Not applicable	X	X	X
KEYGENKY*	Not applicable	X	X	X
MAC	Not applicable	X	X	X
PINGEN	Not applicable	X	X	X

Table 79 on page 221 shows the valid key type and key form combinations for a DES key pair. Key types marked with an "*" must be requested through the specification of a proper control vector in a key token and through the use of the TOKEN keyword.

See Table 80 on page 222 for an explanation of the differences between **E** as compared to **X**.

Table 79. Key Generate Valid Key Types and Key Forms for a Key Pair

Key Type 1	Key Type 2	OPEX	EXEX	OPIM, OPOP, IMIM	IMEX
CIPHER	CIPHER CIPHERXI CIPHERXL CIPHERXO DECIPHER ENCIPHER	X	X	X	X
CIPHERXI	CIPHER ENCIPHER	E	X	X	E
CIPHERXI	CIPHERXO	E	X		E
CIPHERXL	CIPHER	E	X	X	E
CIPHERXL	CIPHERXL	E	X		E
CIPHERXO	CIPHER DECIPHER	E	X	X	E
CIPHERXO	CIPHERXI	E	X		E
CVARDEC*	CVARENC* CVARPINE*	E			E
CVARENC*	CVARDEC* CVARXCVL* CVARXCVR*	E			E
CVARXCVL*	CVARENC*	E			E
CVARXCVR*	CVARENC*	E			E
CVARPINE*	CVARDEC*	E			E
DATA	DATA	X	X	X	X
DATAC*	DATAC*	X	X	X	X
DATAM	DATAM DATAMV	X	X	X	X
DECIPHER	CIPHER CIPHERXO ENCIPHER	X	X	X	X
DKYGENKY*	DKYGENKY*	X	X	X	X
ENCIPHER	CIPHER CIPHERXI DECIPHER	X	X	X	X

Table 79. Key Generate Valid Key Types and Key Forms for a Key Pair (continued)					
Key Type 1	**Key Type 2**	**OPEX**	**EXEX**	**OPIM, OPOP, IMIM**	**IMEX**
EXPORTER	IKEYXLAT IMPORTER	X	X		X
IKEYXLAT	EXPORTER OKEYXLAT	X	X		X
IMPORTER	EXPORTER OKEYXLAT	X	X		X
IPINENC	OPINENC	X	X	E	X
KEYGENKY*	KEYGENKY*	X	X	X	X
MAC	MAC MACVER	X	X	X	X
OKEYXLAT	IKEYXLAT IMPORTER	X	X		X
OPINENC	IPINENC	X	X	E	X
OPINENC	OPINENC			X	
PINVER	PINGEN	X	X		X
PINGEN	PINVER	X	X		X

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 80. Required access control points for Key Generate	
Usage	**Access Control Point**
The key-form and key-type combinations shown with an 'X' in the Key_Form OP column in Table 78 on page 220.	Key Generate – OP
The key-form and key-type combinations shown with an 'X' in the Key_Form IM column in Table 78 on page 220.	Key Generate – Key set
The key-form and key-type combinations shown with an 'X 'in the Key_Form EX column in Table 78 on page 220.	Key Generate - Key set
The key-form and key-type combinations shown with an 'X' in Table 79 on page 221	Key Generate - Key set
The key-form and key-type combinations shown with an 'E' in Table 79 on page 221	Key Generate - Key set extended
The SINGLE-R key-length keyword is specified	Key Generate - SINGLE-R

To use a NOCV IMPORTER key-encrypting key with the key generate service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

To use a NOCV EXPORTER key-encrypting key with the key generate service, the **NOCV KEK usage for export-related functions** access control point must be enabled in addition to one or both of the access control points listed.

To use the SINGLE-R rule array keyword, the **Key Generate – SINGLE-R** access control point must be enable.

If a key-encrypting key identifier is a weaker key than the key being generated, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key or the key-encrypting key is a double-length key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 81. Key generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Generation of CIPHER keys in key form OP, EX, and IM requires the July 2015 or later licensed internal code (LIC). Key lengths TRIPLE and TRIPLE-O and triple-length DES keys require the July 2019 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.

Table 81. Key generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Generation of CIPHER keys in key form OP, EX, and IM requires the July 2015 or later licensed internal code (LIC). Key lengths TRIPLE and TRIPLE-O and triple-length DES keys require the December 2018 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Key lengths TRIPLE and TRIPLE-O and triple-length DES keys require the December 2018 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in output DES CIPHER key tokens requires a CEX6C with the P41458.002 or later MCL. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The CPACF export bit (XPRTCPAC) in the output key token is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Key Generate2 (CSNBKGN2 and CSNEKGN2)

Use the Key Generate2 callable service to generate either one or two CCA key tokens of any type. This callable service does not produce keys in clear form and all keys are returned in encrypted form. When two keys are generated, each key has the same clear value, although this clear value is not exposed outside the secure cryptographic feature.

This service returns variable-length CCA key tokens and uses the AESKW wrapping method.

This service supports HMAC and AES keys. Operational keys will be encrypted under the AES master key.

Some key types are not directly supported by this service because there is no default key usage value. These key types can be generated by using the TOKEN keyword and a skeleton token from the Key Token Build2 service. These AES key types require TOKEN be used: DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW.

The callable service name for AMODE(64) is CSNEKGN2.

Format

```
CALL CSNBKGN2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             clear_key_bit_length,
             key_type_1,
             key_type_2,
             key_name_1_length,
             key_name_1,
             key_name_2_length,
             key_name_2,
             user_associated_data_1_length,
             user_associated_data_1,
             user_associated_data_2_length,
             user_associated_data_2,
             key_encrypting_key_identifier_1_length,
             key_encrypting_key_identifier_1,
             key_encrypting_key_identifier_2_length,
             key_encrypting_key_identifier_2,
             generated_key_identifier_1_length,
             generated_key_identifier_1,
             generated_key_identifier_2_length,
             generated_key_identifier_2 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. Valid values are 2, 3 or 4.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 82. Keywords for Key Generate2 Control Information

Keyword	Meaning
Token algorithm (required)	
HMAC	Specifies to generate an HMAC key token.
AES	Specifies to generate an AES key token.
Key Form (required) The first two characters refer to `key_type_1`. The next two characters refer to `key_type_2`. See the Usage Notes section for further details.	
EX	One key that can be sent to another system.
EXEX	A key pair; both keys to be sent elsewhere, possibly for exporting to two different systems. Both keys have the same clear value.
IM	One key that can be locally imported. The key can be imported onto this system to make it operational at another time.
IMEX	A key pair to be imported; one key to be imported locally and one key to be sent elsewhere. Both keys have the same clear value.
IMIM	A key pair to be imported; both keys to be imported locally at another time. Both keys have the same clear value.
OP	One operational key. The key is returned to the caller in operational form to be used locally.
OPEX	A key pair; one key that is operational and one key to be sent elsewhere. Both keys have the same clear value.
OPIM	A key pair; one key that is operational and one key to be imported locally at another time. Both keys have the same clear value.

Table 82. Keywords for Key Generate2 Control Information (continued)	
Keyword	**Meaning**
OPOP	A key pair; either with the same key type with different associated data or complementary key types. Both keys have the same clear value.
Payload Version for generated_key_identifier_1 (one, optional) **Note:** This keyword overrides payload format version of any corresponding skeleton token.	
V0PYLDK1	Build a token with the old variable-length payload format for the generated_key_identifier_1 parameter. This is the default for AES CIPHER, EXPORTER, and IMPORTER key types and is only valid with those key types.
V1PYLDK1	Build a token with the new fixed-length payload format for the generated_key_identifier_1 parameter. This is the default for AES MAC, PINPROT, PINCALC, PINPRW, and DKYGENKY key types. Not valid with the HMAC MAC key type.
Payload Version for generated_key_identifier_2 (one, optional) **Note:** This keyword overrides payload format version of any corresponding skeleton token.	
V0PYLDK2	Build a token with the old variable-length payload format for the generated_key_identifier_2 parameter. This is the default for AES CIPHER, EXPORTER, and IMPORTER key types and is only valid with those key types.
V1PYLDK2	Build a token with the new fixed-length payload format for the generated_key_identifier_2 parameter. This is the default for AES MAC, PINPROT, PINCALC, PINPRW, and DKYGENKY key types. Not valid with the HMAC MAC key type.

clear_key_bit_length

Direction	Type
Input	Integer

The size (in bits) of the key to be generated.

- For the HMAC algorithm, this is a value between 80 and 2048, inclusive.
- For the AES algorithm, this is a value of 128, 192, or 256.

When *key_type_1* or *key_type_2* is TOKEN, this value overrides the key length contained in *generated_key_identifier_1* or *generated_key_identifier_2*, respectively.

key_type_1

Direction	Type
Input	String

Use the *key_type_1* parameter for the first, or only, key that you want generated. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form, and are documented in Table 85 on page 232 and Table 86 on page 233.

The 8-byte keyword for the *key_type_1* parameter can be one of the following:

Table 83. Keywords and associated algorithms for key_type_1 parameter	
Keyword	**Algorithm**
CIPHER	AES
EXPORTER	AES
IMPORTER	AES
MAC	HMAC
MACVER	HMAC
Specify the keyword TOKEN when supplying a key token in the *generated_key_identifier_1* parameter.	

If *key_type_1* is TOKEN, the associated data in the *generated_key_identifier_1* parameter is examined to derive the key type.

key_type_2

Direction	**Type**
Input	String

Use the *key_type_2* parameter for a key pair, which is shown in Table 86 on page 233. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form.

The 8-byte keyword for the *key_type_2* parameter can be one of the following:

Table 84. Keywords and associated algorithms for key_type_2 parameter	
Keyword	**Algorithm**
CIPHER	AES
EXPORTER	AES
IMPORTER	AES
MAC	HMAC
MACVER	HMAC
Specify the keyword TOKEN when supplying a key token in the *generated_key_identifier_2* parameter.	

If *key_type_2* is TOKEN, the associated data in the *generated_key_identifier_2* parameter is examined to derive the key type.

When only one key is being generated, this parameter is ignored.

key_name_1_length

Direction	**Type**
Input	Integer

The length of the *key_name* parameter for *generated_key_identifier_1*. Valid values are 0 and 64 bytes.

key_name_1

Direction	**Type**
Input	String

A 64-byte key store label to be stored in the associated data structure of *generated_key_identifier_1*.

key_name_2_length

Direction	Type
Input	Integer

The length of the *key_name* parameter for *generated_key_identifier_2*. Valid values are 0 and 64 bytes.

When only one key is being generated, this parameter is ignored.

key_name_2

Direction	Type
Input	String

A 64-byte key store label to be stored in the associated data structure of *generated_key_identifier_2*.

When only one key is being generated, this parameter is ignored.

user_associated_data_1_length

Direction	Type
Input	Integer

The length of the user-associated data parameter for *generated_key_identifier_1*. The valid values are 0 to 255 bytes.

user_associated_data_1

Direction	Type
Input	String

User-associated data to be stored in the associated data structure for *generated_key_identifier_1*.

user_associated_data_2_length

Direction	Type
Input	Integer

The length of the user-associated data parameter for *generated_key_identifier_2*. The valid values are 0 to 255 bytes.

When only one key is being generated, this parameter is ignored.

user_associated_data_2

Direction	Type
Input	String

User associated data to be stored in the associated data structure for *generated_key_identifier_2*.

When only one key is being generated, this parameter is ignored.

key_encrypting_key_identifier_1_length

Direction	Type
Input	Integer

The length of the buffer for *key_encrypting_key_identifier_1* in bytes.

When the Key Form rule is OP, OPOP, OPIM, or OPEX, this length must be zero.

When the Key Form rule is EX, EXEX, IM, IMEX, or IMIM, the value must be 64 bytes when *key_encrypting_key_identifier_1* is a label. Otherwise, the value must be between the actual length of the token and 9992 bytes.

key_encrypting_key_identifier_1

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to wrap the generated key 1 when the key form indicates an external key is generated. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

When *key_encrypting_key_identifier_1_length* is zero, this parameter is ignored.

For CCA key, the identifier is a variable-length AES key token of key type EXPORTER or IMPORTER with key management attributes enable to allow the key to wrap the type of key being generated.

For X9.143 key blocks, the identifier is a key block of an AES key-encrypting key: key usage K0, algorithm A, and mode of use D or E.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

key_encrypting_key_identifier_2_length

Direction	Type
Input	Integer

The length of the buffer for *key_encrypting_key_identifier_2* in bytes.

When the Key Form rule is OPOP, this length must be zero.

When the Key Form rule is EXEX, IMEX, IMIM, OPIM, or OPEX, the value must be 64 when *key_encrypting_key_identifier_2* is a label. Otherwise, the value must be between the actual length of the token and 9992.

When only one key is being generated, this parameter is ignored.

key_encrypting_key_identifier_2

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to wrap the generated key 2 when the key form indicates an external key is generated. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

When *key_encrypting_key_identifier_2_length* is zero or only one key is being generated, this parameter is ignored.

For CCA key, the identifier is a variable-length AES key token of key type EXPORTER or IMPORTER with key management attributes enable to allow the key to wrap the type of key being generated.

For X9.143 key blocks, the identifier is a key block of an AES key-encrypting key: key usage K0, algorithm A, and mode of use D or E.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

generated_key_identifier_1_length

Direction	Type
Input/Output	Integer

On input, the length of the buffer for the *generated_key_identifier_1* parameter in bytes. The maximum value is 900 bytes.

On output, the parameter will hold the actual length of the *generated_key_identifier_1*.

generated_key_identifier_1

Direction	Type
Input/Output	String

The buffer for the first generated key token.

On input, if you specify a key_type_1 of TOKEN, then the buffer contains a valid key token of the key type you want to generate. The key token must be left justified in the buffer. See *key_type_1* for a list of valid key types.

On output, the buffer contains the generated key token.

To generate a compliant-tagged key token, a compliant-tagged skeleton token must be supplied.

generated_key_identifier_2_length

Direction	Type
Input/Output	Integer

On input, the length of the buffer for the *generated_key_identifier_2* in bytes. The maximum value is 900 bytes.

On output, the parameter will hold the actual length of the *generated_key_identifier_2*.

When only one key is being generated, this parameter is ignored.

generated_key_identifier_2

Direction	Type
Input/Output	String

The buffer for the second generated key token.

On input, if you specify a *key_type_2* of TOKEN, then the buffer contains a valid key token of the key type you want to generate. The key token must be left justified in the buffer. See *key_type_2* for a list of valid key types.

On output, the buffer contains the generated key token.

When only one key is being generated, this parameter is ignored.

To generate a compliant-tagged key token, a compliant-tagged skeleton token must be supplied.

Usage notes

The key forms are defined as follows:

Operational (OP)
The key value is enciphered under a master key. The result is placed into an internal key token. The key is then operational at the local system.

Importable (IM)
The key value is enciphered under an importer key-encrypting key. The result is placed into an external key token. The corresponding *key_encrypting_key_identifier_x* parameter must contain an AES IMPORTER key token or label.

Exportable (EX)
The key value is enciphered under an exporter key-encrypting key. The result is placed into an external key token. The corresponding *key_encrypting_key_identifier_x* parameter must contain an AES EXPORTER key token or label.

The following tables list the valid key type and key form combinations and required access control points.

The key usage attributes that are shown are those that are required. Key usage attributes that are not shown are optional. When key usage attributes are not the default values for a key type, a skeleton key token with the desired attributes must be supplied and a key type must be **TOKEN**.

Table 85 on page 232 lists all key types that can be generated as a single key. The **Key Generate2 - OP** access control point must be enabled.

The key types marked with an asterisk (*) must be requested through the specification of a proper key usage field in a key token and the use of the **TOKEN** keyword.

Table 85. Key Generate2 valid key type and key form for one AES or HMAC key

key_type_1 (key usage)	Key Form OP, IM, EX	Notes
CIPHER (ENCRYPT, DECRYPT)	X	If you supply a skeleton token, the key usage must allow decryption and encryption.
***DKYGENKY (D-ALL)**	X	
***DKYGENKY (D-CIPHER)**	X	If you supply a skeleton token, the key usage for the derived key must allow decryption and encryption.
***DKYGENKY (D-MAC)**	X	If you supply a skeleton token, the generate control key usage for the derived key must be GENERATE.
***DKYGENKY (D-PCALC)**	X	
***KDKGENKY**	X	
HMAC MAC (GENERATE)	X	If you supply a skeleton token, the generate control key usage must be GENERATE.
AES MAC (GENERATE)	X	The generate control key usage in the skeleton must be GENERATE.

Table 86 on page 233 lists all pairs of keys that can be generated and the key forms that are allowed. The key forms that are marked with an 'X' required the **Key Generate2 - Key set** access control point to be enabled. The key forms marked with an 'E' required the **Key Generate2 - Key set extended** access control point to be enabled.

The key types marked with an asterisk (*) must be requested through the specification of a proper key usage field in a key token and the use of the **TOKEN** keyword.

Table 86. Key Generate2 Valid key type and key forms for two AES or HMAC keys

key_type_1 (key usage)	key_type_2 (key usage)	Key Form OPOP OPIM IMIM	Key Form OPEX	Key Form EXEX	Key Form IMEX
CIPHER (DECRYPT, ENCRYPT)	CIPHER (DECRYPT, C-XLATE)	X	X	X	X
CIPHER (DECRYPT, ENCRYPT)	CIPHER (DECRYPT, ENCRYPT, C-XLATE)	X	X	X	X
CIPHER (DECRYPT, ENCRYPT)	CIPHER (ENCRYPT, C-XLATE)	X	X	X	X
CIPHER (DECRYPT)	CIPHER (ENCRYPT, C-XLATE)	X	X	X	X
CIPHER (DECRYPT, C-XLATE)	CIPHER (DECRYPT, ENCRYPT)	X	X	X	X
CIPHER (DECRYPT, ENCRYPT, C-XLATE)	CIPHER (DECRYPT, ENCRYPT)	X	E	X	E
CIPHER (ENCRYPT, C-XLATE)	CIPHER (DECRYPT, ENCRYPT)	X	E	X	E
CIPHER (ENCRYPT, C-XLATE)	CIPHER (DECRYPT)	X	E	X	E
CIPHER (DECRYPT, C-XLATE)	CIPHER (ENCRYPT)	X	E	X	E
CIPHER (DECRYPT, ENCRYPT, C-XLATE)	CIPHER (DECRYPT, ENCRYPT, C-XLATE)		E	X	E
CIPHER (DECRYPT, C-XLATE)	CIPHER (ENCRYPT, C-XLATE)		E	X	E
CIPHER (ENCRYPT, C-XLATE)	CIPHER (DECRYPT, C-XLATE)		E	X	E
***DKYGENKY**	*DKYGENKY	X	X	X	X
EXPORTER	IMPORTER		X	X	X
IMPORTER	EXPORTER		X	X	X
***KDKGENKY (KDKTYPEA)**	*KDKGENKY (KDKTYPEB)		X	X	X
***KDKGENKY (KDKTYPEB)**	*KDKGENKY (KDKTYPEA)		X	X	X
MAC (GENERATE)	MAC (GENERATE)	X	X	X	X
MAC (GENERATE)	MAC (VERIFY)	X	X	X	X
MAC (GENERATE)	MAC (GENONLY)	X	X	X	X
MAC (GENONLY)	MAC (GENERATE)	X	X	X	X
MAC (GENONLY)	MAC (VERIFY)	X	X	X	X
MAC (VERIFY)	MAC (GENERATE)	X	X	X	X

Table 86. Key Generate2 Valid key type and key forms for two AES or HMAC keys (continued)					
key_type_1 (key usage)	**key_type_2 (key usage)**	**Key Form OPOP OPIM IMIM**	**Key Form OPEX**	**Key Form EXEX**	**Key Form IMEX**
MAC (VERIFY)	MAC (GENONLY)	X	X	X	X
PINPROT(ENCRYPT, NOFLDFMT)	PINPROT(DECRYPT, NOFLDFMT)	E	X	X	X
PINPROT(DECRYPT, NOFLDFMT)	PINPROT(ENCRYPT, NOFLDFMT)	E	X	X	X
PINPROT(ENCRYPT, NOFLDFMT)	PINPROT(ENCRYPT, NOFLDFMT)	X			
PINPROT (DECRYPT, NOFLDFMT, ISO-4, EPINVER)	PINPROT (ENCRYPT, NOFLDFMT, ISO-4, REFORMAT)	OPOP only[1]			
PINPROT (ENCRYPT, NOFLDFMT, ISO-4, REFORMAT)	PINPROT (DECRYPT, NOFLDFMT, ISO-4, EPINVER)	OPOP only[1]			

[1] Requires access control point Key Generate2 - Allow GEN of OPOP EPVR/OPIN Key Pair.

See Table 90 on page 236 for an explanation of the differences between **E** as compared to **X**.

Note: A pair of DKYGENKY keys can be used to diversify a pair of keys with different key types and key usage attributes. The combination of key types and key usage attributes that can be diversified must meet the requirements of using the KGN2 verb to generate those same keys. A DKYGENKY key with D-ALL usage can only be paired with a DKYGENKY key with D-ALL usage.

For keys for the German Banking Industry Committee (Deutsche Kreditwirtschaft (DK)) PIN method, a key token with the proper key-usage values must be supplied. The key type 1 and 2 are TOKEN. Table 87 on page 234 shows the valid key pairs. Access control points are required to be enabled for the generation of these keys.

Table 87. Valid key pairs that can be generated and their required access points	
Access Control Point	**Table identifier**
Key Generate2 - OP	O
Key Generate2 - Key set	X
Key Generate2 - DK PIN key set	D
Key Generate2 - DK PIN admin1 key PINPROT	D1P
Key Generate2 - DK PIN admin1 key MAC	D1M
Key Generate2 - DK PIN print key	DP
Key Generate2 - DK PIN admin2 key MAC	D2M

Table 88. Key type and key form keywords for AES keys - DK PIN methods					
key type 1 (key usage)	**key type 2 (key usage)**	**Key Form OPOP OPIM IMIM**	**Key Form OPEX IMEX**	**Key Form EXEX**	**Key Form OP EX IM**
MAC(GENONLY, DKPINOP)	MAC(VERIFY, DKPINOP)	D	X		
MAC(VERIFY, DKPINOP)	MAC(GENONLY, DKPINOP)	D			
MAC(GENONLY, DKPINAD1)	MAC(VERIFY, DKPINAD1)	D1M	D1M		
MAC(VERIFY, DKPINAD1)	MAC(GENONLY, DKPINAD1)	D1M			
MAC(GENONLY, DKPINAD2)	MAC(VERIFY, DKPINAD2)	D	D2M		
MAC(VERIFY, DKPINAD2)	MAC(GENONLY, DKPINAD2)	D			
PINCALC(GENONLY, DKPINOP)					O
PINPROT(ENCRYPT, DKPINOP)	PINPROT(DECRYPT, DKPINOP)	D	X		
PINPROT(DECRYPT, DKPINOP)	PINPROT(ENCRYPT, DKPINOP)	D			
PINPROT(ENCRYPT, DKPINAD1)	PINPROT(DECRYPT, DKPINAD1)	D	D1P		
PINPROT(DECRYPT, DKPINAD1)	PINPROT(ENCRYPT, DKPINAD1)	D			
PINPROT(ENCRYPT, DKPINOPP)	CIPHER(DECRYPT)	D	DP		
CIPHER(DECRYPT)	PINPROT(ENCRYPT, DKPINOPP)	D			
PINPRW(GENONLY, DKPINOP)	PINPRW(VERIFY, DKPINOP)	X	X		
PINPRW(VERIFY, DKPINOP)	PINPRW(GENONLY, DKPINOP)	X			

The strength of the key-encrypting key used to wrap a generated key will affect the results of the service. The resulting return code and reason code when using a key-encrypting key that is weaker than the key being generated depends on the **Prohibit weak wrapping - Transport keys** and **Warn when weak wrap - Transport keys** access control points:

- If the **Prohibit weak wrapping - Transport keys** access control point is disabled, the key strength requirement will not be enforced. Using a weaker key will result in return code 0 with a non-zero reason code if the **Warn when weak wrap - Transport keys** access control point is enabled. Otherwise, a reason code of zero will be returned.
- If the **Prohibit weak wrapping - Transport keys** access control point is enabled, the key strength requirement will be enforced, and attempting to use a weaker key will result in return code 8.

For AES keys, the AES KEK must be at least as strong as the key being generated to be considered sufficient strength.

For HMAC keys, the AES KEK must be sufficient strength as described in the following table.

Table 89. AES KEK strength required for generating an HMAC key under an AES KEK	
Key-usage field 2 in the HMAC key contains	**Minimum strength of AES KEK to adequately protect the HMAC key**
SHA-256, SHA-384, SHA-512	256 bits
SHA-224	192 bits
SHA-1	128 bits

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the generated key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 90. Required access control points for Key Generate2	
Access Control Point	**Function control**
Key Generate2 - Allow GEN of OPOP EPVR/ OPIN Key Pair	Generation of OPOP EPVR/OPIN key pair.
Key Generate2 - OP	Key Form OP, EX, IM.
Key Generate2 - Key set	The key-form and key-type combinations shown with an X in Table 86 on page 233 and Table 88 on page 235.
Key Generate2 - Key set extended	The key-form and key-type combinations shown with an E in Table 86 on page 233.
Key Generate2 - DK PIN key set	The key-form and key-type combinations shown with an D in Table 88 on page 235.
Key Generate2 - DK PIN admin1 set PINPROT	The key-form and key-type combinations shown with an D1P in Table 88 on page 235.
Key Generate2 - DK PIN admin1 set MAC	The key-form and key-type combinations shown with an D1M in Table 88 on page 235.
Key Generate2 - DK PIN print key	The key-form and key-type combinations shown with an DP in Table 88 on page 235.
Key Generate2 - DK PIN admin2 set MAC	The key-form and key-type combinations shown with an D2 in Table 88 on page 235.
Prohibit weak wrapping - Transport keys	Prohibit wrapping a key with a weaker key.
Warn when weak wrap - Transport keys	Issue a non-zero reason code when using a weak wrapping key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 91. Key Generate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	AES key type KDKGENKY requires the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	AES key type KDKGENKY requires the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	AES key type KDKGENKY requires the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Key Import (CSNBKIM and CSNEKIM)

Use the key import callable service to reencipher a DES key from encryption under an importer key-encrypting key to encryption under the master key. The reenciphered key is in operational form.

Choose one of these options:

- Specify the *key_type* parameter as TOKEN and specify the external key token in the *source_key_identifier* parameter. The key type information is determined from the control vector in the external key token.
- Specify a key type in the *key_type* parameter and specify an external key token in the *source_key_identifier* parameter. The specified key type must be compatible with the control vector in the external key token.
- Specify a valid key type in the *key_type* parameter and a null key token in the *source_key_identifier* parameter. The default control vector for the *key_type* specified will be used to process the key.

For DATA keys, this service generates a key of the same length as that contained in the input token.

The callable service name for AMODE(64) invocation is CSNEKIM.

Format

```
CALL CSNBKIM(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_type,
             source_key_identifier,
             importer_key_identifier,
             target_key_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_type

Direction	Type
Input	Character String

The type of key you want to reencipher under the master key. Specify an 8-byte keyword or the keyword TOKEN. The keyword must be left-justified and padded on the right with blanks.

If the key type is TOKEN, ICSF determines the key type from the control vector (CV) field in the external key token provided in the *source_key_identifier* parameter.

TOKEN is never allowed when the importer_key_identifier is NOCV.

Supported key_type values are CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, DATA, DATAM, DATAMV, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEXLAT, OPINENC, PINGEN and PINVER. Use key_type TOKEN for all other key types.

source_key_identifier

Direction	Type
Input	String

The key you want to reencipher under the master key. The parameter is a 64-byte field for the enciphered key to be imported containing either an external key token or a null key token.

If you specify a null token, start with 64 bytes of binary zeros and then insert the encrypted key values as follows:

- Bytes 16-23 for the first or only key part.
- Bytes 24-31 for a second key part (double-length and triple-length keys only).
- Bytes 48-55 for a third key part (triple-length keys only). Only the DATA key type is supported when importing a triple-length key.

The use of a null token works only when the legacy wrapping method is used. If the key value is enhanced wrapped, an external key token must be used. The CSNBKTB service can be used to create the token if you only have the encrypted key value.

If key type is TOKEN, this field may not specify a null token.

This service supports the no-export function in the CV.

importer_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to unwrap the source key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

When the identifier is a label, it is 64-byte string, left-justified and padded with blanks.

For CCA keys, the identifier is a 64-byte DES key token of key type IMPORTER.

For X9.143 keys, the identifier is a variable-length key block of a TDES importer key-encrypting key usage K0, algorithm T, and mode of use D.

Note: If you specify a CCA NOCV importer in the *importer_key_identifier* parameter, the key to be imported must be enciphered under the importer key itself.

If the key token or key block supplied was encrypted under the old master key, the key token or key block will be returned encrypted under the current master key.

target_key_identifier

Direction	Type
Input/Output	String

On input, this parameter contains a 64-byte key token.

- When the WRAPENH3 method is required, a WRAPENH3 skeleton key token or WRAPENH3 internal key token must be specified.
- Otherwise, a null token.

On output, this parameter receives the imported operational key token.

When the imported key TYPE is IMPORTER or EXPORTER and the token key TYPE is the same, and the application passes a valid internal key token for an IMPORTER or EXPORTER key in this parameter, the NOCV bit is propagated to the imported key token.

To import a key token as a compliant-tagged key token, a compliant-tagged skeleton token must be supplied.

The output *target_key_identifier* will use the default wrapping method unless a skeleton token is supplied as input. If a skeleton token is supplied as input, the wrapping method in the skeleton token will be used.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

Restrictions

The DES IMP-PKA key type is not supported.

Current applications will fail if they use an equal key halves importer to import a key with unequal key halves. You must have access control point 'Key Import - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Use of NOCV keys are controlled by an access control point. Creation of NOCV key-encrypting keys is only available for standard IMPORTERs and EXPORTERs.

This service will mark an imported KEK as a NOCV-KEK:

- If a token is supplied in the target token field, it must be a valid importer or exporter token. If the token fails token validation, processing continues, but the NOCV flag will not be copied.
- The source token (key to be imported) must be an importer or exporter with a valid control vector.
- If the target token is valid and the NOCV flag is on and the control vector of the target token is exactly the same as the source token, the imported token will have the NOCV flag set on.
- If the target token is valid and the NOCV flag is on and the control vector of the target token is NOT exactly the same as the source token, a non-zero return code will be given.
- All other scenarios will complete successfully, but the NOCV flag will not be copied

The software bit used to mark the imported token with export prohibited is not supported. The internal token for an export prohibited key will have the appropriate control vector that prohibits export.

For key types CIPHERXI, CIPHERXL, and CIPHERXO, the key-encrypting key in the importer_key_identifier parameter must have a control vector with the key halves guaranteed unique flag on in the key form bits. An existing key-encrypting key can have its control vector updated using the restrict key attribute callable service.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 92. Required access control points for Key Import	
Access Control Point	**Restrictions**
Key Import	None
Key Import - Unrestricted	Key-encrypting key may have equal key halves

To use a NOCV key-encrypting key with the key import service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 93. Key import required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in output DES CIPHER key tokens requires a CEX6C with the P41458.002 or later MCL. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 93. Key import required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The CPACF export bit (XPRTCPAC) in the output key token is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Key Part Import (CSNBKPI and CSNEKPI)

Use the Key Part Import callable service to combine, by exclusive ORing, the clear key parts of any DES key type and return the combined key value either in an internal token or as an update to the CKDS.

Prior to using the Key Part Import service for the first key part, you must use the key token build service to create the internal key token into which the key will be imported. Subsequent key parts are combined with the first part in internal token form or as a label from the CKDS.

The preferred way to specify key parts is FIRST, ADD-PART, and COMPLETE in the *rule_array*. Only when the combined key parts have been marked as COMPLETE can the key token be used in any other service. Key parts can also be specified as FIRST, MIDDLE, or LAST in the *rule_array*. ADD-PART or MIDDLE can be executed multiple times for as many key parts as necessary. Only when the LAST part has been combined can the key token be used in any other service.

New applications should employ the ADD-PART and COMPLETE keywords in lieu of the MIDDLE and LAST keywords in order to ensure a separation of responsibilities between someone who can add key-part information and someone who can declare that appropriate information has been accumulated in a key.

The Key Part Import callable service can also be used to import a key without using key parts. Call the Key Part Import service FIRST with key part value X'0000...' then call the Key Part Import service LAST with the complete value.

Keys created via this service have odd parity. The FIRST key part is adjusted to odd parity. All subsequent key parts are adjusted to even parity prior to being combined.

The callable service name for AMODE(64) invocation is CSNEKPI.

Format

```
CALL CSNBKPI(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
```

```
                key_part,
                key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1, 2, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 94. Keywords for Key Part Import Control Information

Keyword	Meaning
Key Part (Required)	

Table 94. Keywords for Key Part Import Control Information (continued)	
Keyword	**Meaning**
FIRST	This keyword specifies that an initial key part is being entered. The callable service returns this key-part encrypted by the master key in the key token that you supplied.
ADD-PART	This keyword specifies that additional key-part information is provided.
COMPLETE	This keyword specifies that the key-part bit shall be turned off in the control vector of the key rendering the key fully operational. Note that no key-part information is added to the key with this keyword.
MIDDLE	This keyword specifies that an intermediate key part, which is neither the first key part nor the last key part, is being entered. Note that the command control point for this keyword is the same as that for the LAST keyword and different from that for the ADD-PART keyword.
LAST	This keyword specifies that the last key part is being entered. The key-part bit is turned off in the control vector.
Key part buffer length (one, optional)	
KEYBUF16	Specifies a length of 16 bytes for the buffer identified by the *key_part* parameter. This is the default.
KEYBUF24	Specifies a length of 24 bytes for the buffer identified by the *key_part* parameter. This is required for triple-length keys when key part keyword COMPLETE is not specified.
Key Wrapping Method (optional) **Note:** When a wrapping method keyword is specified, it must be specified on all calls for a key including the COMPLETE call.	
USECONFG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword. The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the *z/OS Cryptographic Services ICSF System Programmer's Guide*.
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This service will set CV bit 56 = B'1' (ENH-ONLY), which is required for the WRAPENH2 wrapping method.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code. This service will set CV bit 56 = B'1' (ENH-ONLY), which is required for the WRAPENH3 wrapping method.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens.

key_part

Direction	Type
Input	String

The 16-byte or 24-byte buffer containing the clear key part to be imported. When the key part buffer length keyword is KEYBUF16, a 16-byte clear key part is imported. This keyword is used for single-length and double-length keys. For KEYBUF24, a 24-byte key part is imported. This keyword is used for triple-length keys.

If the key is a single-length key, the 8-byte key part must be left-justified and padded on the right with zeros.

This field is ignored when COMPLETE is specified.

key_identifier

Direction	Type
Input/Output	String

A 64-byte field containing an internal token or a label of an existing CKDS record. If *rule_array* is FIRST, this field is the skeleton of an internal token with the KEY-PART marking. If *rule_array* is MIDDLE or LAST, this is an internal token or the label of a CKDS record of a partially combined key. Depending on the input format, the accumulated partial or complete key is returned as an internal token or as an updated CKDS record. The returned *key_identifier* will be encrypted under the current master key.

The DES key wrapping methods available are described in “CCA key wrapping” on page 20.

The output *key_identifier* uses the default wrapping method unless a rule array keyword overriding the default for the FIRST key part is specified. Triple-length keys with a non-zero control vector can be wrapped with the WRAPENH2 or WRAPENH3 method. When the WRAPENH3 method is selected, a skeleton key token is required when the first key part is supplied. A secure internal key token wrapped with the WRAPENH3 method obfuscates the key length. When the *key_identifier* is an existing token, the same wrapping method as the existing token will be used

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

Restrictions

If a label is specified on *key_identifier*, the label must be unique. If more than one record is found, the service fails.

You must have access control point 'Key Part Import - Unrestricted' explicitly enabled. Otherwise, current applications will fail with either of these conditions:

- The first 8 bytes of key identifier is different than the second 8 bytes AND the first 8 bytes of the combined key are the same as the last second 8 bytes.
- The first 8 bytes of key identifier is the same as the second 8 bytes AND the first 8 bytes of the combined key are different than the second 8 bytes.

This callable service only supports version X'00' or X'01' DES key tokens.

Usage notes

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 95. Required access control points for Key Part Import

Rule array keyword	Access control point
FIRST	Key Part Import - first key part
MIDDLE or LAST	Key Part Import - middle and last
ADD-PART	Key Part Import - ADD-PART
COMPLETE	Key Part Import - COMPLETE

When the key wrapping method keyword specifies a wrapping method that is not the default method, the **Key Part Import - Allow wrapping override keywords** access control must be enabled.

A "replicated key-halves" key (both cleartext halves of a double-length key are equal) is not as secure as a double-length key with key halves that are not the same. The Key Part Import service verb enforces the key-halves restriction when the **Key Part Import - Unrestricted** access control point is disabled in the domain role.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 96. Key Part Import required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys and keywords KEYBUF16 and KEYBUF24 require the July 2019 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).

Table 96. Key Part Import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys and keywords KEYBUF16 and KEYBUF24 require the July 2019 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Triple-length DES keys and keywords KEYBUF16 and KEYBUF24 require the December 2018 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in output DES CIPHER key tokens requires a CEX6C with the P41458.002 or later MCL. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	The CPACF export bit (XPRTCPAC) in the output key token is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express7 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Related information

This service is consistent with the Transaction Security System key part import verb.

Key Part Import2 (CSNBKPI2 and CSNEKPI2)

Use the Key Part Import2 callable service to combine, by exclusive ORing, the clear key parts of any key type and return the combined key value either in a variable-length internal token or as an update to the CKDS.

Prior to using the key part import2 service for the first key part, you must use the Key Token Build2 service to create the internal key token into which the key will be imported. Subsequent key parts are combined with the first part in internal token form or as a label from the CKDS.

On each call to Key Part Import2 (except with the COMPLETE keyword), specify the number of bits to use for the clear key part. Place the clear key part in the *key_part* parameter, and specify the number of bits using the *key_part_length* variable. Any extraneous bits of *key_part* data will be ignored.

Consider using the Key Test2 callable service to ensure a correct key value has been accumulated prior to using the COMPLETE option to mark the key as fully operational.

The callable service name for AMODE(64) is CSNEKPI2.

Format

```
CALL CSNBKPI2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_part_bit_length,
             key_part,
             key_identifier_length,
             key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 2 or 3.

rule_array

Direction	Type
Input	Integer

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 97. Keywords for Key Part Import2 Control Information

Keyword	Meaning
Token Algorithm (Required)	
AES	Specifies to import an AES key token.
DES	Specifies to import a DES key. This keyword is only valid with a TR-31 key block.
HMAC	Specifies to import an HMAC key token.
Key Part (One required)	
FIRST	This keyword specifies that an initial key part is being entered. The callable service returns this key-part encrypted by the master key in the key token that you supplied.
ADD-PART	This keyword specifies that additional key-part information is provided.
COMPLETE	This keyword specifies that the key-part bit shall be turned off in the control vector of the key rendering the key fully operational. Note that no key-part information is added to the key with this keyword.
Split Knowledge (One required). Use only with FIRST keyword.	
MIN3PART	Specifies that the key must be entered in at least three parts.
MIN2PART	Specifies that the key must be entered in at least two parts.
MIN1PART	Specifies that the key must be entered in at least one part.

key_part_bit_length

Direction	Type
Input	Integer

The length of the clear key in bits. This indicates the bit length of the key supplied in the *key_part* field.

For FIRST and ADD-PART keywords, valid values are:

For HMAC keys
80 to 2048.

For AES keys
128, 192, or 256.

For DES keys
64, 128, or 192.

The value must be 0 for the COMPLETE keyword.

key_part

Direction	Type
Input	String

This parameter is the clear key value to be applied. The key part must be left-justified. This parameter is ignored if COMPLETE is specified.

key_identifier_length

Direction	Type
Input/Output	Integer

On input, the length of the buffer for the *key_identifier* parameter. For labels, the value is 64 bytes. The *key_identifier* must be left justified in the buffer. The buffer must be large enough to receive the updated token. The maximum value is 9992 bytes. The output token will be longer when the first key part is imported.

On output, the actual length of the token returned to the caller. For labels, the value will be 64.

key_identifier

Direction	Type
Input/Output	String

The parameter containing an internal key token or key block or a 64-byte label of an existing CKDS record.

If the Key Part rule is FIRST, the key is a skeleton token or block.

If the Key Part rule is ADD-PART, this is an internal token or block or the label of a CKDS record of a partially combined key.

Depending on the input format, the accumulated partial or complete key is returned as an internal token or block or as an updated CKDS record.

The returned *key_identifier* will be encrypted under the current master key

Usage notes

On each call to Key Part Import2, also specify a rule-array keyword to define the service action: FIRST, ADD-PART, or COMPLETE.

- With the FIRST keyword, the input key-token must be a skeleton token (no key material).
- With the ADD-PART keyword, the service exclusive-ORs the clear key-part with the key value in the input key-token. The key remains incomplete in the updated key token returned from the service.
- With the COMPLETE keyword, the KEY-PART bit is set off in the updated key token that is returned from the service. The *key_part_bit_length* parameter must be set to zero.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 98. Required access control points for Key Part Import2

Rule array keywords	Access control point
ADD-PART	Key Part Import2 - Add second of 3 or more key parts
ADD-PART	Key Part Import2 - Add last required key part
ADD-PART	Key Part Import2 - Add optional key part
COMPLETE	Key Part Import2 - Complete key
FIRST MIN3PART	Key Part Import2 - Load first key part, require 3 key parts
FIRST MIN2PART	Key Part Import2 - Load first key part, require 2 key parts
FIRST MIN1PART	Key Part Import2 - Load first key part, require 1 key parts

In addition, the **KPI2 - Allow TR-31 clear key import** access control point is required when *key_identifier* is a TR-31 key block or the label of a TR-31 key block.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 99. Key Part Import2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Key Test (CSNBKYT and CSNEKYT)

Use the Key Test callable service to generate or verify a secure, cryptographic verification pattern for DES and AES keys. Encrypted keys are specified in internal, fixed-length key tokens. Keywords in the *rule_array* specify whether the callable service generates or verifies a verification pattern.

DES keys use the algorithm defined in "DES algorithm (single-length and double-length keys)" on page 1654 as the default algorithm (except for triple-length keys). When generating a verification pattern, the service generates a random number and calculates the verification pattern. The random number and verification pattern are returned to the caller. When verifying a key, the random number and key are used to verify the verification pattern.

AES keys use the SHA-256 algorithm as the default algorithm. An 8-byte verification pattern is generated for the key specified. The random number parameter is not used.

The optional ENC-ZERO algorithm can be used with any key. A 4-byte verification pattern is generated for non-compliant-tagged tokens. A 3-byte verification pattern is generated for compliant-tagged tokens. The random number parameter is not used.

CSNBKYT is consistent with the IBM 4767 PCIe Cryptographic Coprocessor verb of the same name. If you generate a key on the IBM 4767 PCIe Cryptographic Coprocessor, you can verify it on ICSF and vice versa.

See "Key Test Extended (CSNBKYTX and CSNEKYTX)" on page 267 to verify the value of a DES key encrypted using a KEK.

The callable service name for AMODE(64) invocation is CSNEKYT.

Format

```
CALL CSNBKYT(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier,
             random_number,
             verification_pattern)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value can be 2, 3 or 4.

rule_array

Direction	Type
Input	String

Keywords provide control information to the callable service. Table 100 on page 253 lists the keywords. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 100. Keywords for Key Test Control Information

Keyword	Meaning
Key or key part rule (one keyword required)	
CLR-A128	Process a 128-bit AES clear key.
CLR-A192	Process a 192-bit AES clear key.
CLR-A256	Process a 256-bit AES clear key.
KEY-CLR	Specifies the DES key supplied in *key_identifier* is a single-length clear key.
KEY-CLRD	Specifies the DES key supplied in *key_identifier* is a double-length clear key.
KEY-CLRT	Specifies the DES key supplied in *key_identifier* is a triple-length clear key. ENC-ZERO is the only valid verification process rule.
KEY-ENC	Specifies the DES key supplied in *key_identifier* is a single-length clear or encrypted key in a fixed-length key token or label.

Table 100. Keywords for Key Test Control Information (continued)	
Keyword	**Meaning**
KEY-ENCD	Specifies the DES key supplied in *key_identifier* is a double-length clear or encrypted key in a fixed-length key token or label.
KEY-ENCT	Specifies the DES key supplied in *key_identifier* is a triple-length encrypted key in a fixed-length key token. ENC-ZERO is the only valid verification process rule.
TOKEN	Process an AES clear or encrypted key token or label.
Process Rule (one keyword required)	
GENERATE	Generate a verification pattern for the key supplied in *key_identifier*.
VERIFY	Verify a verification pattern for the key supplied in *key_identifier*.
Parity Adjustment - cannot be specified with any of the AES keywords (optional)	
ADJUST	Adjust the parity of test key to odd prior to generating or verifying the verification pattern. The *key_identifier* field itself is not adjusted.
NOADJUST	Do not adjust the parity of test key to odd prior to generating or verifying the verification pattern. This is the default.
Verification Process Rule (optional) **Note:** If this keyword is not specified and a single-length or double-length DES key is supplied, the verification pattern is calculated using the IBM algorithm. For more information, see "Key test verification pattern algorithms" on page 1654.	
ENC-ZERO	ENC-ZERO can be used with any of the rules. Required for triple-length DES keys.
SHA-256	Use the 'SHA-256' method. Use with CLR-A128, CLR-A192, CLR-A256, and TOKEN. SHA-256 is also the default for the AES rules.

key_identifier

Direction	Type
Input/Output	String

The key for which to generate or verify the verification pattern. The parameter is a 64-byte string of an internal token, key label, or a clear key value left-justified.

Note: If you supply a key label for this parameter, it must be unique on the CKDS.

When a key token is supplied and the key was encrypted under the old master key, the token is returned encrypted under the current master key.

random_number

Direction	Type
Input/Output	String

This is an 8-byte field that contains a random number supplied as input for the test pattern verification process and returned as output with the test pattern generation process. *random_number* is only used with the default algorithm for DES operational keys.

verification_pattern

Direction	Type
Input/Output	String

This is an 8-byte field that contains a verification pattern supplied as input for the test pattern verification process and returned as output with the test pattern generation process.

Restrictions

Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.

This callable service does not support external key tokens.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms, that is, clear, operational or external.

The parity of DES keys is not tested.

There is support for the generation and verification of single-length, double-length, and triple-length keys for the ENC-ZERO verification process. For triple-length DATA keys, use KEY-ENCD or KEY-ENCT with ENC-ZERO.

Access control points

The access control point in the domain role that controls the function of this service is **Key Test and Key Test 2**. This access control point cannot be disabled. It is required for ICSF master key validation.

If the access control point **Key Test - For encrypted DES keys, warn when keyword inconsistent with DES key length** is enabled, a warning will be generated if the Key Rule specified does not match the *key_identifier* provided.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 101. Key Test required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys and keyword KEY-ENCT require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.

Table 101. Key Test required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys and keyword KEY-ENCT require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys and keyword KEY-ENCT require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Key Test2 (CSNBKYT2 and CSNEKYT2)

Use this callable service to generate or verify a secure, cryptographic verification pattern (also referred to as a key check value) for AES, DES and HMAC keys. The key to test can be in the clear, encrypted under the master key, or encrypted under a key-encrypting key. Keywords in the *rule_array* specify whether the callable service generates or verifies a verification pattern.

For AES, CCA key tokens may be either internal, fixed-length (version 04) tokens, or external and internal variable-length (version 05) tokens, or an AES clear key part. X9.143 (TR-31) key blocks may be either internal or external (algorithm A).

For DES, CCA key tokens may be external and internal, fixed-length (versions 00 or 01) tokens, or external variable-length tokens with a DESUSECV key, or an DES clear key part. X9.143 key blocks may be either internal or external (algorithm D or T).

For HMAC, CCA key tokens are external and internal variable-length (version 05) key tokens. X9.143 key blocks may be either internal or external (algorithm H).

Note: Internal X9.143 key block and external key blocks with a key context of 1, require the CCA release 8.1 and later licensed internal code. External key blocks with a key context of 0 are supported by all CCA releases available.

Clear key tokens are not supported.

This service returns the key length of an AES or DES key in a secure key token or X9.143 (TR-31) key block. The information is returned in the *verification_pattern* parameter.

The callable service name for AMODE(64) invocation is CSNEKYT2.

Format

```
CALL CSNBKYT2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              key_encrypting_key_identifier_length,
              key_encrypting_key_identifier,
              reserved_length,
              reserved,
              verification_pattern_length,
              verification_pattern )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 2, 3, 4, or 5.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 102. Keywords for Key Test2 Control Information

Keyword	Meaning
Token algorithm (Required)	
AES	Specifies the key supplied in the *key_identifier* parameter is an AES key in an internal or external CCA key token, a TR-31 key block, or an AES clear key part.
DES	Specifies the key supplied in the *key_identifier* parameter is a DES key in an internal or external CCA key token, a TR-31 key block, a DES key in a DESUSECV token, or a DES clear key part.
HMAC	Specifies the key supplied in the *key_identifier* parameter is an internal or external HMAC key in a CCA key token or TR-31 key block.
Process rule (One required)	
GENERATE	Generate a verification pattern for the specified key.
VERIFY	Verify that a verification pattern matches the specified key.
KEY-LEN	Return the length of the specified key in the *verification_pattern* parameter. Valid for token algorithm AES and DES.
Verification pattern calculation algorithm (Optional). Not valid with KEY-LEN process rule.	
CMACZERO	Specifies that the verification pattern for AES and DES keys will be calculated by computing a MAC upon a data block of 0x00 bytes using the CMAC algorithm.
ENC-ZERO	Specifies that the verification pattern for AES and DES keys will be calculated by encrypting a data block filled with 0x00 bytes using the ECB mode. This is the default method for DES.
SHA-256	Specifies that the verification pattern will be calculated for an AES token using the same method as the Key Test service with the SHA-256 rule. This is the default method for AES. This rule can be used to verify that the same key value is present in a version 4 DATA token and version 5 AES CIPHER token or to verify that the same key value is present in a version 5 AES complementary key pair.
SHA2VP1	Specifies that the verification pattern will be calculated using the SHA-256 algorithm. For more information, see "SHAVP1 algorithm" on page 1654. This is the default and only method available for HMAC.
Token type rule (One required if a TR-31 key block, a DESUSECV token, or a DES or AES clear key part is passed; not valid otherwise).	

Table 102. Keywords for Key Test2 Control Information (continued)	
Keyword	**Meaning**
TR-31	Specifies that *key_identifier* contains a TR-31 key block. When an external DES wrapped key block (key block protection method VARXOR-A, VARDRV-B, and VARXOR-C) is supplied, the DES key-encrypting key must be supplied. When an external AES wrapped key block (VARDRV-D) is supplied, the AES key-encrypting key must be supplied.
KEY-CLR	Process a clear key or clear key-part that is not in a token. The algorithm is given by the token algorithm keyword group and is only valid with AES or DES. The length is given in the *key_identifier_length* parameter. Not valid with KEY-LEN or KEK identifier rules.
AESKWCV	Specifies that *key_identifier* contains an external variable length symmetric key token whose type is DESUSECV. The IKEK-AES keyword must be specified for the KEK identifier rule.
KEK identifier rules (One required if the AESKWCV token type is specified)	
IKEK-AES	The wrapping KEK for the key to test is an AES KEK. This is the default for AES and HMAC token algorithms.
IKEK-DES	The wrapping KEK for the key to test is a DES KEK. This is the default for DES token algorithm, and is only allowed with the DES token algorithm.
IKEK-PKA	The wrapping KEK for the key to test is an RSA or (other key stored in PKA key storage.) This is not the default for any token algorithm and must be specified if an RSA KEK is used. This rule is not allowed with DES token algorithm.
Verification pattern length rule (Optional). May only be specified with rules DES and CMACZERO.	
VPLEN3	Specifies that a 3-byte verification pattern will be calculated. This is the default.
VPLEN5	Specifies that a 5-byte verification pattern will be calculated.

key_identifier_length

Direction	Type
Input	Integer

The length of the key_identifier in bytes. When the *key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The key for which to generate or verify the verification pattern. The key identifier is a variable-length key token or key block or the label of an operational token or block in key storage, or an AES or DES clear key part.

Table 103. AES keys and input key description

AES keys	Input key description
CCA fixed-length token version X'04'	Internal AES key wrapped with the AES master-key.
CCA variable-length token version X'05'	External AES key wrapped with an CCA AES key-encrypting key using the AESKW key-formatting method.
	External AES key wrapped with a TR-31 AES key-encrypting key using the AESKW key-formatting method. (CCA Release 8.1 or later).
	External AES key wrapped with an CCA RSA public-key using the PKOAEP2 key-wrapping method.
	Internal AES key that is wrapped with the AES master-key using the AESKW key-formatting method.
X9.143 (TR-31) key block (key block version ID 'D')	External AES key wrapped with a CCA AES key-encrypting key using a TR-31 key-wrapping method.
	External AES key wrapped with a TR-31 AES key-encrypting key using the TR-31 key-wrapping method. (CCA Release 8.1 or later).
	Internal AES key wrapped with an AES master key using the TR-31 key-wrapping method. (CCA Release 8.1 or later).

Table 104. DES keys and input key description

DES keys	Input key description
CCA fixed-length token version X'00' and X'01'	External DES key wrapped with a CCA DES key-encrypting key using any CCA wrapping method.
	External DES key wrapped with a TR-31 DES key-encrypting key using any CCA wrapping method. (CCA Release 8.1 or later).
	Internal DES key wrapped with the DES master-key using any wrapping method.

Table 104. DES keys and input key description (continued)

DES keys	Input key description
CCA variable-length token version X'05'	External DES key with key type DESUSECV wrapped with an CCA AES key-encrypting key using the AESKW key-formatting method. **Note:** A key token with a key type of DESUSECV contains the control vector and other information that is needed to re-create the original fixed-length DES key-token. A DESUSECV key can only be created using the CSNDSYX service.
	External DES key with key type DESUSECV wrapped with a TR-31 AES key-encrypting key using the AESKW key-formatting method. (CCA Release 8.1 or later). **Note:** A key token with a key type of DESUSECV contains the control vector and other information that is needed to re-create the original fixed-length DES key-token. A DESUSECV key can only be created using the CSNDSYX service.
X9.143 (TR-31) key block (key block version ID 'A', 'B', 'C')	External DES key wrapped with a CCA DES key-encrypting key using a TR-31 key-wrapping method.
	External DES key wrapped with a TR-31 DES key-encrypting key using a TR-31 key-wrapping method. (CCA Release 8.1 or later).
	Internal DES key wrapped with the DES master-key using TR-31 key-wrapping method "B". (CCA Release 8.1 or later).
X9.143 (TR-31) key block (key block version ID 'D')	External DES key wrapped with a CCA AES key-encrypting key using the TR-31 key-wrapping method.
	External DES key wrapped with a TR-31 AES key-encrypting key using the TR-31 key-wrapping method. (CCA Release 8.1 or later)
	Internal DES key wrapped with the AES master-key using the TR-31 key-wrapping method. (CCA Release 8.1 or later)

Table 105. HMAC keys and input key description	
HMAC keys	**Input key description**
CCA variable-length token version X'05'	External HMAC key wrapped with a CCA AES key-encrypting key using the AESKW key-formatting method.
	External HMAC key wrapped with a TR-31 AES key-encrypting key using the AESKW key-formatting method. (CCA Release 8.1 or later)
	External HMAC key wrapped with an RSA public-key using the PKOAEP2 key-wrapping method.
	Internal HMAC key wrapped with the AES master-key using the AESKW key-formatting method.
X9.143 (TR-31) variable-length key block (key block version ID 'D')	External HMAC key wrapped with an CCA AES key-encrypting key using the TR-31 key-wrapping method.
	External HMAC key wrapped with a TR-31 AES key-encrypting key using the TR-31 key-wrapping method. (CCA Release 8.1 or later)
	Internal HMAC key wrapped with an AES master-key using the TR-31 key-wrapping method. (CCA Release 8.1 or later)

Clear key tokens are not supported.

If an internal token or key block was supplied and was encrypted under the old master key, the token or key block will be returned encrypted under the current master key.

key_encrypting_key_identifier_length

Direction	**Type**
Input	Integer

The length of the *key_encrypting_key_identifier* parameter. When *key_identifier* is an internal token, the value must be zero.

If *key_encrypting_key_identifier* is a label for either the CKDS (IKEK-AES or IKEK-DES rules) or PKDS (IKEK-PKA rule), the value must be 64.

If *key_encrypting_key_identifier* is an AES KEK, the value must be between the actual length of the token and 9992.

If *key_encrypting_key_identifier* is a DES KEK, the value must be between the actual length of the token and 9992.

If *key_encrypting_key_identifier* is an RSA KEK, the maximum length is 3500.

key_encrypting_key_identifier

Direction	**Type**
Input/Output	String

The identifier of the key-encrypting key to unwrap the key to be processed. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

When *key_encrypting_key_identifier_length* is zero, this parameter is ignored.

For CCA symmetric keys, the identifier is a 64-byte DES key token of a key-encrypting key or a variable-length AES key token of a key-encrypting key. For CCA asymmetric keys, the identifier is a variable-length key token of an RSA private key.

For X9.143 key, the identifier is a variable-length token of a DES or AES key-encrypting key matching the algorithm of the key to be unwrapped: key usage K0 or K1, algorithm T or A, and mode of use D or E).

If the key identifier supplied was an AES or DES token or key block encrypted under the old master key, the token or key block will be returned encrypted under the current master key.

reserved_length

Direction	Type
Input	Integer

The length of the reserved parameter. The value must be zero.

reserved

Direction	Type
Input/Output	String

This parameter is ignored.

verification_pattern_length

Direction	Type
Input/Output	Integer

The length of the *verification_pattern* parameter in bytes.

Table 106. Length of the verification pattern for each algorithm or process rule supported

Calculation algorithm or process rule	Length of the verification pattern in bytes
CMACZERO	AES: 5 DES: 3 or 5
ENC-ZERO	Non-compliant-tagged tokens: 4 Compliant-tagged tokens: 3 The verification pattern is left-justified in an 8-byte field and padded on the right with zeros.
SHA-256	8
SHA2VP1	8
KEY-LEN (process rule)	AES: 2 DES: 4

On input for GENERATE, the value should be the size of the buffer for the *verification_pattern* parameter.

On input for VERIFY, the length must be the length of the verification pattern supplied in the *verification_pattern* parameter.

On output, the parameter will be updated for the length of the *verification_pattern* returned.

verification_pattern

Direction	Type
Input/Output	String

For GENERATE, the verification pattern generated for the key.

For VERIFY, the supplied verification pattern to be verified.

For KEY-LEN, the length of the key in the supplied *key_identifier* as shown in the following table.

Table 107. Returned data for KEY-LEN process rule

Token algorithm	Data format
DES	4 bytes: 1. Stored key bit length: 2 byte binary integer. 2. Effective key bit length: 2 byte binary integer. Possible values: 56 (X'0038'), 112 (X'0070'), 168 (X'00A8').
AES	2 bytes: 1. Stored key bit length: 2 byte binary integer. Possible values: 128 (X'0080'), 192 (X'00C0'), 256 (X'0100').

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms: clear, operational or external.

Access control point

The access control point in the domain role that controls the function of this service is **Key Test and Key Test 2**. This access control point cannot be disabled. It is required for ICSF master key validation.

Some of the verification pattern calculation algorithm or process rule keywords require an additional access control point to be enabled. Keywords not in the table are always available.

Table 108. Required access control points for Key Test2

Rule array keyword	Key algorithm	Access control point
CMACZERO	AES	Key Test2 - AES, CMACZERO
CMACZERO	DES	Key Test2 - DES, CMACZERO
ENC-ZERO	AES	Key Test2 - AES, CMACZERO
KEY-LEN	AES	Key Test2 - AES, KEY-LEN
KEY-LEN	DES	Key Test2 - DES, KEY-LEN

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 109. Key Test2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	The CMACZERO keyword requires the March 2016 or later licensed internal code (LIC). The VPLEN3 and VPLEN5 keywords are not supported. Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. TR-31 key block containing an HMAC key requires the June 2020 or later licensed internal code. TR-31 key block containing an AES key with key usage 'B1' is not supported. Rule array keyword KEY-LEN requires the May 2021 or later licensed internal code (LIC). Keyword KEY-CLR requires the CCA release 5.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 109. Key Test2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	The CMACZERO keyword requires the March 2016 or later licensed internal code (LIC). The VPLEN3 and VPLEN5 keywords are not supported. Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. TR-31 key block containing an HMAC key requires the June 2020 or later licensed internal code. TR-31 key block containing an AES key with key usage 'B1' is not supported. Rule array keyword KEY-LEN requires the May 2021 or later licensed internal code (LIC). Keyword KEY-CLR requires the CCA release 5.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). TR-31 key block containing an HMAC key requires the June 2020 or later licensed internal code. TR-31 key block containing an AES key with key usage 'B1' is not supported. Rule array keyword KEY-LEN requires the May 2021 or later licensed internal code (LIC). Keyword KEY-CLR requires the CCA release 6.6 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 109. Key Test2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The VPLEN3 and VPLEN5 keywords are not supported. TR-31 key block containing an HMAC key requires the June 2020 or later licensed internal code. TR-31 key block containing an AES key with key usage 'B1' is not supported. Rule array keyword KEY-LEN requires the May 2021 or later licensed internal code (LIC). Keyword KEY-CLR requires the CCA release 5.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	TR-31 key block containing an HMAC key requires the June 2020 or later licensed internal code. TR-31 key block containing an AES key with key usage 'B1' requires the September 2020 or later licensed internal code (LIC). Rule array keyword KEY-LEN requires the May 2021 or later licensed internal code (LIC). Keyword KEY-CLR requires the CCA release 6.6 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keyword KEY-LEN requires the May 2021 or later licensed internal code (LIC). Keyword KEY-CLR requires the CCA release 7.3 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Key Test Extended (CSNBKYTX and CSNEKYTX)

Use the Key Test Extended service to generate or verify a secure, cryptographic verification pattern for DES keys. The key to test is encrypted under a key-encrypting key (KEK). AES keys are not supported by this service. Keywords in the rule array specify whether the callable service generates or verifies a verification pattern.

The default algorithm supports single-length and double-length keys. Single-length, double-length, and triple-length keys are supported with the ENC-ZERO algorithm.

When the service generates a verification pattern, it creates and cryptographically processes a random number. The service returns the random number with the verification pattern.

When the service tests a verification pattern against a key, you must supply the random number and the verification pattern from a previous call to Key Test Extended. The service returns the verification result in the return and reason codes.

The callable service name for AMODE(64) invocation is CSNEKYTX.

Format

```
CALL CSNBKYTX(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               rule_array_count,
               rule_array,
               key_identifier,
               random_number,
               verification_pattern,
               KEK_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value can be 2, 3 or 4.

rule_array

Direction	Type
Input	String

Two or three keywords that provide control information to the callable service. Table 110 on page 269 lists the keywords. The keywords must be in 16 or 24 bytes of contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 110. Keywords for Key Test Extended Control Information

Keyword	Meaning
Key Rule (required)	
KEY-ENC	Specifies the key supplied in *key_identifier* is a single-length encrypted DES key.
KEY-ENCD	Specifies the key supplied in *key_identifier* is a double-length encrypted DES key.
KEY-ENCT	Specifies the key supplied in *key_identifier* is a triple-length encrypted DES key. ENC-ZERO is the only valid verification process rule.
Process Rule (required)	
GENERATE	Generate a verification pattern for the key supplied in *key_identifier*.
VERIFY	Verify a verification pattern for the key supplied in *key_identifier*.
Parity Adjustment (optional)	
ADJUST	Adjust the parity of test key to odd prior to generating or verifying the verification pattern. The *key_identifier* field itself is not adjusted.
NOADJUST	Do not adjust the parity of test key to odd prior to generating or verifying the verification pattern. This is the default.
Verification Process Rule (optional) **Note:** If this keyword is not specified and a DES key is supplied, the verification pattern is calculated using the IBM algorithm. For more information, see "Key test verification pattern algorithms" on page 1654.	
ENC-ZERO	Specifies use of the "encrypted zeros" method. A 4-byte verification pattern is generated for non-compliant-tagged tokens. A 3-byte verification pattern is generated for compliant-tagged tokens. Required for triple-length DES keys.

key_identifier

Direction	Type
Input/Output	String

The key for which to generate or verify the verification pattern. The parameter is a 64-byte string of an internal token or key label that is left-justified.

Note: If you supply a key label for this parameter, it must be unique on the CKDS.

random_number

Direction	Type
Input/Output	String

This is an 8-byte field that contains a random number supplied as input for the test pattern verification process and returned as output with the test pattern generation process.

verification_pattern

Direction	Type
Input/Output	String

This is an 8-byte field that contains a verification pattern supplied as input for the test pattern verification process and returned as output with the test pattern generation process.

KEK_key_identifier

Direction	Type
Input/Output	String

If *key_identifier* is an external token, then this is a 64-byte string of an internal token or a key label of an IMPORTER or EXPORTER used to encrypt the test key. If *key_identifier* is an internal token, then the parameter is ignored.

Note: If you supply a key label for this parameter, it must be unique on the CKDS.

Restrictions

Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms, that is, clear, operational or external.

The parity of DES keys is not tested.

When using the ENC-ZERO verification rule, there is support for enciphered single-length, double-length, and triple-length DES keys.

Access control point

The access control point in the domain role that controls the function of this service is **Key Test and Key Test 2**. This access control point cannot be disabled. It is required for ICSF master key validation.

If the access control point **Key Test - For encrypted DES keys, warn when keyword inconsistent with DES key length** is enabled, a warning is generated if the Key Rule specified does not match the *key_identifier* provided.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 111. Key Test Extended required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys and keyword KEY-ENCT require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys and keyword KEY-ENCT require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys and keyword KEY-ENCT require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Key Token Build (CSNBKTB and CSNEKTB)

Use the Key Token Build callable service to build an external or internal fixed-length symmetric key token from information which you supply. To build a variable-length symmetric key-token, see "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297.

The token can be used as input for the key generate and key part import callable services. You can specify a control vector or the service can build a control vector based upon the key type you specify and the control vector-related keywords in the rule array. ICSF supports the building of an internal key token with the key encrypted under a master key other than the current master key and building internal clear AES and DES tokens.

The callable service name for AMODE(64) invocation is CSNEKTB.

Format

```
CALL CSNBKTB(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_token,
             key_type,
             rule_array_count,
             rule_array,
             key_value,
             master_key_version_number,
             key_register_number,
             token_data_1,
             control_vector,
             initialization_vector,
             pad_character,
             cryptographic_period_start,
             master_key_verification_pattern)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

key_token

Direction	Type
Output	String

The 64-byte fixed-length key token built.

key_type

Direction	Type
Input	String

An 8-byte field that specifies the type of key you want to build. See "DES key types" on page 23 and "AES key types" on page 26 for the services that use the key types. See "Clear keys" on page 28 for the services that use the CLRAES and CLRDES key types.

These are the valid *key_type* keywords for AES:

Table 112. Valid key_type keywords for AES

Key type	Description
CLRAES	The *key_token* parameter is a clear AES DATA key token. The *rule_array* must contain the keyword INTERNAL and one of the optional keywords: KEYLN16, KEYLN24, or KEYLN32. A key value parameter must also be provided.
DATA	The *rule_array* keyword AES must be specified to build an encrypted AES key token.

These are the valid *key_type* keywords for DES:

Table 113. Valid key_type keywords for DES

Key type	Description
CLRDES	The *key_token* parameter is a clear DES DATA key token. The *rule_array* must contain the keyword INTERNAL and one of the optional keywords: KEYLN8, KEYLN16, or KEYLN24. A key value parameter must also be provided.
DATA-CV	The key type is DATA and the default DATA control vector will be used. The control vector can be modified by rule array keywords as described in Table 22 on page 120.
KEYGENKY	CLR8-ENC or UKPT must be coded in the *rule_array* parameter.
SECMSG	SMKEY or SMPIN must be specified in the *rule_array* parameter.

Table 113. Valid key_type keywords for DES (continued)

Key type	Description
USE-CV	A user-supplied control vector, supplied in the *control_vector* parameter, is used to build the token. The CV rule array keyword should be specified if USE-CV is specified. When the key type is USE-CV, control vector keywords in the rule array are ignored. The number of bytes of the control vector copied into the output key token depends on the DES key length keyword specified in the rule array: • If no keyword is specified, 16 bytes are copied. • If KEYLN8 or SINGLE is specified, 8 bytes are copied. • If KEYLN16, DOUBLE, or DOUBLE-0 is specified, 16 bytes are copied. • If KEYLN24, TRIPLE, or TRIPLE-O is specified, 16 bytes are copied. A DES key wrapping method keyword may be required to match the CCA control vector and key length specified. When the KEY keyword is specified, the default length is 16 bytes. The key length keywords for DES keys are used to change the length to 8 or 24.
CIPHER CIPHERXI CIPHERXL CIPHERXO CVARDEC CVARENC CVARPINE CVARXCVL CVARXCVR DATA DATAC DATAM DATAMV DECIPHER DKYGENKY ENCIPHER EXPORTER IKEYXLAT IMPORTER IPINENC MAC MACVER OKEYXLAT OPINENC PINGEN PINVER	The default control vector for the key type will be used. The control vector can be modified by rule array keywords as described in Table 22 on page 120.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. See Table 114 on page 275 for a list. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 114. Keywords for Key Token Build Control Information

Keyword	Meaning	Algorithm
Token Algorithm (optional - zero or one keyword)		
AES	Specifies that an AES key token will be built. This keyword is required when building an encrypted AES token. It is optional when using the CLRAES key type to build a clear AES DATA key token.	AES
DES	Specifies a DES token will be built.	DES
SYS-ENC	Tolerated for compatibility reasons.	DES
Token Type (one keyword required)		
EXTERNAL	Specifies that an external key token will be built.	DES
INTERNAL	Specifies that an internal key token will be built.	AES or DES
Key Status (optional - zero or one keyword)		
KEY	This keyword indicates that the key token to build will contain an encrypted key. The *key_value* parameter identifies the field that contains the key.	AES or DES
NO-KEY	This keyword indicates that the key token to build will not contain a key. This is the default key status.	AES or DES
Key Length (one keyword required for AES keys): See Table 116 on page 280 for valid key types for these key length values.		
KEYLN8	Single-length or 8-byte key. Default for CLRDES.	DES
KEYLN16	Specifies that the key is 16-bytes long.	AES or DES
KEYLN24	Specifies that the key is 24-bytes long.	AES
KEYLN24	Specifies that the key is 24-bytes long. Valid with CLRDES and DATA key type only.	DES
KEYLN32	Specifies that the key is 32-bytes long.	AES
DOUBLE	Double-length or 16-byte key. Synonymous with KEYLN16. Not valid for CLRDES.	DES
DOUBLE-O	Double-length key with guaranteed unique key values. The key is 16 bytes long. This key length can be used with any key type that supports DOUBLE.	DES

Table 114. Keywords for Key Token Build Control Information (continued)		
Keyword	**Meaning**	**Algorithm**
MIXED	Double-length key. Indicates that the key can either be a replicated single-length key or a double-length key with two different 8–byte values. Not valid for CLRDES.	DES
SINGLE	Single-length or 8-byte key. Synonymous with KEYLN8. Not valid for CLRDES.	DES
TRIPLE	Triple-length or 24-byte key. Not valid for CLRDES.	DES
TRIPLE-O	Triple-length key with guaranteed unique key values. The key is 24 bytes long. This key length can be used with any key type that supports TRIPLE. Not valid for CLRDES.	DES
Key Part Indicator (optional) — not valid for CLRDES		
KEY-PART	This token is to be used as input to the key part import service.	DES
Control vector (CV) source (optional - zero or one of these keywords is permitted)		
CV	This specifies that the key token should be built using the control_vector supplied in the *control_vector* parameter. **Note:** When this keyword is specified, all control vector related keywords in the rule array are ignored.	DES
NO-CV	This specifies that the key token should be built using a control vector that is based on the supplied key type control vector related rule array keywords. It is the default.	DES
Control vector on the link specification (optional) — valid only for IMPORTER and EXPORTER.		
CV-KEK	This keyword indicates marking the KEK as a CV KEK. The control vector is applied to the KEK prior to using it in encrypting other keys. This is the default.	DES
NOCV-KEK	This keyword indicates marking the KEK as a NOCV KEK. The control vector is not applied to the KEK prior to its use in encrypting other keys.	DES
Control vector keywords (optional - zero or more of these keywords are permitted)		
See Table 116 on page 280 for the key-usage and key-management keywords that can be specified for a given key type. See Table 22 on page 120 for a description of the keywords and their usage.		DES
Master Key Verification Pattern (optional) — not valid for CLRDES or CLRAES keywords		

Table 114. Keywords for Key Token Build Control Information (continued)		
Keyword	**Meaning**	**Algorithm**
MKVP	This keyword indicates that the *key_value* is enciphered under the master key which corresponds to the master key verification pattern specified in the *master_key_verification_pattern* parameter. If this keyword is not specified, the output *key_token* will contain the current MKVP for the specified algorithm. The key contained in *key_value* must be enciphered under that same current master key or incorrect results will occur when the token is used in a cryptographic operation.	AES and DES
Key Wrapping Method (optional)		
WRAP-ENH	Use enhanced key wrapping method with SHA-1, which is compliant with the ANSI X9.24 standard.	DES
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens. This is the default.	DES
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only for TRIPLE or TRIPLE-O. This is the default for TRIPLE and TRIPLE-O.	DES
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.	DES
Translation Control (optional)		
ENH-ONLY	Restrict rewrapping of the token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method. Valid with WRAP-ENH, WRAPENH2, and WRAPENH3. This is the default for WRAPENH2 and WRAPENH3.	DES

key_value

Direction	**Type**
Input	String

If you use the KEY keyword, this parameter is a 16-byte string that contains the encrypted key value. Single-length keys must be left-justified in the field and padded on the right with X'00'. If you are building a triple-length key, this parameter is a 24-byte string containing the encrypted key value. You must specify a DES key length rule array keyword (TRIPLE, TRIPLE-O, or KEYLN24) to indicate that 24-bytes should be used.

If you supply an encrypted key value and also specify INTERNAL, the service will check for the presence of the MKVP keyword. If MKVP is present, the service will assume the *key_value* is enciphered under the master key which corresponds to the master key verification pattern specified in the *master_key_verification_pattern* parameter, and will place the key into the internal token along with the verification pattern from the *master_key_verification_pattern* parameter. If MKVP is not

specified, ICSF assumes the key is enciphered under the current host master key and places the key into an internal token along with the verification pattern for the current master key. In this case, the application must ensure that the master key has not changed since the key was generated or imported to this system. Otherwise, use of this parameter is not recommended.

For *key_type* CLRDES and CLRAES, this field is required to contain the clear key value. For KEYLN8, this is an 8-byte field. For KEYLN16, this is a 16-byte field. For KEYLN24, this a 24-byte field. For KEYLN32, this is a 32-byte field.

Table 115. Key types and field lengths for AES keys

Key type	Field length
AES-128 clear text key	16-bytes
AES-192 clear text key	24-bytes
AES-256 clear text key	32-bytes
AES-128, AES-192, AES-256 encrypted key	32-bytes

master_key_version_number

Direction	Type
Input	Integer

This field is examined only if the KEY keyword is specified, in which case, this field must be zero.

key_register_number

Direction	Type
Input	Integer

This field is ignored.

token_data_1

Direction	Type
Input	String

This parameter is ignored for DES keys.

This parameter is the LRC value for AES keys. For clear AES keys it is 8-bytes of X'00' indicating to the service that it must compute the LRC field value. For encrypted AES keys, you provide a 1-byte area containing the LRC value for the key passed in the *key_value* parameter. The service copies it into the LRC field of the key token.

control_vector

Direction	Type
Input	String

Specifies a 16-byte DES control vector. When the key type is USE-CV, this parameter is copied into the key token.

Note that the value in this parameter is not examined by ICSF. When used in conjunction with the KEY keyword, the default length is 16. You must use a DES key length rule array keyword for lengths 8 or 24.

This parameter is ignored for AES keys.

initialization_vector

Direction	Type
Input	String

This field is ignored.

pad_character

Direction	Type
Input	Integer

The only allowed value for key types MAC and MACVER is 0. This field is ignored for all other key types.

cryptographic_period_start

Direction	Type
Input	String

This field is ignored.

master_key_verification_pattern

Direction	Type
Input	String

8-byte verification pattern of the master key used to encrypt the key value. It is used when the KEY and INTERNAL *rule_array* keywords are specified. The value is inserted into the master key verification pattern field of the key token. If the KEY and INTERNAL keywords are specified in *rule_array*, the service will check for the existence of the MKVP rule array keyword. This parameter is ignored for any other *rule_array* keyword combinations.

Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

No pre- or post-processing or security exits are enabled for this service. No RACF checking is done, and no calls to RACF are issued when this service is used.

You can use this service to create skeleton key tokens with the desired data encryption algorithm bits for use in some key management services to override the default system specifications.

- To create an internal token with a specified KEY value, supply a valid master key verification pattern (MKVP).

This illustrates the key type, key usage, and key management keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector. See Table 22 on page 120 for a description of the key usage and key management keywords and their usage.

Table 116. Control Vector Generate and Key Token Build keyword combinations by DES key types		
Key class	**DES key type**	**Keyword combinations flowchart**
Cipher (data operation)	CIPHER	Figure 8 on page 281
	DECIPHER	Figure 9 on page 282
	ENCIPHER	
	CIPHERXI	Figure 10 on page 283
	CIPHERXL	
	CIPHERXO	
	DATA	Figure 11 on page 284
	DATAC	Figure 12 on page 285
	DATAM	
	DATAMV	
	MAC	Figure 13 on page 286
	MACVER	
	SECMSG	Figure 14 on page 287
PIN processing	IPINENC	Figure 15 on page 288
	OPINENC	Figure 16 on page 289
	PINGEN	Figure 17 on page 290
	PINVER	Figure 18 on page 291
Key encrypting key	EXPORTER	Figure 19 on page 292
	IMPORTER	Figure 20 on page 293
	IKEYXLAT	Figure 21 on page 294
	OKEYXLAT	
Key generating key	DKYGENKY	Figure 22 on page 295
	KEYGENKY	Figure 23 on page 296
Cryptographic variable encrypting key	CVARDEC	Figure 24 on page 297
	CVARENC	
	CVARPINE	
	CVARXCVL	
	CVARXCVR	

Figure 8 on page 281 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type CIPHER.

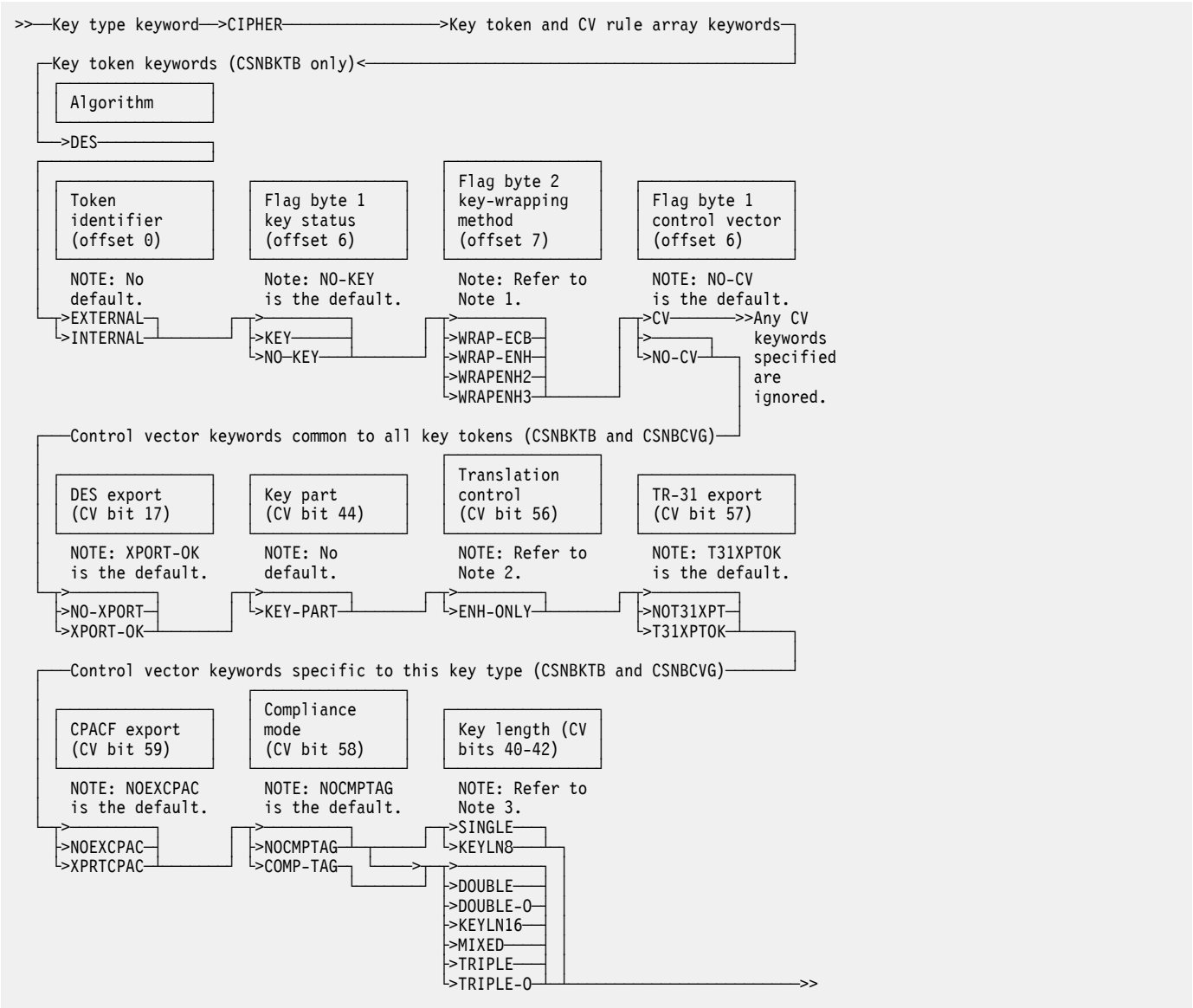


Figure 8. Keyword combinations for DES CIPHER keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.
2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. KEYLN8 is synonymous with SINGLE. KEYLN16 and MIXED are synonymous with DOUBLE. SINGLE and KEYLN8 are not valid when COMP-TAG is specified. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. DOUBLE is the default when COMP-TAG is specified. Otherwise, the default is SINGLE.

Figure 9 on page 282 shows the Control Vector Generate and Key Token Build keyword combinations for DES key types DECIPHER and ENCIPHER.

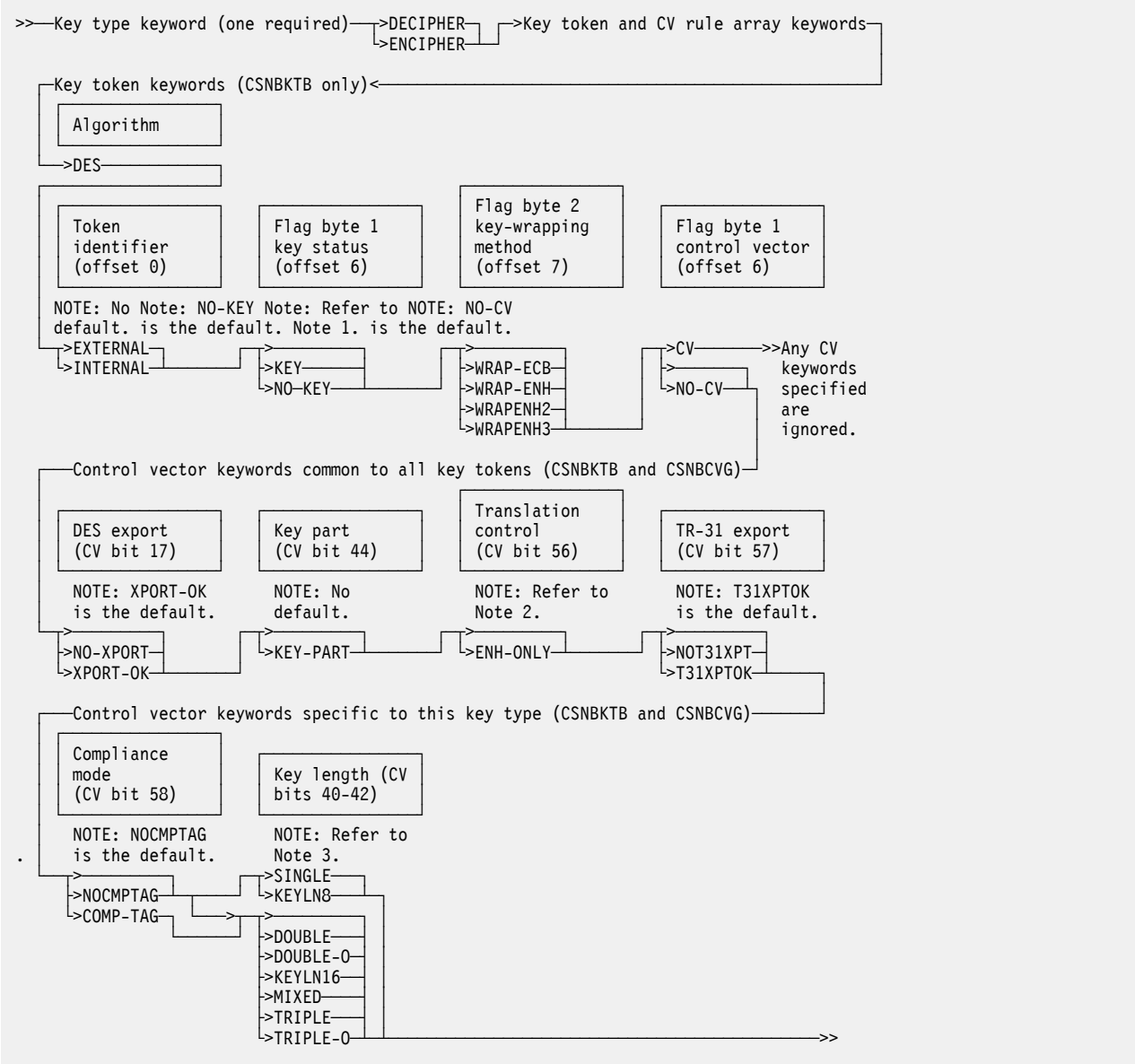


Figure 9. Keyword combinations for DES DECIPHER and ENCIPHER keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.
2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. KEYLN8 is synonymous with SINGLE. KEYLN16 and MIXED are synonymous with DOUBLE. SINGLE and KEYLN8 are not valid when COMP-TAG is specified. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. DOUBLE is the default when COMP-TAG is specified. Otherwise, the default is SINGLE.

Figure 10 on page 283 shows the Control Vector Generate and Key Token Build keyword combinations for DES key types CIPHERXI, CIPHERXL, and CIPHERXO.

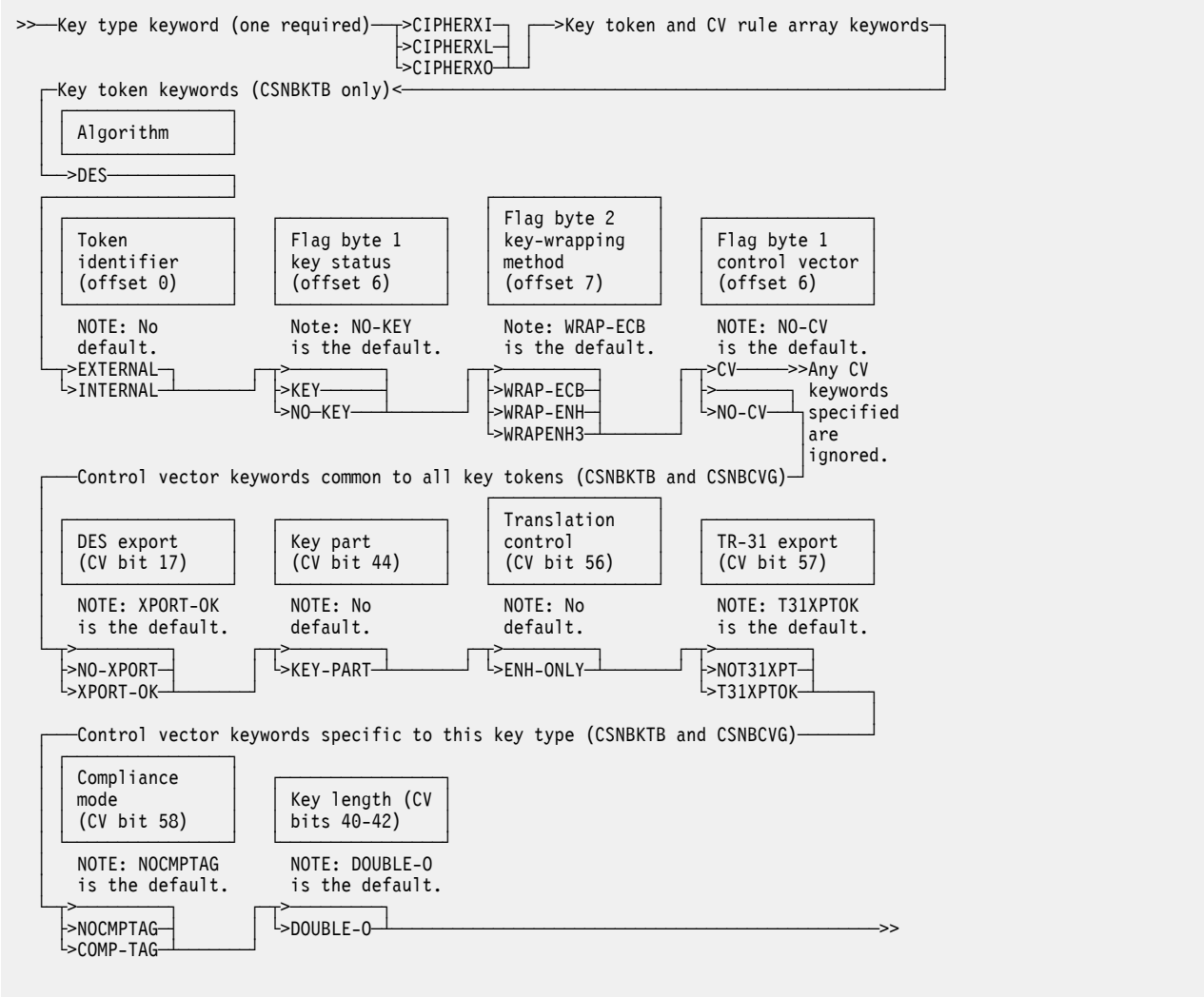


Figure 10. CSNBCVG and CSNBKTB keyword combinations for DES CIPHERXI, CIPHERXL, and CIPHERXO keys

Figure 11 on page 284 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type DATA.

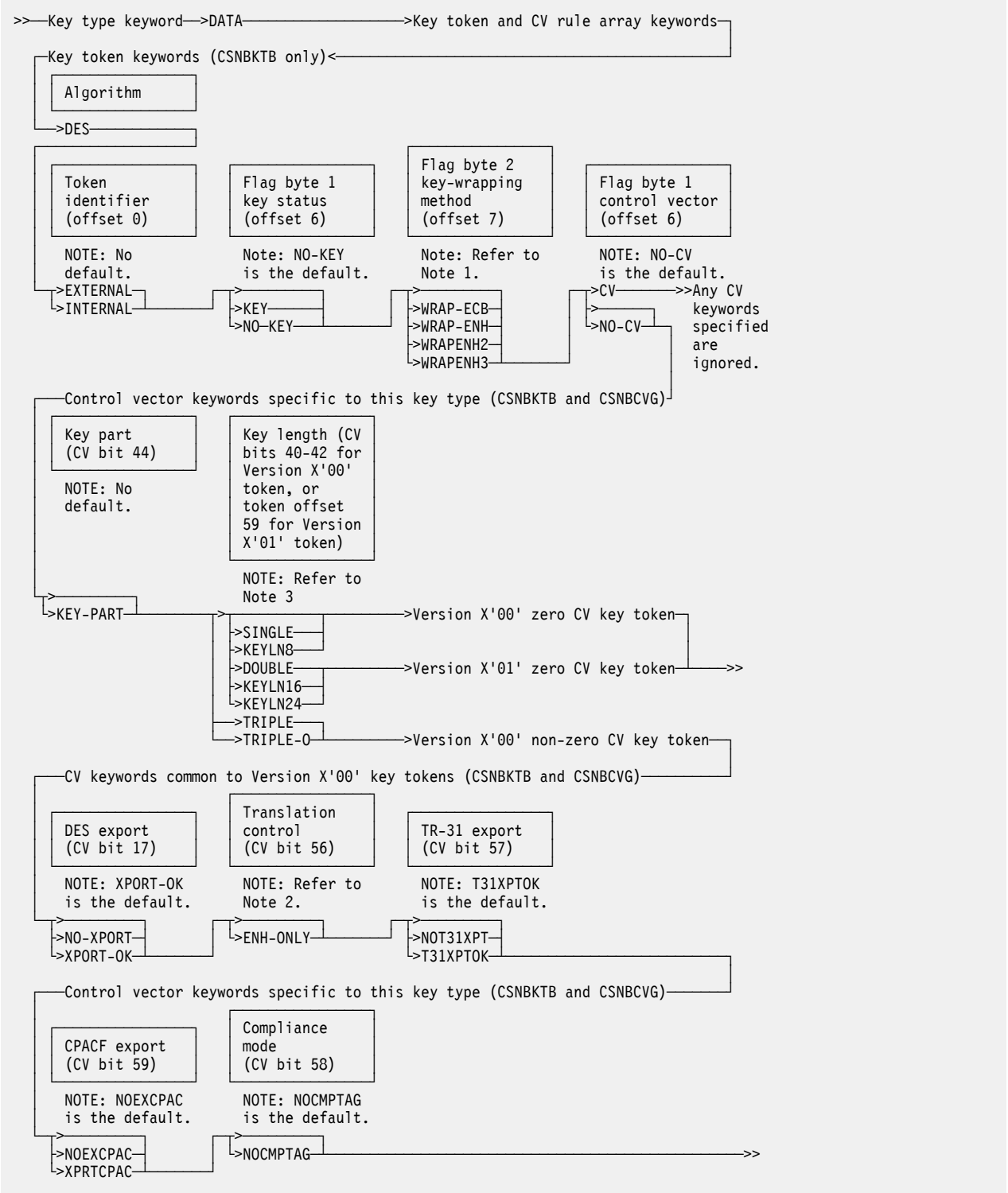


Figure 11. CSNBCVG and CSNBKTB keyword combinations for DES DATA keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.
2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. KEYLN8 is synonymous with SINGLE. KEYLN16 and MIXED are synonymous with DOUBLE. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. SINGLE is the default.

Figure 12 on page 285 shows the Control Vector Generate and Key Token Build keyword combinations for DES key types DATAC, DATAM, and DATAMV. These key types continue to only support two-key Triple-DES.

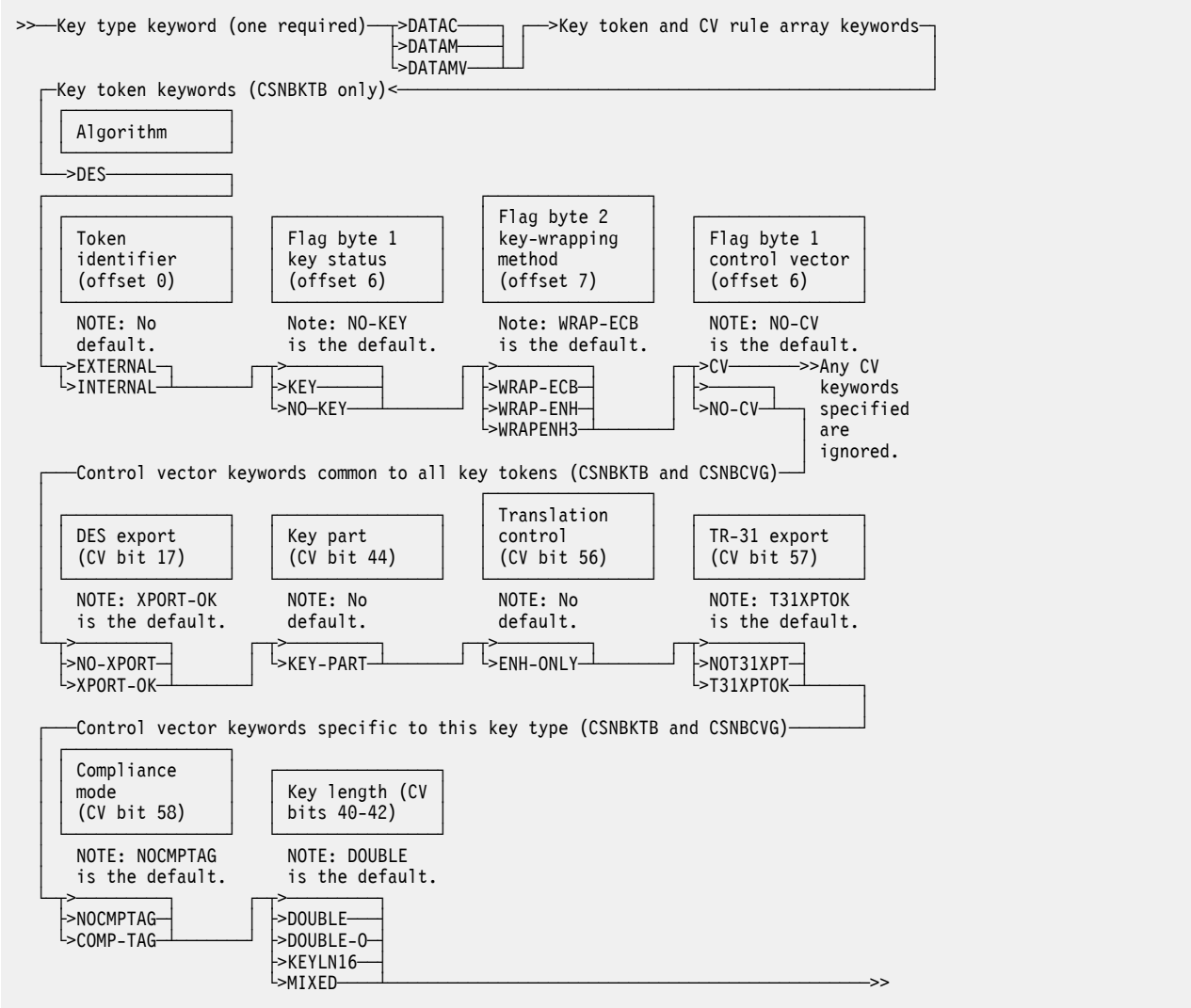


Figure 12. CSNBCVG and CSNBKTB keyword combinations for DES DATAC, DATAM, and DATAMV keys

Note: KEYLN16 and MIXED are synonymous with DOUBLE. DOUBLE is the default.

Figure 13 on page 286 shows the Control Vector Generate and Key Token Build, keyword combinations for DES key types MAC and MACVER.

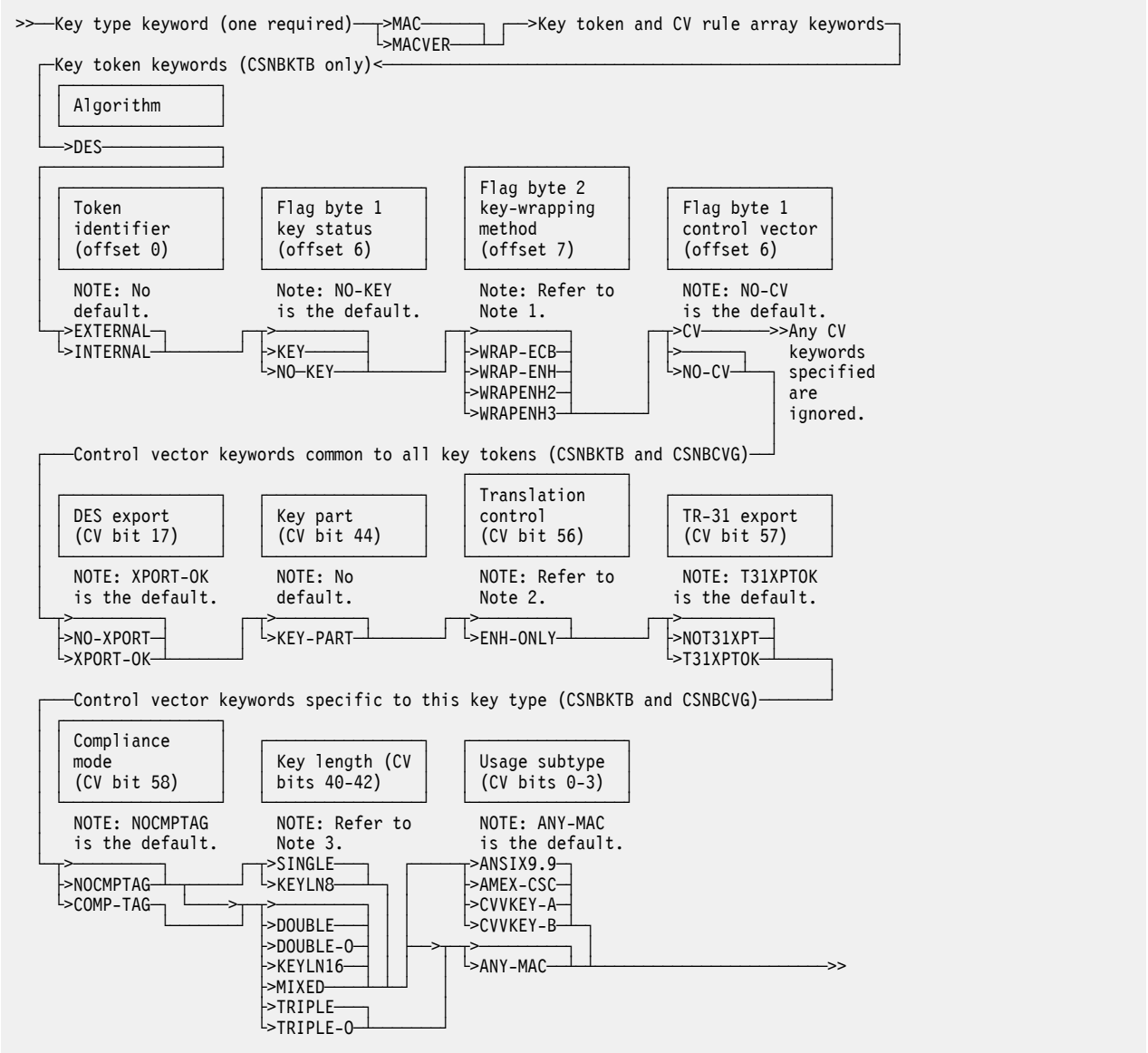


Figure 13. CSNBCVG and CSNBKTB keyword combinations for DES MAC and MACVER keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.
2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. KEYLN8 is synonymous with SINGLE. KEYLN16 and MIXED are synonymous with DOUBLE. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. SINGLE is the default.

Figure 14 on page 287 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type SECMSG. This key type does not support three-key Triple-DES.

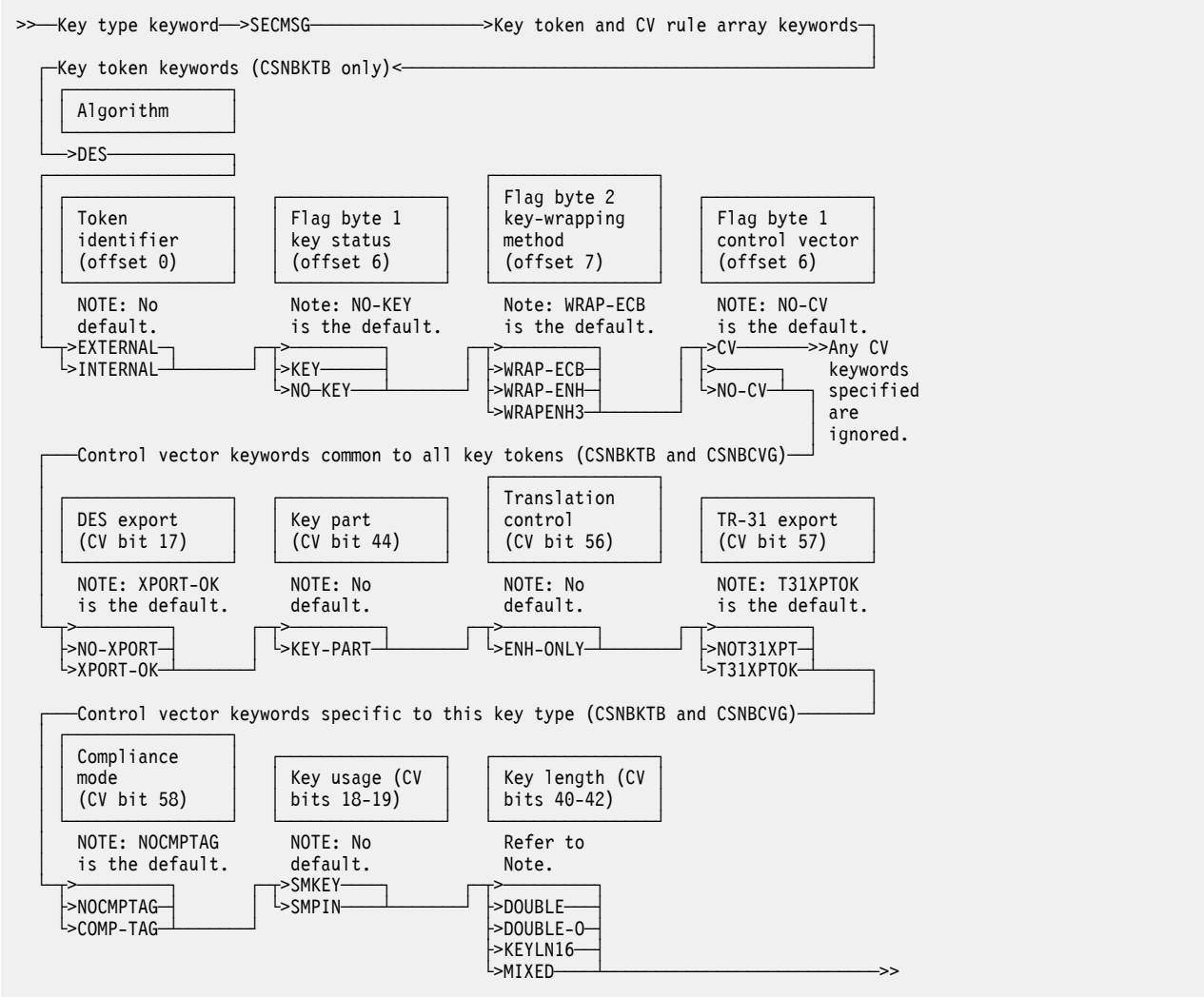


Figure 14. CSNBCVG and CSNBKTB keyword combinations for DES SECMSG keys

Note: KEYLN16 and MIXED are synonymous with DOUBLE. DOUBLE is the default.

Figure 15 on page 288 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type IPINENC.

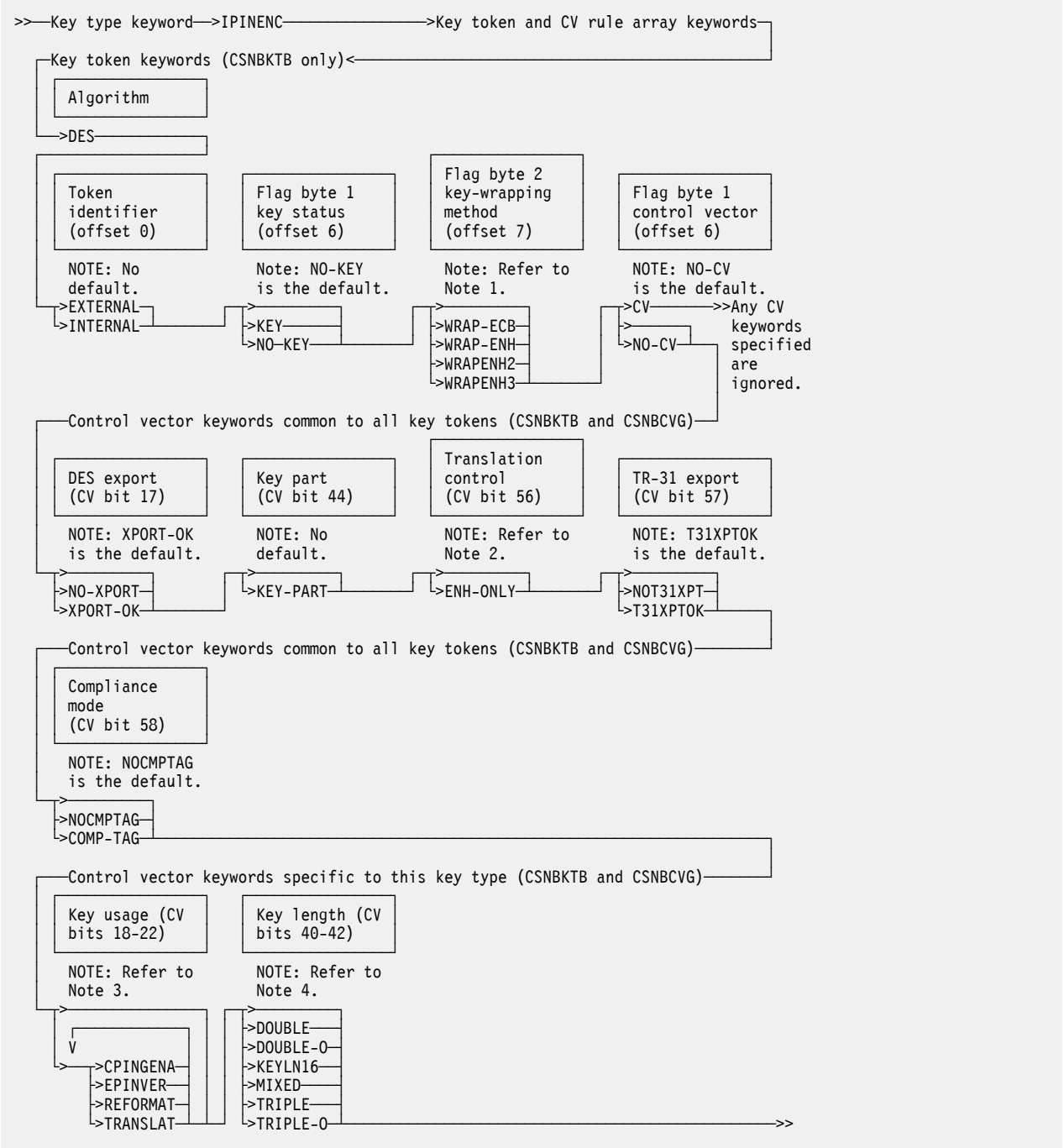


Figure 15. CSNBCVG and CSNBKTB keyword combinations for DES IPINENC keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.
2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. All keywords in this group are defaults unless one or more of these keywords are specified.
4. KEYLN16 and MIXED are synonymous with DOUBLE. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. DOUBLE is the default.

Figure 16 on page 289 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type OPINENC.

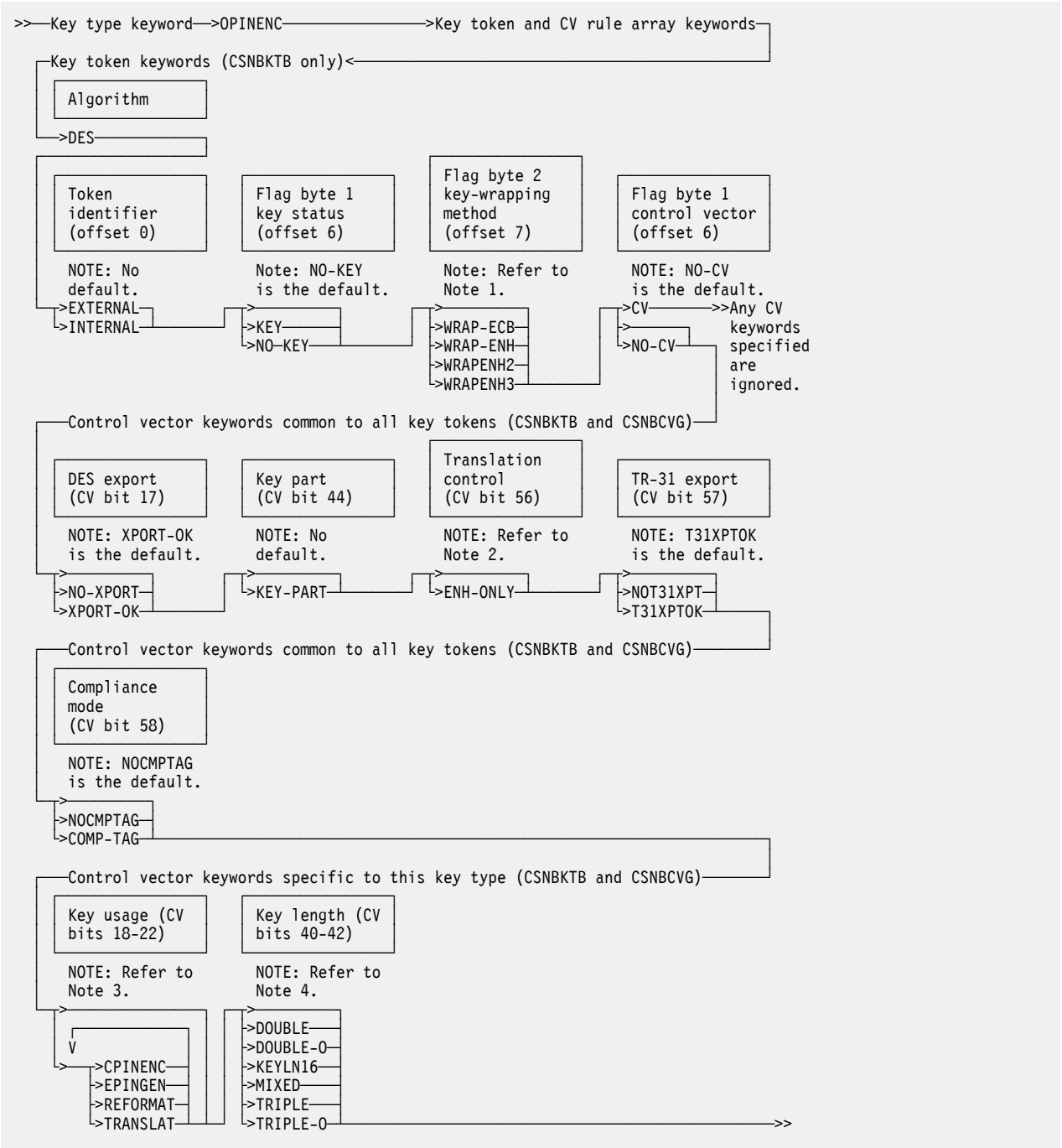


Figure 16. CSNBCVG and CSNBKTB keyword combinations for DES OPINENC keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.
2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. All keywords in this group are defaults unless one or more of these keywords are specified.
4. KEYLN16 and MIXED are synonymous with DOUBLE. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. DOUBLE is the default.

Figure 17 on page 290 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type PINGEN.

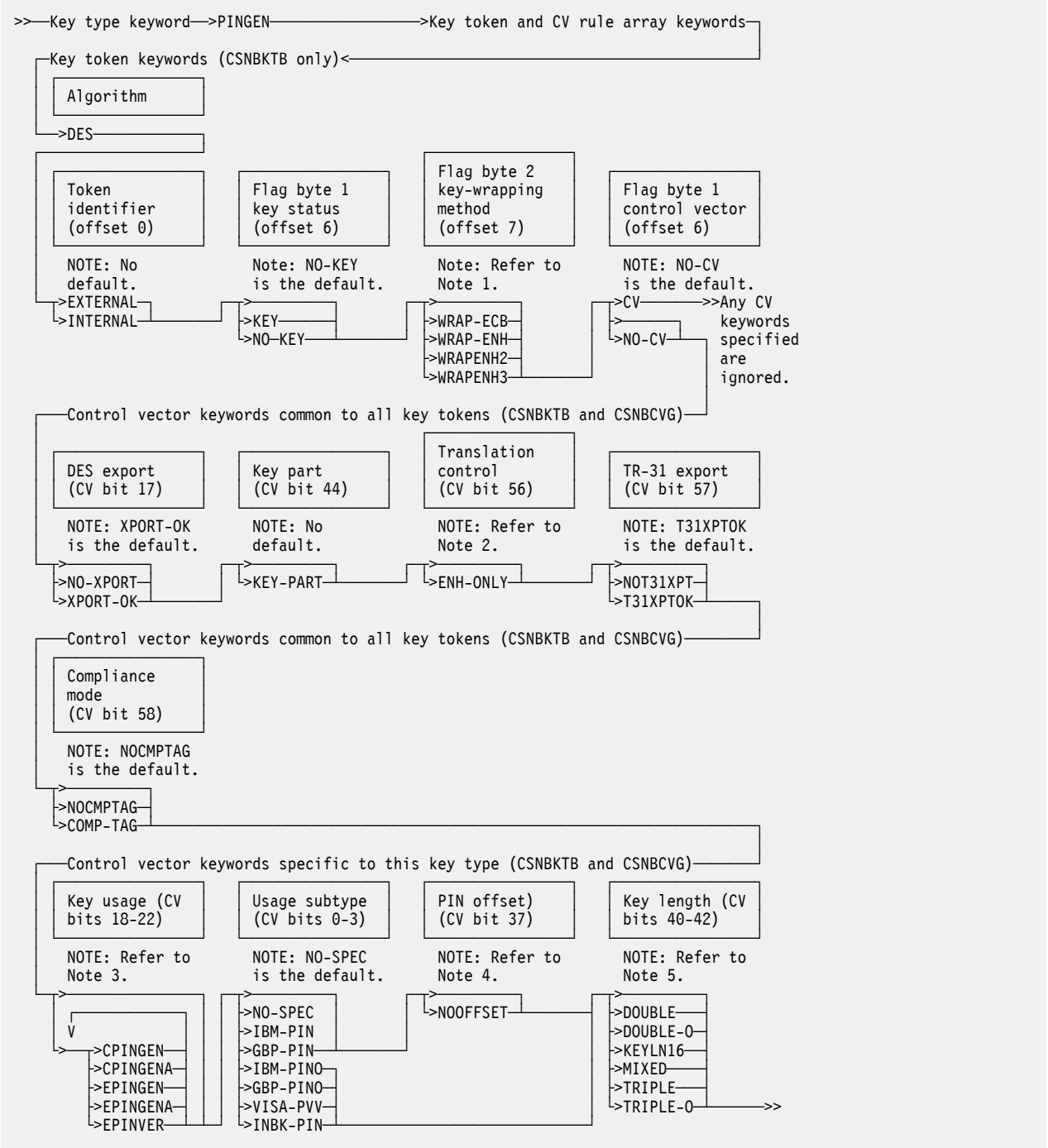


Figure 17. CSNBCVG and CSNBKTB keyword combinations for DES PINGEN keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.
2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. All keywords in this group are defaults unless one or more of these keywords are specified.
4. NOOFFSET has no effect with NO-SPEC, but is supported for backward compatibility. There is no default.
5. KEYLN16 and MIXED are synonymous with DOUBLE. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. DOUBLE is the default.

Figure 18 on page 291 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type PINVER.

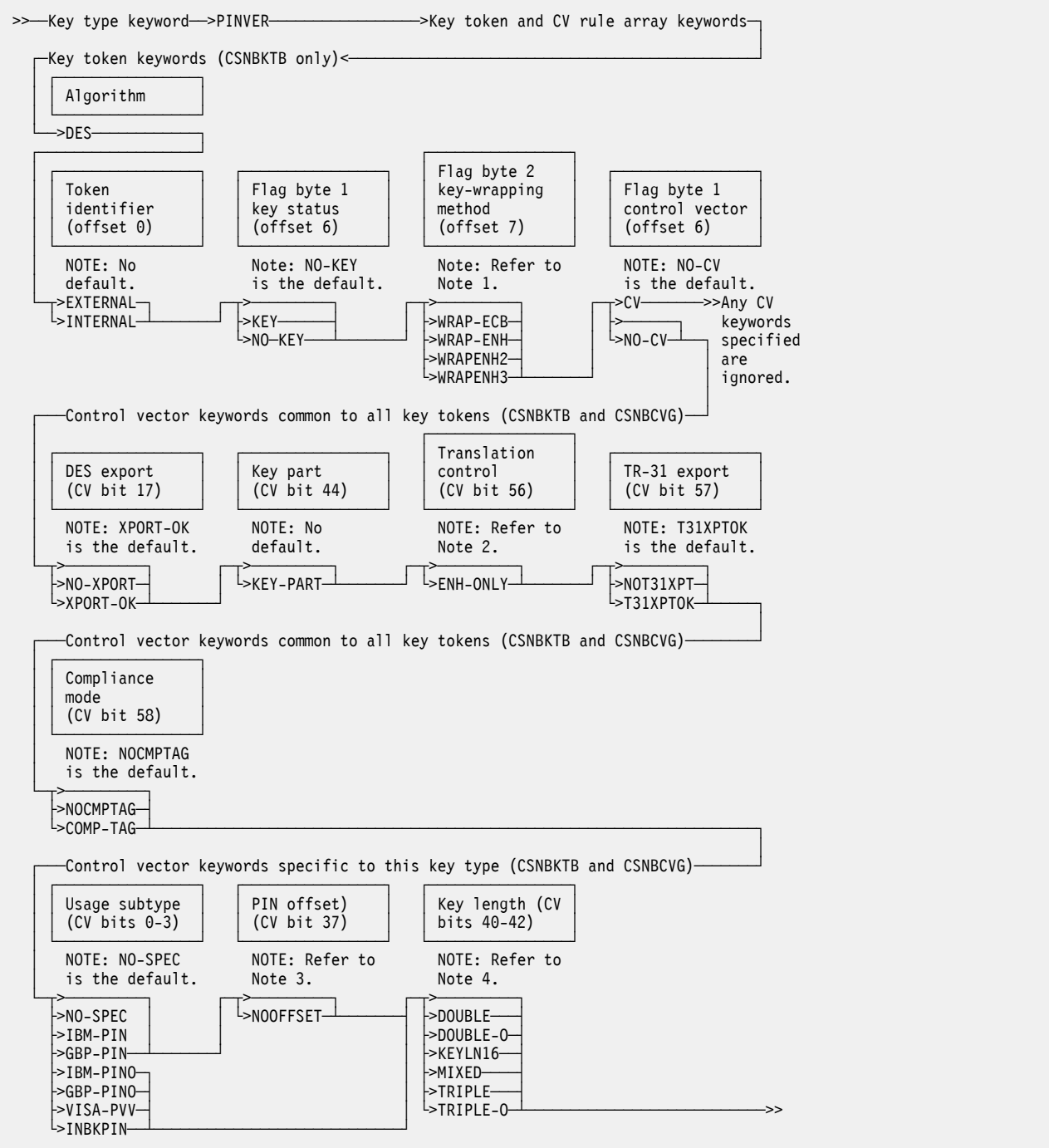


Figure 18. CSNBCVG and CSNBKTB keyword combinations for DES PINVER keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.
2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. NOOFFSET has no effect with NO-SPEC, but is supported for backward compatibility. There is no default.

4. KEYLN16 and MIXED are synonymous with DOUBLE. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. DOUBLE is the default.

Figure 19 on page 292 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type EXPORTER.

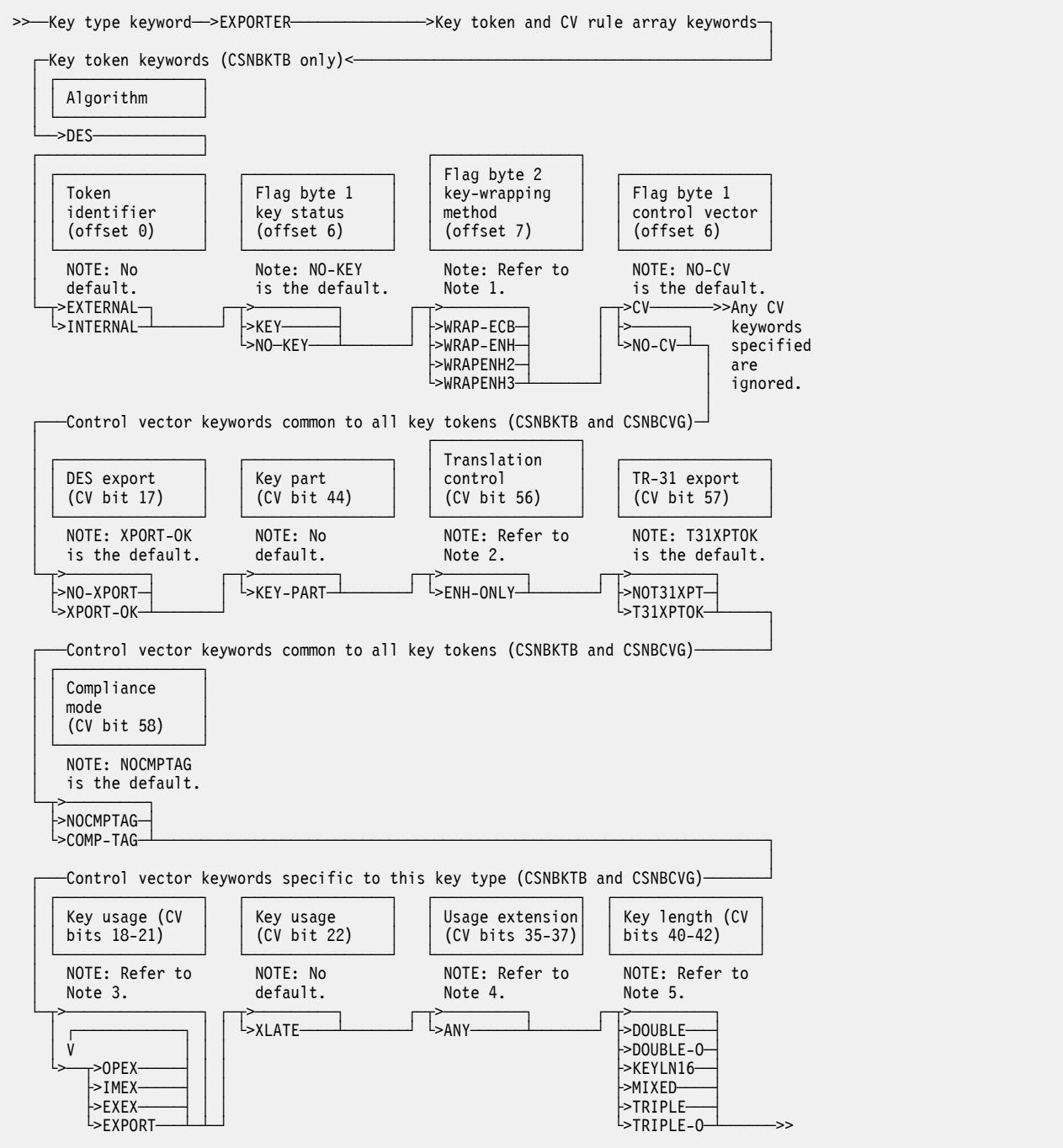


Figure 19. CSNBCVG and CSNBKTB keyword combinations for DES EXPORTER keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.
2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. All keywords in this group are defaults unless one or more of these keywords are specified.

4. Keyword ANY has been deprecated. Using ANY has no effect, but is allowed for backward compatibility.
5. KEYLN16 and MIXED are synonymous with DOUBLE. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. DOUBLE is the default.

Figure 20 on page 293 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type IMPORTER.

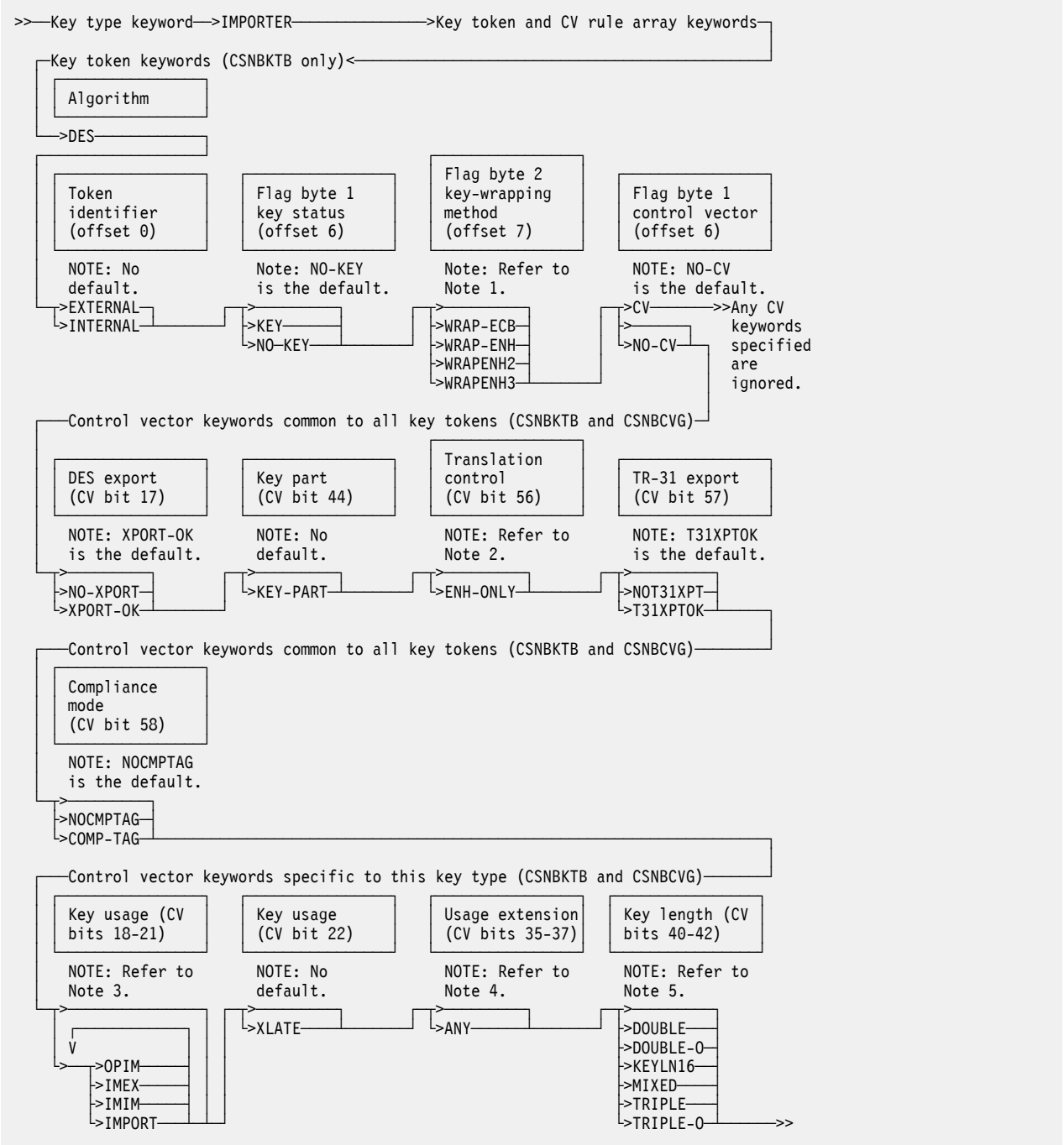


Figure 20. CSNBCVG and CSNBKTB, keyword combinations for DES IMPORTER keys

Notes:

1. WRAPENH2 is only valid for TRIPLE or TRIPLE-O. If TRIPLE or TRIPLE-O is specified, WRAPENH2 is the default. Otherwise, WRAP-ECB is the default.

2. ENH-ONLY is only valid with WRAP-ENH, WRAPENH2, or WRAPENH3. ENH-ONLY is the default for TRIPLE, TRIPLE-O, or WRAPENH3. Otherwise, there is no default.
3. All keywords in this group are defaults unless one or more of these keywords are specified.
4. Keyword ANY has been deprecated. Using ANY has no effect, but is allowed for backward compatibility.
5. KEYLN16 and MIXED are synonymous with DOUBLE. TRIPLE and TRIPLE-O are not valid with WRAP-ECB or WRAP-ENH. DOUBLE is the default.

Figure 21 on page 294 shows the Control Vector Generate and Key Token Build keyword combinations for DES key types IKEYXLAT and OKEYXLAT.

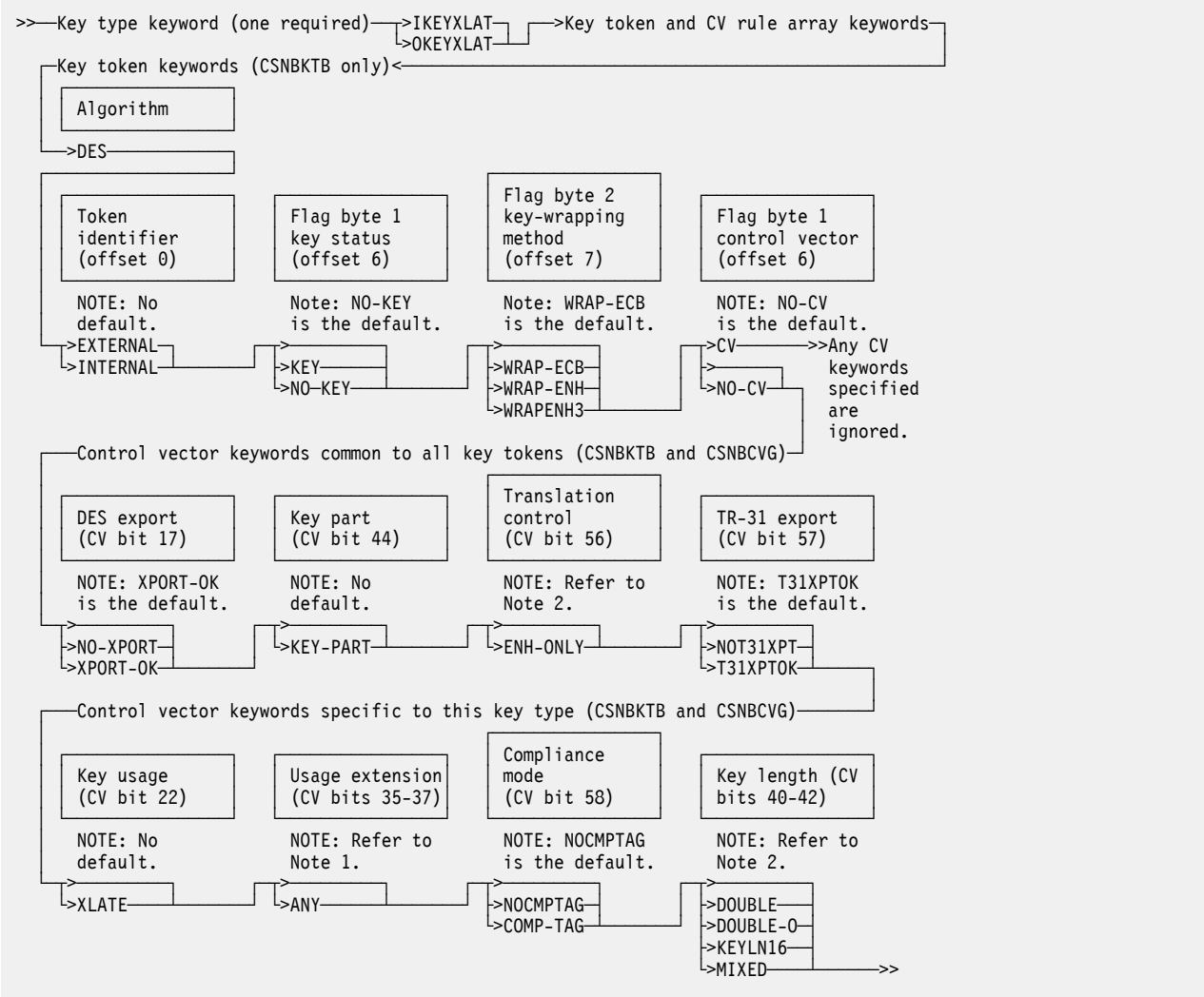


Figure 21. CSNBCVG and CSNBKTB keyword combinations for DES IKEYXLAT and OKEYXLAT keys

Notes:

1. Keyword ANY has been deprecated. Using ANY has no effect, but is allowed for backward compatibility.
2. KEYLN16 and MIXED are synonymous with DOUBLE. DOUBLE is the default.

Figure 22 on page 295 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type DKYGENKY.

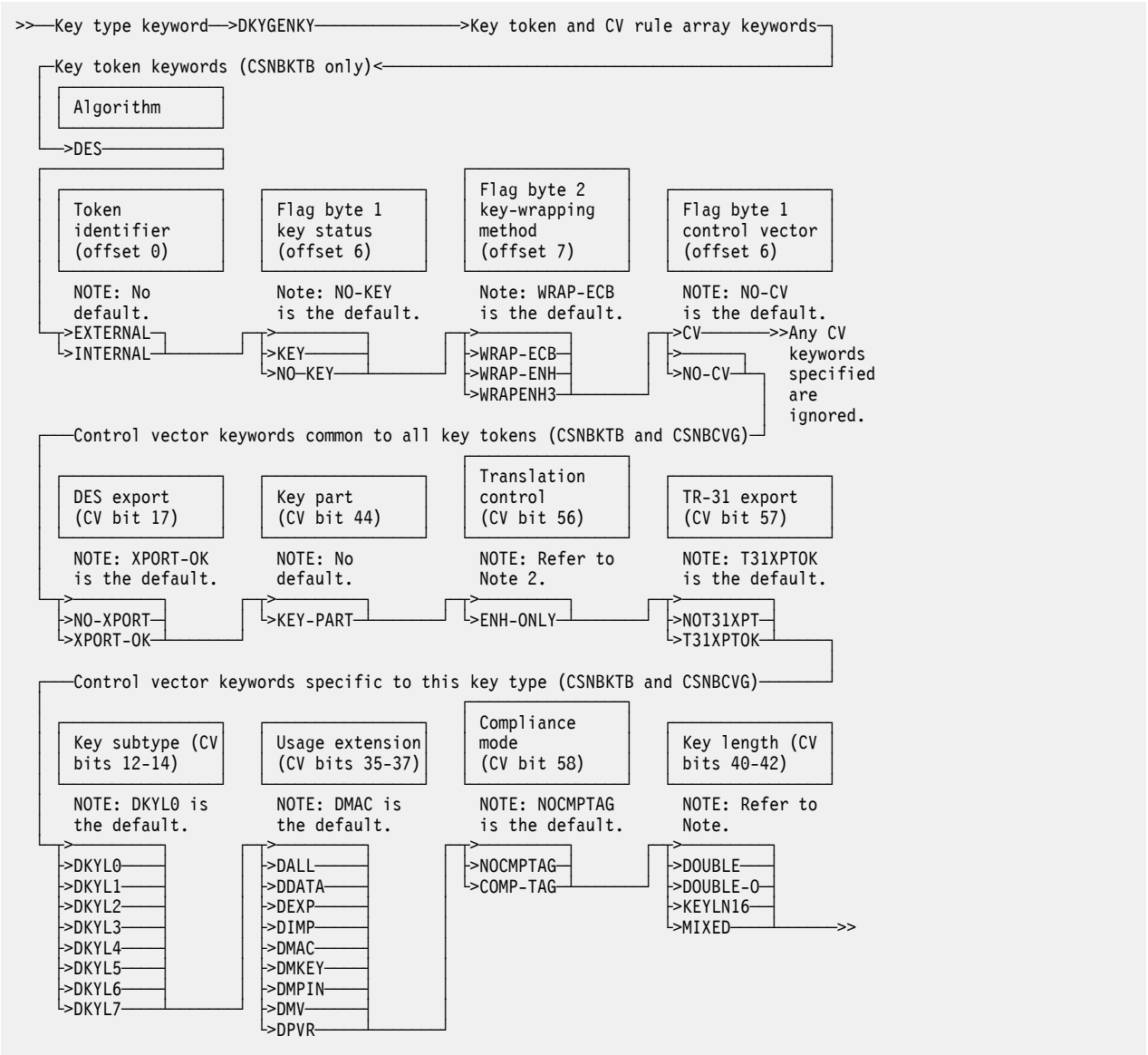


Figure 22. CSNBCVG and CSNBKTB keyword combinations for DES DKYGENKY keys

Note: KEYLN16 and MIXED are synonymous with DOUBLE. DOUBLE is the default.

Figure 23 on page 296 shows the Control Vector Generate and Key Token Build keyword combinations for DES key type KEYGENKY.

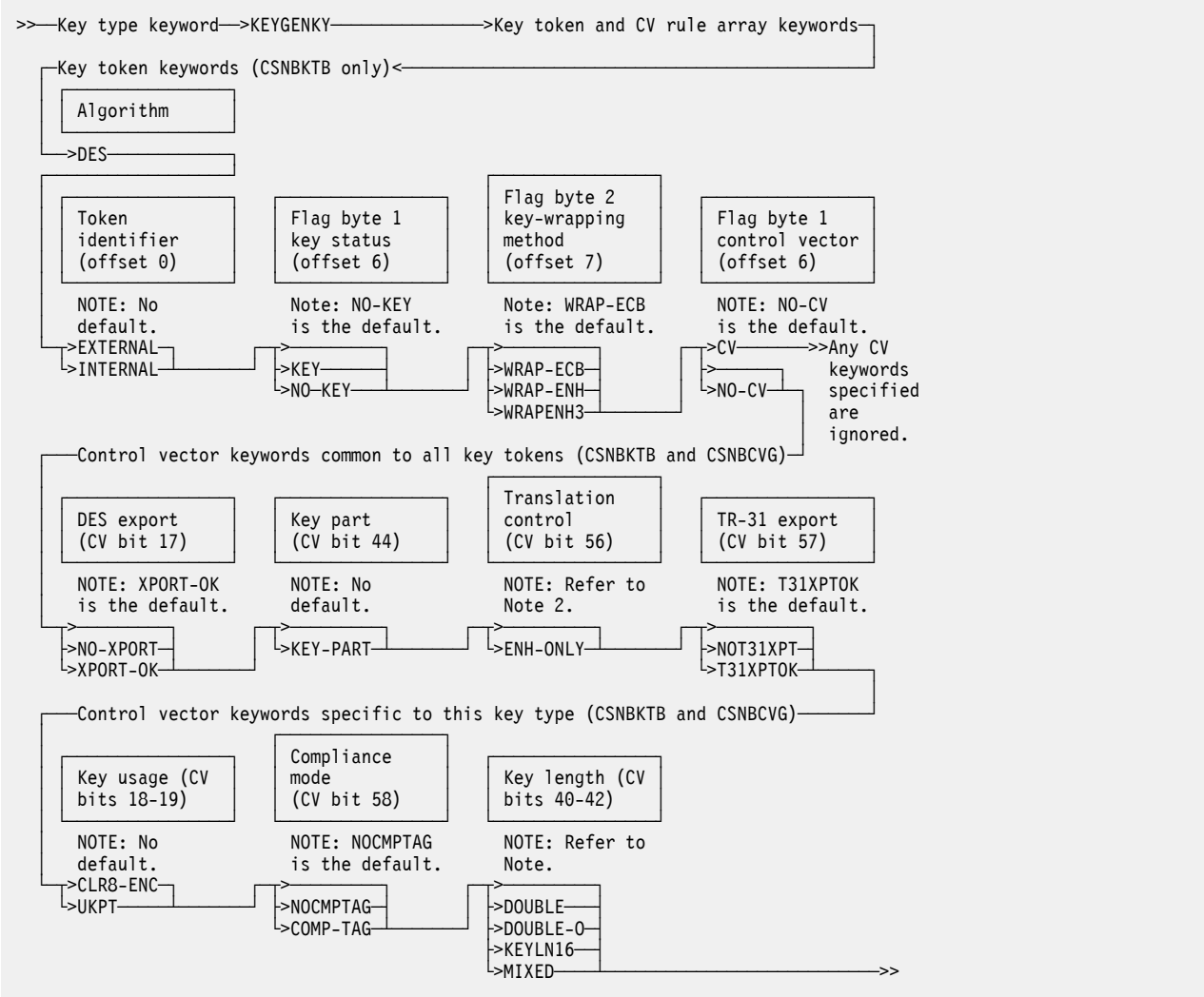


Figure 23. CSNBCVG and CSNBKTB keyword combinations for DES KEYGENKY keys

Note: KEYLN16 and MIXED are synonymous with DOUBLE. DOUBLE is the default.

Figure 24 on page 297 shows the Control Vector Generate and Key Token Build keyword combinations for DES key types CVARDEC, CVARENC, CVARPINE, CVARXCVL, and CVARXCVR.

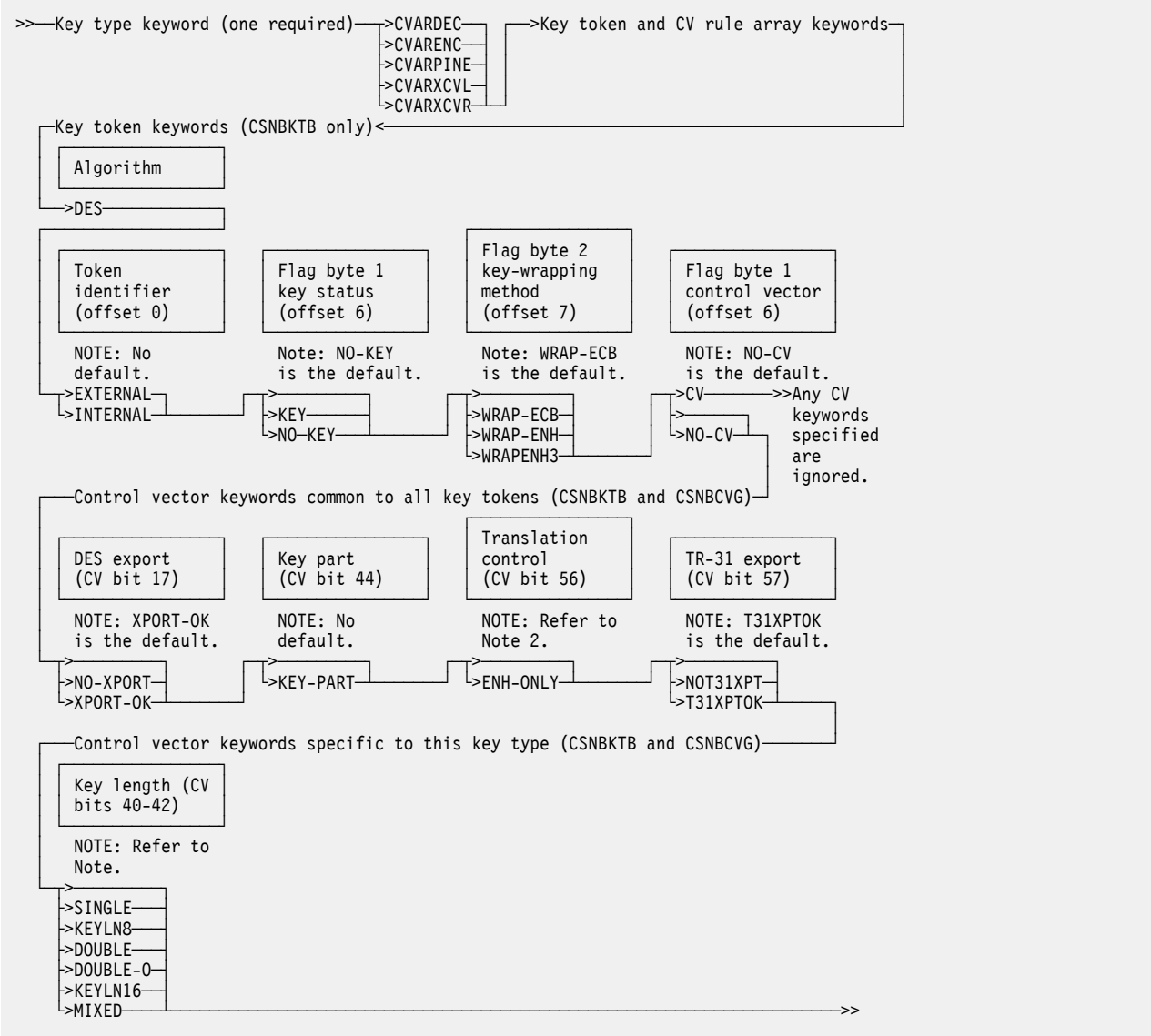


Figure 24. CSNBCVG and CSNBKTB keyword combinations for DES CVARDEC, CVARENC, CVARPINE, CVARXCVL, and CVARXCVR keys

Note: KEYLN8 is synonymous with SINGLE. KEYLN16 and MIXED are synonymous with DOUBLE. SINGLE is the default. The Cryptographic Variable Encipher service only supports single length keys, so do not specify DOUBLE, DOUBLE-O, KEYLN16, or MIXED if the key is to be used by this service.

Required hardware

When building clear key tokens, no cryptographic hardware is required. Otherwise, the master key associated with the specified algorithm must be active.

Key Token Build2 (CSNBKTB2 and CSNEKTB2)

Use the Key Token Build2 callable service to build a variable-length CCA symmetric key token in application storage from information that you supply. A clear key token built by this service can be used as input for the Key Test2 callable service. A skeleton token built by this service can be used as input for the Diversified Key Generate2, Key Generate2, Key Part Import2, and Secure Key Import2 callable services.

This service will build internal or external HMAC and AES tokens, both as clear key tokens and as skeleton tokens containing no key.

The callable service name for AMODE(64) is CSNEKTB2.

Format

```
CALL CSNBKTB2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              clear_key_bit_length,
              clear_key_value,
              key_name_length,
              key_name,
              user_associated_data_length,
              user_associated_data,
              token_data_length,
              token_data,
              service_data_length,
              service_data,
              target_key_token_length,
              target_key_token )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 3, and the maximum value is 34.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 117. Keywords for Key Token Build2 Control Information

Keyword	Meaning
Token type (one required)	
EXTERNAL	Specifies to build an external key token.
INTERNAL	Specifies to build an internal key token.
Token algorithm (one required)	
AES	Specifies to build an AES key token.
HMAC	Specifies to build an HMAC key token.
Key status (one, optional)	
KEY-CLR	Specifies to build the key token with a clear key value. This creates a key token that can be used with the Key Test2 service to generate a verification pattern for the key value.
NO-KEY	Specifies to build the key token without a key value. This creates a skeleton key token that can later be supplied to the Key Generate2 service. This is the default.
Payload version (one, optional)	
V0PYLD	Build a token with the old variable-length payload format for the target token. This is the default for AES CIPHER, EXPORTER, IMPORTER key types and is only valid with those key types.
V1PYLD	Build a token with the new fixed-length payload format for the target token. This is the default for AES MAC, PINPROT, PINCALC, PINPRW, DKYGENKY, and SECMSG key types. Not valid with the HMAC MAC key type.
Key type (one required)	
CIPHER	Specifies that this key is for data-encryption. Only valid for AES algorithm. See Figure 25 on page 303 and Table 118 on page 303 for all the valid keyword combinations and their defaults for AES key type CIPHER.
DKYGENKY	Specifies that this key is for key-generation. Only valid for AES algorithm. See Figure 30 on page 319 and Table 123 on page 320 for all the valid keyword combinations and their defaults for AES key type DKYGENKY.
EXPORTER	Specifies that this key is an EXPORTER key-encrypting key. Only valid for AES algorithm. See Figure 28 on page 312 and Table 121 on page 313 for all the valid keyword combinations and their defaults for AES key type EXPORTER.

Table 117. Keywords for Key Token Build2 Control Information (continued)	
Keyword	**Meaning**
IMPORTER	Specifies that this key is an IMPORTER key-encrypting key. Only valid for AES algorithm. See Figure 29 on page 315 and Table 122 on page 316 for all the valid keyword combinations and their defaults for AES key type IMPORTER.
KDKGENKY	Specifies that this key is a key diversification key key-encrypting key. Only valid for AES algorithm. See Figure 33 on page 329 and Table 128 on page 330 for all the valid keyword combinations and their defaults for AES key type KDKGENKY.
MAC	Specifies that this key is for message authentication code operations. Valid for HMAC and AES algorithms. See Figure 26 on page 306 and Table 119 on page 306 for all the valid keyword combinations and their defaults for AES key type MAC. See Figure 27 on page 309 and Table 120 on page 309 for all the valid keyword combinations and their defaults for HMAC key type MAC.
PINCALC	Specifies that this key is for calculating PINs. Only valid for AES algorithm. See Figure 31 on page 324 and Table 125 on page 324 for all the valid keyword combinations and their defaults for AES key type PINCALC.
PINPROT	Specifies that this key is for wrapping and unwrapping PIN blocks. Only valid for AES algorithm. See Figure 32 on page 326 and Table 127 on page 327 for all the valid keyword combinations and their defaults for AES key type PINPROT.
PINPRW	Specifies that this key is for generating and verifying PIN reference values. Only valid for AES algorithm. See Figure 34 on page 334 and Table 130 on page 334 for all the valid keyword combinations and their defaults for AES key type PINPRW.
SECMSG	Specifies that this key is for encrypting PINs in an EMV script. Only valid for AES algorithm. See Figure 35 on page 336 and Table 131 on page 337 for all the valid keyword combinations and their defaults for AES key type SECMSG.
Compliance (Optional)	
COMP-TAG	Build a compliant-tagged key token. Not valid with EXTERNAL, HMAC, or V0PYLD.
NOCMPTAG	Do not build a compliant-tagged key token. This is the default.

clear_key_bit_length

Direction	Type
Input	Integer

The length of the clear key in bits. Specify 0 when no key value is supplied (Key status rule NO-KEY). Specify a valid key bit length when a key value is supplied (Key status rule KEY-CLR):

- For HMAC algorithm, MAC key type, this is a value between 80 and 2048.
- For AES algorithm, this is a value of 128, 192, or 256.

clear_key_value

Direction	Type
Input	String

This parameter is used when the KEY-CLR keyword is specified. This parameter is the clear key value to be put into the token being built.

key_name_length

Direction	Type
Input	Integer

The length of the *key_name* parameter. Valid values are 0 and 64.

key_name

Direction	Type
Input	String

A 64-byte key store label to be stored in the associated data structure of the token.

user_associated_data_length

Direction	Type
Input	Integer

The length of the user-associated data. The valid values are 0 to 255 bytes.

user_associated_data

Direction	Type
Input	String

User-associated data to be stored in the associated data structure.

token_data_length

Direction	Type
Input	Integer

This parameter is reserved. The value must be zero.

token_data

Direction	Type
Ignored	Integer

This parameter is ignored.

service_data_length

Direction	Type
Input	Integer

The length of the *service_data* parameters in bytes. When the key type is DKYGENKY and the rule array keyword DKYUSAGE is specified or the key type is KDKGENKY, the value must be a multiple of 8. Otherwise, the value must be 0. The maximum value is 792.

service_data

Direction	Type
Input	String

Data to be processed by this service when building the skeleton token.

When the DKYUSAGE keyword and key type DKYGENKY are specified in the rule array, this parameter contains an array of key usage keywords for the type of key to be derived.

When the key type KDKGENKY is specified, this parameter contains an array of key usage keywords for the type of key to be derived. See the usage notes for the KDKGENKY key type for the description of the content of this parameter.

target_key_token_length

Direction	Type
Input/Output	Integer

On input, the length of the *target_key_token* parameter supplied to receive the token. On output, the actual length of the token returned to the caller. Maximum length is 725 bytes.

target_key_token

Direction	Type
Output	String

The key token built by this service.

Usage notes

The topic contains information for all key types detailing the key-usage and key-management keywords that are supported for each key type.

Figure 25 on page 303 shows all the valid keyword combinations and their defaults for AES key type CIPHER. For a description of these keywords, see Table 118 on page 303.

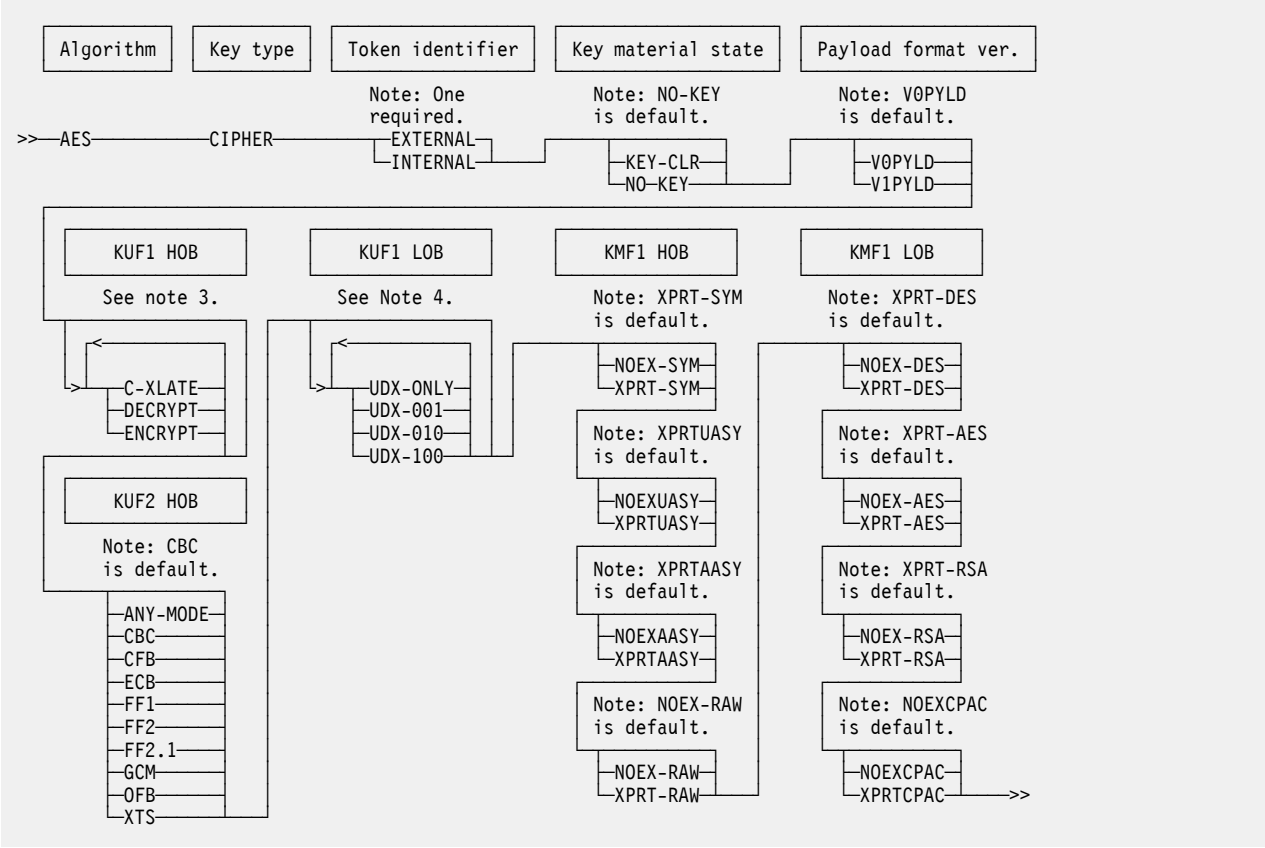


Figure 25. Key Token Build2 keyword combinations for AES CIPHER keys

Notes:

1. Keyword V0PYLD is the default for compatibility reasons. V1PYLD is recommended because it provides improved security.
2. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of a high-order byte (HOB) and a low-order byte (LOB).
3. Keywords DECRYPT and ENCRYPT are defaults if neither of these keywords is specified, regardless of whether C-XLATE is specified or not.
4. Choose any number of keywords in this group. No keywords in the group are defaults.
5. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
6. XPRTCPAC requires a CEX6C to generate the key token using the Key Generate2 (CSNBKGN2/ CSNEKGN2) callable service.

Table 118. Rule array keywords for AES CIPHER keys

Keyword	**Meaning**
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
KEY-CLR	Build a key token that contains a clear key.

Table 118. Rule array keywords for AES CIPHER keys (continued)	
Keyword	**Meaning**
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V0PYLD	Build a key token with a version 0 payload format. This format has a variable length and the key length can be inferred from the size of the payload. This format is compatible with all releases. This is the default.
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
CIPHER	Key can be used for encryption, decryption, and translation of data.
Encryption and translation control (one or more, optional). Key-usage field 1, high-order byte. Keywords DECRYPT and ENCRYPT are defaults unless one or more keywords in the group are specified.	
DECRYPT	Key can be used for decryption.
ENCRYPT	Key can be used for encryption.
Ciphertext Translate Control (optional). Key-usage field 1, high-order byte.	
C-XLATE	Key can only be used for data translate.
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
Encryption mode (one, optional). Key-usage field 2, high-order byte.	
ANY-MODE	Specifies that this key can be used for any encryption mode.
CBC	Specifies that this key can be used for cipher block chaining. This is the default.
CFB	Specifies that this key can be used for cipher feedback.
ECB	Specifies that this key can be used for electronic code book.
FF1	Specifies that this key can be used for Format Preserving method FF1.
FF2	Specifies that this key can be used for Format Preserving method FF2.
FF2.1	Specifies that this key can be used for Format Preserving method FF2.1.
GCM	Specifies that this key can be used for Galois/counter mode.
OFB	Specifies that this key can be used for output feedback.
XTS	Specifies that this key can be used for Xor-Encrypt-Xor-based Tweaked Stealing.

Table 118. Rule array keywords for AES CIPHER keys (continued)	
Keyword	**Meaning**
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.
CPACF export (one, optional).	
NOEXCPAC	Prohibit export to CPACF protected key format. This is the default.
XPRTCPAC	Allow export to CPACF protected key format.

Figure 26 on page 306 shows all the valid keyword combinations and their defaults for AES key type MAC. For a description of these keywords, see Table 119 on page 306.

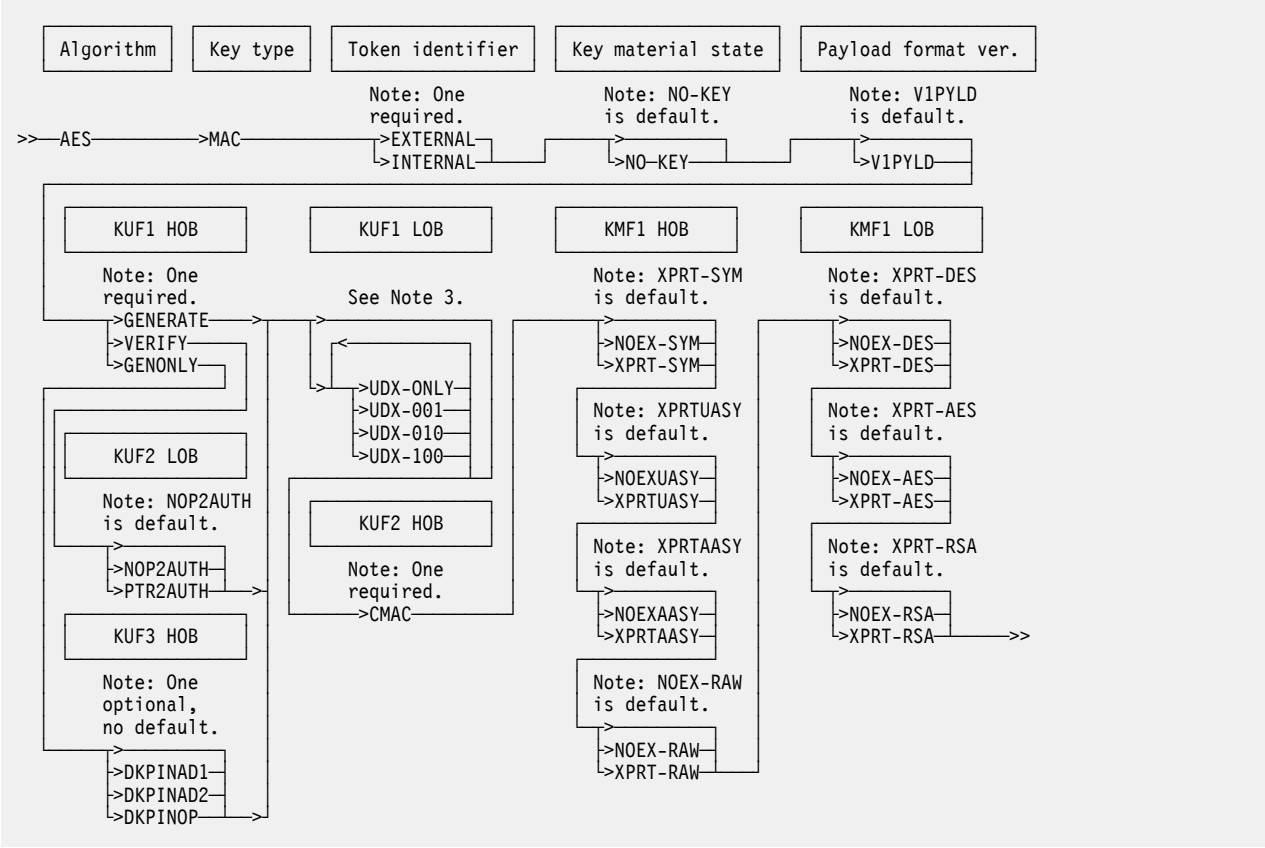


Figure 26. Key Token Build2 keyword combinations for AES MAC keys

Notes:

1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. Choose any number of keywords in this group. No keywords in the group are defaults.

Table 119. Rule array keywords for AES MAC keys

Keyword	**Meaning**
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	

Table 119. Rule array keywords for AES MAC keys (continued)	
Keyword	**Meaning**
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
MAC	Key can be used for generation and verification of message authentication codes.
MAC control (one, required). Key-usage field 1, high-order byte.	
GENERATE	Specifies that this key can be used to generate a MAC. A key that can generate a MAC can also verify a MAC. Not valid with keywords DKPINOP, DKPINAD1, and DKPINAD2.
GENONLY	Specifies that this key can only be used to generate a MAC. It cannot be used to verify a MAC.
VERIFY	Specifies that this key cannot be used to generate a MAC. It can only be used to verify a MAC.
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
MAC mode (one, required). Key-usage field 2, high-order byte.	
CMAC	Key can be used for block cipher-based MAC algorithm, called CMAC (NIST SP 800-38B).
Authentication data verification (one, optional). Key-usage field 2, low-order byte.	
NOP2AUTH	Key cannot be used by CSNBPTR2 to verify authentication data using NIST SP 800-38B CMAC for ISO-4 to ISO-4 PAN change. Only valid with key usage VERIFY. This is the default.
PTR2AUTH	Key can be used by CSNBPTR2 to verify authentication data using NIST SP 800-38B CMAC for ISO-4 to ISO-4 PAN change. Only valid with key usage VERIFY.
Common control (one, optional). Key-usage field 3, high-order byte. Use of a common control keyword causes key-usage field 3, low-order byte (field format identifier at token offset 050) to be set to X'01' (DK enabled).	
DKPINAD1	Specifies that this key may be used to create or verify a pin block to allow the changing of the account number associate with a PIN.
DKPINAD2	Specifies that this key may be used to create or verify an account change string to allow the changing of the account number associated with a PIN.
DKPINOP	Specifies that this key may be used as a general-purpose key. It may not be used as a special-purpose key.
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.

Table 119. Rule array keywords for AES MAC keys (continued)	
Keyword	**Meaning**
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.

Figure 27 on page 309 shows all the valid keyword combinations and their defaults for HMAC key type MAC. For a description of these keywords, see Table 120 on page 309.

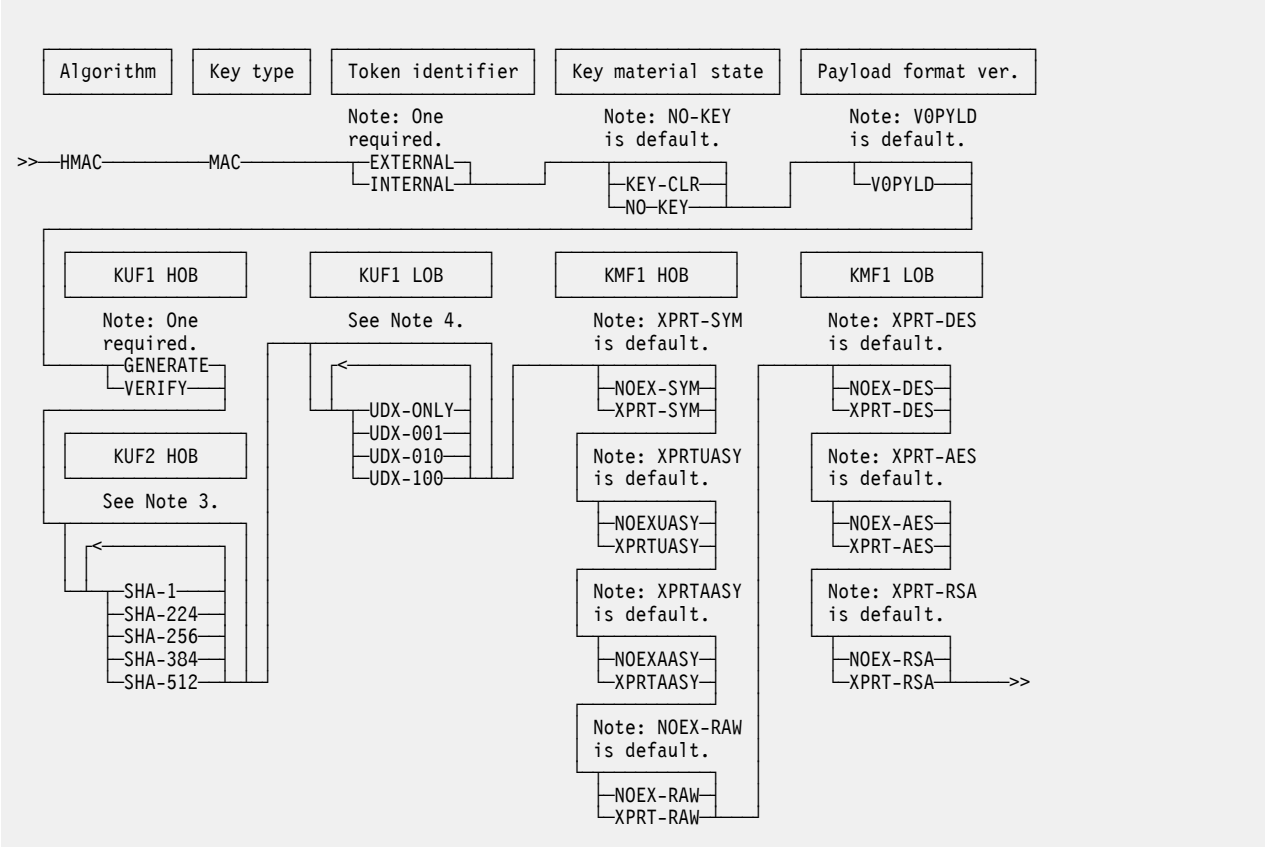


Figure 27. Key_Token_Build2 keyword combinations for HMAC MAC keys

Notes:

1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. All keywords in the group are defaults unless one or more keywords in the group are specified.
4. Choose any number of keywords in this group. No keywords in the group are defaults.

Table 120. Rule array keywords for HMAC MAC keys

Keyword	**Meaning**
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
KEY-CLR	Build a key token that contains a clear key.
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	

Table 120. Rule array keywords for HMAC MAC keys (continued)	
Keyword	**Meaning**
V0PYLD	Build a key token with a version 0 payload format. This format has a variable length and the key length can be inferred from the size of the payload. This is the default.
Associated data section	
Algorithm type (one required).	
HMAC	Key can be used for HMAC algorithm.
Key type (one required).	
MAC	Key can be used for generation and verification of message authentication codes.
MAC control (one, required). Key-usage field 1, high-order byte.	
GENERATE	Specifies that this key can be used to generate a MAC. A key that can generate a MAC can also verify a MAC.
VERIFY	Specifies that this key cannot be used to generate a MAC. It can only be used to verify a MAC.
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
Hash method control (any combination, optional). Key-usage field 2, high-order byte. Note: All keywords in the following list are defaults unless one or more keywords in the list are specified.	
SHA-1	Specifies that the SHA-1 hash method is allowed for the key.
SHA-224	Specifies that the SHA-224 hash method is allowed for the key.
SHA-256	Specifies that the SHA-256 hash method is allowed for the key.
SHA-384	Specifies that the SHA-384 hash method is allowed for the key.
SHA-512	Specifies that the SHA-512 hash method is allowed for the key.
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.

Table 120. Rule array keywords for HMAC MAC keys (continued)	
Keyword	**Meaning**
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.

Figure 28 on page 312 shows all the valid keyword combinations and their defaults for AES key type EXPORTER. For a description of these keywords, see Table 121 on page 313.

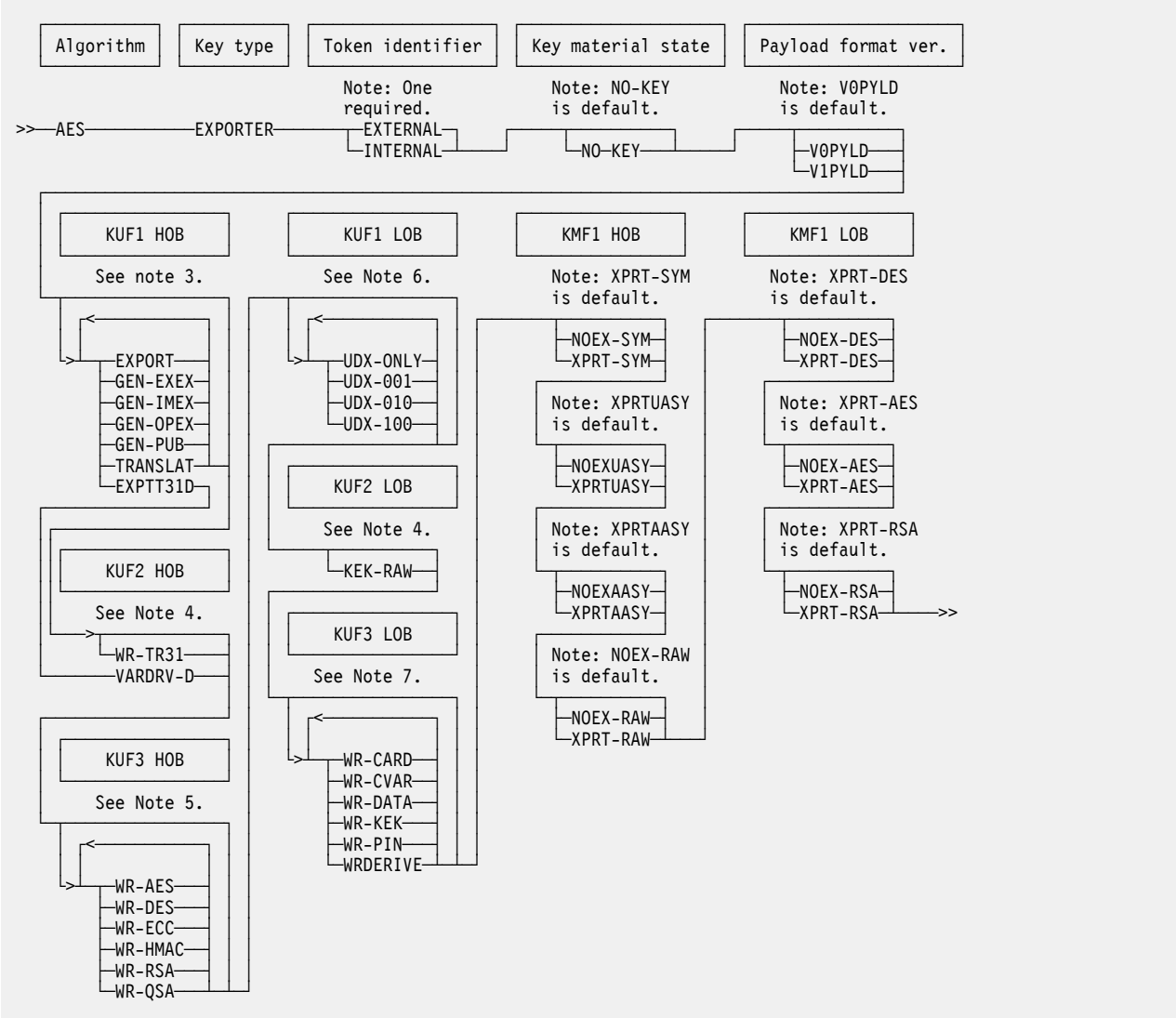


Figure 28. Key Token Build2 keyword combinations for AES EXPORTER keys

Notes:

1. Keyword V0PYLD is the default for compatibility reasons. V1PYLD is recommended because it provides improved security.
2. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of a high-order byte (HOB) and a low-order byte (LOB).
3. All keywords in this group are defaults except for EXPTT31D unless one or more of these keywords are specified. EXPTT31D cannot be specified with any other keyword in this group.
4. Keyword WR-TR31 is defined for future use and its meaning is currently undefined. To avoid this restriction in the future when the meaning is defined, specify this keyword. No keywords in this group are defaults.
5. Keywords WR-AES, WR-DES, and WR-HMAC are defaults unless one or more keywords in the group are specified, or if KUF2 HOB keyword VARDRV-D is specified. If VARDRV-D is specified, only keywords WR-AES, WR-DES, and WR-HMAC are allowed, and one or more of these keywords must be specified.
6. Choose any number of keywords in this group. No keywords in the group are defaults.
7. Keywords WR-CARD, WR-DATA, WR-KEK, WR-PIN, and WRDERIVE are defaults unless one or more keywords in this group are specified.
8. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.

Table 121. Rule array keywords for AES EXPORTER keys	
Keyword	**Meaning**
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V0PYLD	Build a key token with a version 0 payload format. This format has a variable length and the key length can be inferred from the size of the payload. This is the default.
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
EXPORTER	Key can be used for wrap a key to be exported.
Exporter control (any combination, optional). Key-usage field 1, high-order byte. Note: All keywords in the following list are defaults except for EXPTT31D unless one or more keywords in the list are specified.	
EXPORT	Specifies that this key can be used for export.
TRANSLAT	Specifies that this key can be used for translate.
GEN-OPEX	Specifies that this key can be used for generate OPEX.
GEN-IMEX	Specifies that this key can be used for generate IMEX.
GEN-EXEX	Specifies that this key can be used for generate EXEX.
GEN-PUB	Specifies that this key can be used for generate PUB.
EXPTT31D	Specifies that this key-encrypting key can wrap an AES or DES key using the Key Block Binding key wrapping method (key block protection) as defined in ISO/DIS 20038 (KBP1).
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
TR-31 wrap control (one, optional). Key-usage field 2, high-order byte.	

Table 121. Rule array keywords for AES EXPORTER keys (continued)	
Keyword	**Meaning**
VARDRV-D	Specifies that this key-encrypting key can wrap using ECB mode an AES TR-31 key block version “D”. Only valid with EXPTT31D.
WR-TR31	Specifies that this key-encrypting key can wrap or unwrap a TR-31 key block. Defined for future use. Not valid with EXPTT31D.
Raw key wrap control (one, optional). Key-usage field 2, low-order byte.	
KEK-RAW	Specifies that this key-encrypting key can export a RAW key. Defined for future use.
Key-usage wrap algorithm control (any combination, optional). Key-usage field 3, high-order byte. Note: Keywords WR-DES, WR-AES, and WR-HMAC are defaults unless one or more keywords are specified.	
WR-DES	Specifies that this key can be used to wrap DES keys.
WR-AES	Specifies that this key can be used to wrap AES keys.
WR-HMAC	Specifies that this key can be used to wrap HMAC keys.
WR-RSA	Specifies that this key can be used to wrap RSA keys.
WR-ECC	Specifies that this key can be used to wrap ECC keys.
WR-QSA	Specifies that this key can be used to wrap QSA keys.
Key-usage wrap class control (any combination, optional). Key-usage field 4, high-order byte. Note: Keywords WR-DATA, WR-KEK, WR-PIN, WRDERIVE, and WR-CARD in the following list are defaults unless one or more keywords in the list are specified.	
WR-DATA	Specifies that this key can be used to wrap DATA class keys.
WR-KEK	Specifies that this key can be used to wrap KEK class keys.
WR-PIN	Specifies that this key can be used to wrap PIN class keys.
WRDERIVE	Specifies that this key can be used to wrap DERIVATION class keys.
WR-CARD	Specifies that this key can be used to wrap CARD class keys.
WR-CVAR	Specifies that this key can be used to wrap CVAR class keys.
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.

Table 121. Rule array keywords for AES EXPORTER keys (continued)	
Keyword	**Meaning**
XPRT-RAW	Permits the export of the key in RAW format.
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.

Figure 29 on page 315 shows all the valid keyword combinations and their defaults for AES key type IMPORTER. For a description of these keywords, see Table 122 on page 316.

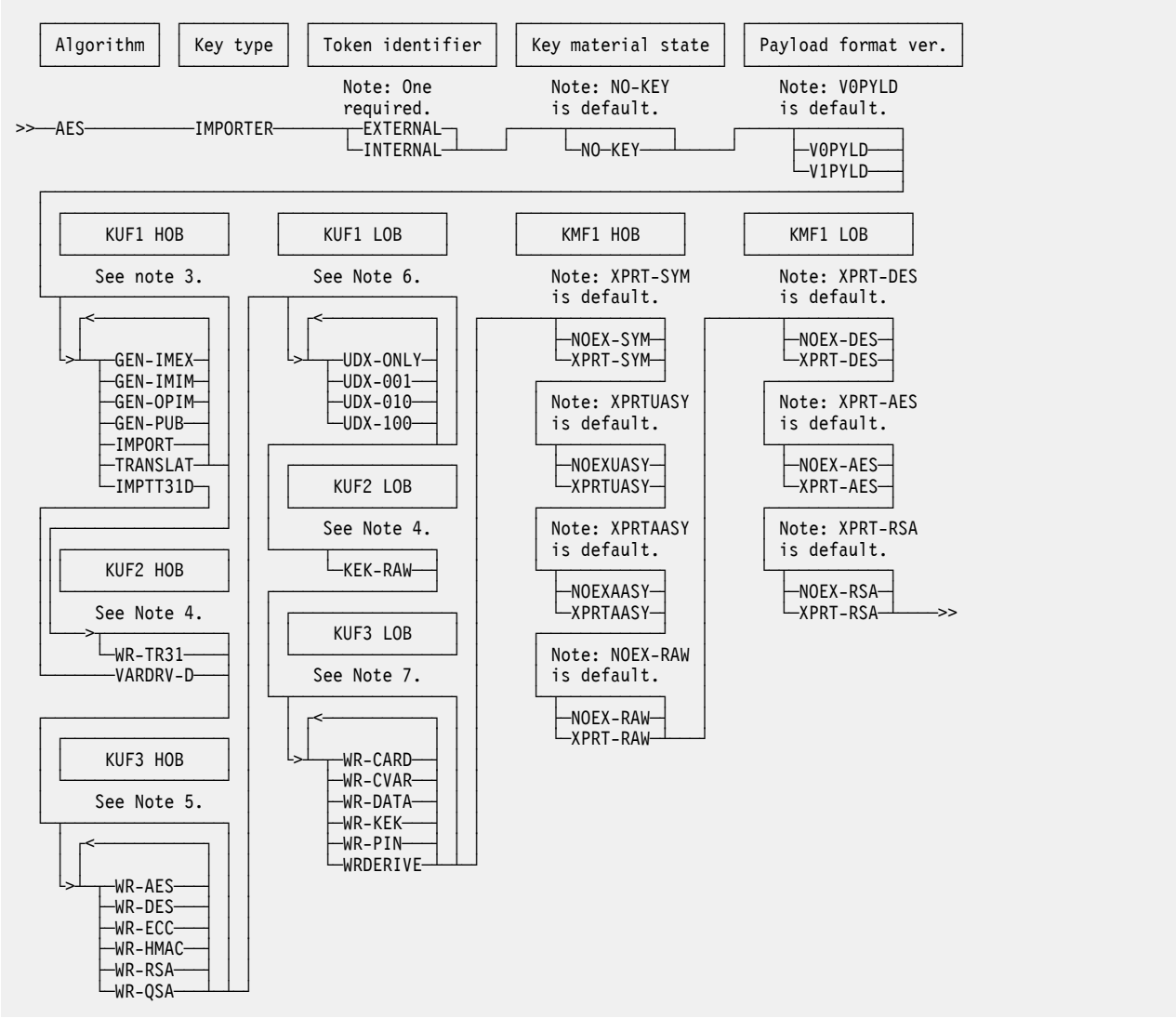


Figure 29. Key Token Build2 keyword combinations for AES IMPORTER keys

Notes:

1. Keyword V0PYLD is the default for compatibility reasons. V1PYLD is recommended because it provides improved security.
2. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of a high-order byte (HOB) and a low-order byte (LOB).
3. All keywords in this group are defaults except for IMPTT31D unless one or more of these keywords are specified. IMPTT31D cannot be specified with any other keyword in this group.
4. Keyword WR-TR31 is defined for future use and its meaning is currently undefined. To avoid this restriction in the future when the meaning is defined, specify this keyword. No keywords in this group are defaults.
5. Keywords WR-AES, WR-DES, and WR-HMAC are defaults unless one or more keywords in the group are specified, or if KUF2 HOB keyword VARDRV-D is specified. If VARDRV-D is specified, only keywords WR-AES, WR-DES, and WR-HMAC are allowed, and one or more of these keywords must be specified.
6. Choose any number of keywords in this group. No keywords in the group are defaults.
7. Keywords WR-CARD, WR-DATA, WR-KEK, WR-PIN, and WRDERIVE in the group are defaults unless one or more keywords in the group are specified.
8. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.

Table 122. Rule array keywords for AES IMPORTER keys

Keyword	Meaning
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V0PYLD	Build a key token with a version 0 payload format. This format has a variable length and the key length can be inferred from the size of the payload. This is the default.
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
IMPORTER	Key can be used for wrap a key to be imported.
Importer control (any combination, optional). Key-usage field 1, high-order byte. Note: All keywords in the following list are defaults except for IMPTT31D unless one or more keywords in the list are specified.	
IMPORT	Specifies that this key can be used for import.
TRANSLAT	Specifies that this key can be used for translate.

Table 122. Rule array keywords for AES IMPORTER keys (continued)	
Keyword	**Meaning**
GEN-OPIM	Specifies that this key can be used for generate OPIM.
GEN-IMEX	Specifies that this key can be used for generate IMEX.
GEN-IMIM	Specifies that this key can be used for generate IMIM.
GEN-PUB	Specifies that this key can be used for generate PUB.
IMPTT31D	Specifies that this key-encrypting key can unwrap an AES or DES key using the Key Block Binding key wrapping method (key block protection) as defined in ISO/DIS 20038 (KBP1).
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
TR-31 wrap control (one, optional). Key-usage field 2, high-order byte.	
VARDRV-D	Specifies that this key-encrypting key can wrap using ECB mode an AES TR-31 key block version “D”. Only valid with IMPTT31D.
WR-TR31	Specifies that this key-encrypting key can wrap or unwrap a TR-31 key block. Defined for future use. Not valid with IMPTT31D.
Raw key wrap control (one, optional). Key-usage field 2, low-order byte.	
KEK-RAW	Specifies that this key-encrypting key can export a RAW key. Defined for future use.
Key-usage wrap algorithm control (any combination, optional). Key-usage field 3, high-order byte. Note: Keywords WR-DES, WR-AES, and WR-HMAC are defaults unless one or more keywords are specified.	
WR-DES	Specifies that this key can be used to wrap DES keys.
WR-AES	Specifies that this key can be used to wrap AES keys.
WR-HMAC	Specifies that this key can be used to wrap HMAC keys.
WR-RSA	Specifies that this key can be used to wrap RSA keys.
WR-ECC	Specifies that this key can be used to wrap ECC keys.
WR-QSA	Specifies that this key can be used to wrap QSA keys.
Key-usage wrap class control (any combination, optional). Key-usage field 4, high-order byte. Note: Keywords WR-DATA, WR-KEK, WR-PIN, WRDERIVE, and WR-CARD in the following list are defaults unless one or more keywords in the list are specified.	
WR-DATA	Specifies that this key can be used to wrap DATA class keys.
WR-KEK	Specifies that this key can be used to wrap KEK class keys.
WR-PIN	Specifies that this key can be used to wrap PIN class keys.
WRDERIVE	Specifies that this key can be used to wrap DERIVATION class keys.
WR-CARD	Specifies that this key can be used to wrap CARD class keys.

Table 122. Rule array keywords for AES IMPORTER keys (continued)	
Keyword	**Meaning**
WR-CVAR	Specifies that this key can be used to wrap CVAR class keys.
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.

Figure 30 on page 319 shows all the valid keyword combinations and their defaults for AES key type DKYGENKY. For a description of these keywords, see Table 123 on page 320.

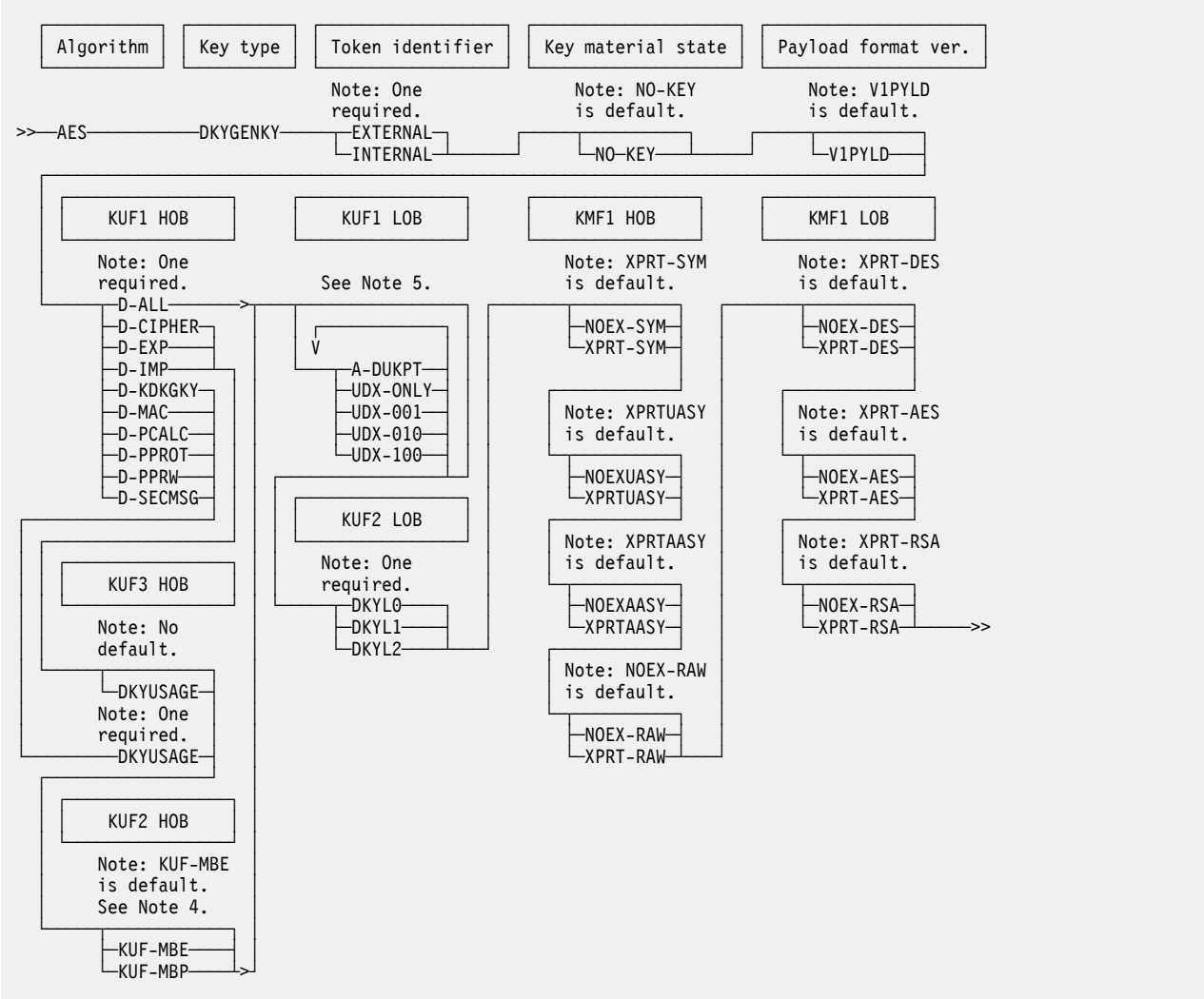


Figure 30. Key Token Build2 keyword combinations for AES DKYGENKY keys

Notes:

1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. DKYUSAGE specifies that the *service_data* variable contains the keywords that define the key usage attributes related to the type of key to diversify. Based on the *service_data* keywords, CSNBKTB2 appends the key usage attributes of the type of key to diversify to the key usage fields of the DKYGENKY key. The related key usage fields control which key usage attributes are permissible for the final generated diversified key. DKYUSAGE is not valid with D-ALL because the type of key to diversify is unspecified. DKYUSAGE is optional with D-CIPHER, D-EXP, and D-IMP because key types CIPHER, EXPORTER, and IMPORTER have default key usage attributes. For these key types, if DKYUSAGE is not specified, CSNBKTB2 assigns default key usage attributes to the related KUF fields. DKYUSAGE is required for the remaining values of type of key to diversify because those key types do not have default key usage attributes. DKYUSAGE is not valid with A-DUKPT.
4. KUF-MBP is not valid if DKADMIN1, DKADMIN2, DKPINOP, or DKPINOPP is specified in the *service_data* parameter (that is, the type of key to diversify is DK enabled) or if D-KDKGKY is specified in the *rule_array* variable.
5. Choose any number of keywords in this group. No keywords in this group are defaults.

6. To create a base derivation key skeleton to be used in AES DUKPT key derivation, specify key-usage keywords A-DUKPT, D-ALL, and DKYL0.

Table 123. Rule array keywords for AES DKYGENKY keys	
Keyword	**Meaning**
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
DKYGENKY	Key can be used for generating a diversified key.
Type of key to diversify (one, required). Key-usage field 1, high-order byte. DKYUSAGE is required for D-KDKGKY, D-MAC, D-PCALC, D-PPROT, D-PPRW, and D-SECMSG. A-DUKPT is only valid with D-ALL.	
D-ALL	Specifies that this can derive any AES key type listed in this section.
D-CIPHER	Specifies that this key can derive an AES CIPHER key.
D-EXP	Specifies that this key can derive an AES EXPORTER key.
D-IMP	Specifies that this key can derive an AES IMPORTER key.
D-KDKGKY	Specifies that this key can generate an AES KDKGENKY key.
D-MAC	Specifies that this key can derive an AES MAC key.
D-PCALC	Specifies that this key can derive an AES PINCALC key.
D-PPROT	Specifies that this key can derive an AES PINPROT key.
D-PPRW	Specifies that this key can derive an AES PINPRW key.
D-SECMSG	Specifies that this key can derive an AES SECMSG key.
Special usages and user-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
A-DUKPT	This key can be used as the base derivation key (BDK) in the AES DUKPT key derivation algorithm.
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.

Table 123. Rule array keywords for AES DKYGENKY keys (continued)	
Keyword	**Meaning**
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
Key-usage field level of control (one, optional). Key-usage field 2, high-order byte. Note: Not valid when D-ALL key derivation control is specified.	
KUF-MBE	Specifies that the key usage fields of the key to be generated must be equal to the related generated key usage fields of the DKYGENKY generating key. This is the default.
KUF-MBP	Specifies that the key usage fields of the key to be generated must be permitted based on the related generated key usage fields of the DKYGENKY generating key. The key to be diversified is not permitted to have a higher level of usage than the related key usage fields permit. The key to be diversified is only permitted to have key usage that is less than or equal to the related key usage fields. The UDX-ONLY bit of the related key usage fields must always be equal in both the generating key and the generated key. Note: This value is not valid if the key is to be used to derive keys for the DK PIN methods.
Key-derivation sequence level (one, required). Key-usage field 2, low-order byte. A-DUKPT is only valid with DKYL0.	
DKYL0	Specifies that this key-generating key can be used to derive the key specified by the key derivation and derived key usage controls.
DKYL1	Use this diversifying key to generate a level 0 diversified key.
DKYL2	Use this diversifying key to generate a level 1 diversified key.
Related generated key usage fields (not allowed for D-ALL, one, optional, for D-CIPHER, D-EXP, and D-IMP, otherwise one required). Key-usage field 3, high-order byte.	
DKYUSAGE	Specifies that the *service_data* parameter identifies key usage information for a DKYGENKY. This information pertains to the allowable key usage of the key to be derived. This keyword is not valid with A-DUKPT.
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.

Table 123. Rule array keywords for AES DKYGENKY keys (continued)	
Keyword	**Meaning**
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.

Table 124. Meaning of service_data parameter when DKYUSAGE specified

Type of key to diversify	DKYUSAGE keyword	Description of verb_data variable when DKYUSAGE specified
D-ALL	Not allowed	Not applicable.
D-CIPHER	Optional	If keyword DKYUSAGE is specified, the *service_data* parameter must contain key usage fields keywords related to an AES CIPHER key. If not specified, the related key usage fields will be that of a default AES CIPHER key.
D-EXP	Optional	If keyword DKYUSAGE is specified, the *service_data* parameter must contain key usage fields keywords related to an AES EXPORTER key. If not specified, the related key usage fields will be that of a default AES EXPORTER key.
D-IMP	Optional	If keyword DKYUSAGE is specified, the *service_data* parameter must contain key usage fields keywords related to an AES IMPORTER key. If not specified, the related key usage fields will be that of a default AES IMPORTER key.
D-KDKGKY	Required	The *service_data* parameter must contain key usage fields keywords related to an AES KDKGENKY key.
D-MAC	Required	The *service_data* parameter must contain key usage fields keywords related to an AES MAC key.
D-PCALC	Required	The *service_data* parameter must contain key usage fields keywords related to an AES PINCALC key.
D-PPROT	Required	The *service_data* parameter must contain key usage fields keywords related to an AES PINPROT key.
D-PPRW	Required	The *service_data* parameter must contain key usage fields keywords related to an AES PINPRW key.
D-SECMSG	Required	The *service_data* parameter must contain key usage fields keywords related to an AES SECMSG key.

Building a DKYGENKY key

The way that the DKYGENKY tokens are built is different from the way they were previously built. The token layout itself has been updated. The DKYGENKY key is used to derive other key types.

In order to control the key usage of the key to be derived, key usage field information for the derived key is included in the DKYGENKY token. Consider these scenarios based on the type of key to derive:

- DKYGENKY has a type of key to be derived of D-ALL.

  This type of key is allowed to derive any of the allowed key types. No key usage field information is included in this key. Usage is determined by the skeleton token identified by the generated_key_identifier parameter of the CSNBDKG2 callable service. A special access control point must be enabled in the active role to use this option.
- DKYGENKY has a type of key to be derived that has default key usage (D-CIPHER, D-EXP, D-IMP).

  Several key types have default key usage defined, while other key types do not. For those key types which have default key usage defined (D-CIPHER, D-EXP, and D-IMP), the only requirement is to specify the type of key to derive. The default key usage fields is included in the DKYGENKY key, beginning with key usage field 3.
- DKYGENKY has a type of key to be derived that requires non-default key usage.

  If non-default key usage of a key to be derived is required or desired, specify rule array keyword DKYUSAGE. With this keyword, the verb_data parameter is used to identify all of the key-usage field keywords for the key to be diversified. Do not specify any token identifier, type of algorithm, key type, or key management field keywords. Set the verb_data_length value to the number of bytes in the verb_data variable. This length must be a multiple of 8.

  When rule array keyword DKYUSAGE is specified, choose between whether the key usage field attributes in the DKYGENKY starting at key-usage field 3 have the strictest control (KUF-MBE or 'must be equal', which is the default) or allow flexibility in the key usage attributes of the key to be generated (KUF-MBP or 'must be permitted').

  Choosing KUF-MBE ('key usage fields must be equal') provides a one-to-one mapping of usage fields between the generating key and the generated key. The key usage fields related to the key to be diversified in the DKYGENKY key must match exactly with the key usage fields of any skeleton key provided as input to the CSNBDKG2 callable service.

  Choosing KUF-MBP ('key usage fields must be permitted') provides that the key to be diversified is allowed to have any key usage attribute that is enabled in the DKYGENKY. For example, if a DKYGENKY with D-EXP usage has default EXPORTER key usage fields, KUF-MBP allows the diversified EXPORTER key to have only the EXPORT bit on in key-usage field 1. This is permitted because the diversified key actually is more restrictive than the usage allowed by the DKYGENKY key. Conversely, if a DKYGENKY with D-EXP usage has only the EXPORT bit on in key-usage field 3 (which maps to key usage field 1 of the diversified EXPORTER key), it would not be permitted for the skeleton key used as input to the CSNBDKG2 callable service to have the XLATE bit on in key-usage field 1.

  Notes:

  – For rule array keyword KUF-MBP, one exception exists where the value of the UDX-ONLY bit in key usage field 3 of a DKYGENKY key must always match the value of the UDX-ONLY bit in key usage field 1 of the diversified key.

  – Under access control point control, there is one case where a many-to-one mapping is permitted and verb data is not used. This case is when you specify D-ALL which says any allowable key type can be derived.

Figure 31 on page 324 shows all the valid keyword combinations and their defaults for AES key type PINCALC. For a description of these keywords, see Table 125 on page 324.

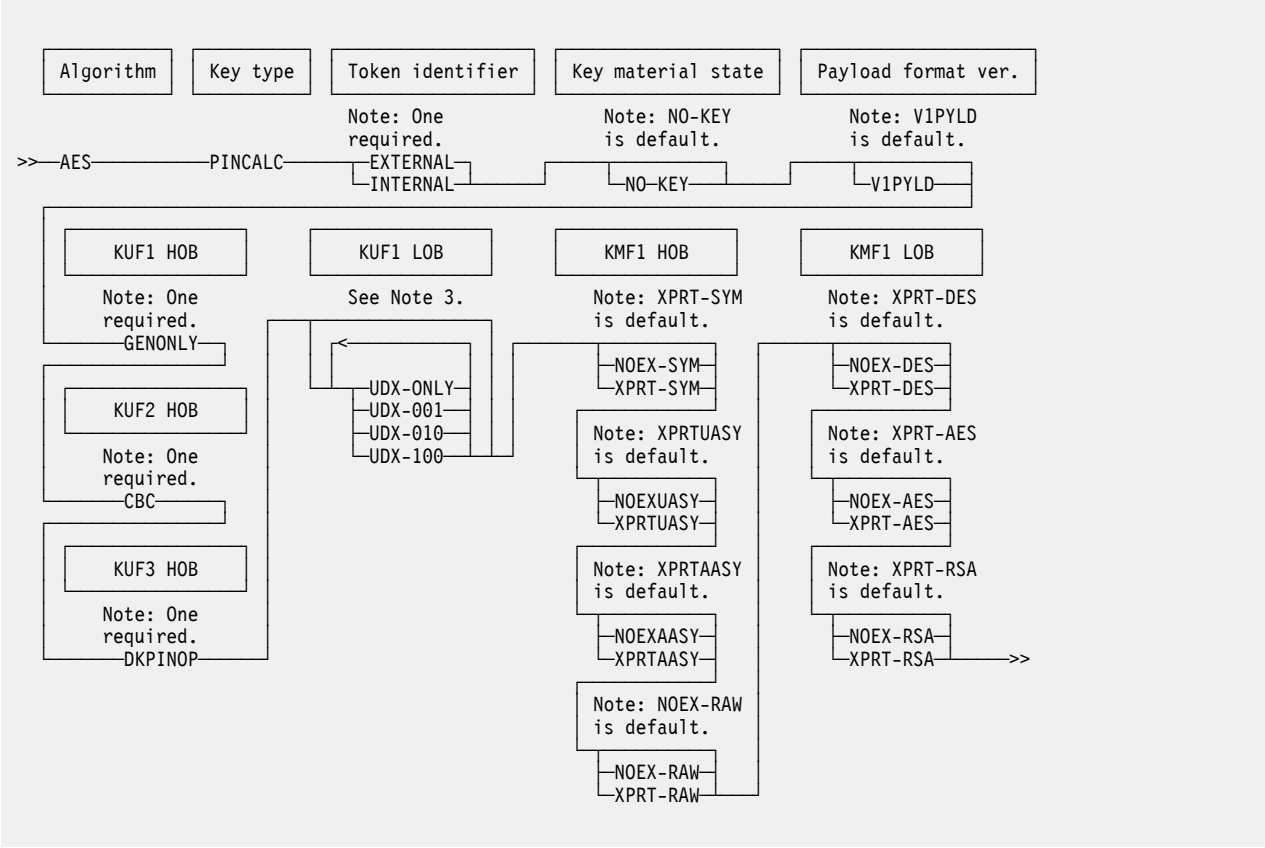


Figure 31. Key Token Build2 keyword combinations for AES PINCALC keys

Notes:

1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. Choose any number of keywords in this group. No keywords in the group are defaults.

Table 125. Rule array keywords for AES PINCALC keys

Keyword	Meaning
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	

Table 125. Rule array keywords for AES PINCALC keys (continued)	
Keyword	**Meaning**
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
PINCALC	Key can be used for generating PINs.
Generate control (one required). Key-usage field 1, high-order byte.	
GENONLY	Specifies that this key can only be used to generate a PIN. It cannot be used to verify a PIN.
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
Encryption mode (one, required). Key-usage field 2, high-order byte.	
CBC	Specifies that this key can be used for cipher block chaining.
Common control (one, required). Key-usage field 3, high-order byte. Use of a common control keyword causes key-usage field 3, low-order byte (field format identifier at token offset 050) to be set to X'01' (DK enabled).	
DKPINOP	Specifies that this key may be used as a general-purpose key. It may not be used as a special-purpose key.
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.

Table 125. Rule array keywords for AES PINCALC keys (continued)	
Keyword	**Meaning**
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.

Figure 32 on page 326 shows all the valid keyword combinations and their defaults for AES key type PINPROT. For a description of these keywords, see Table 127 on page 327.

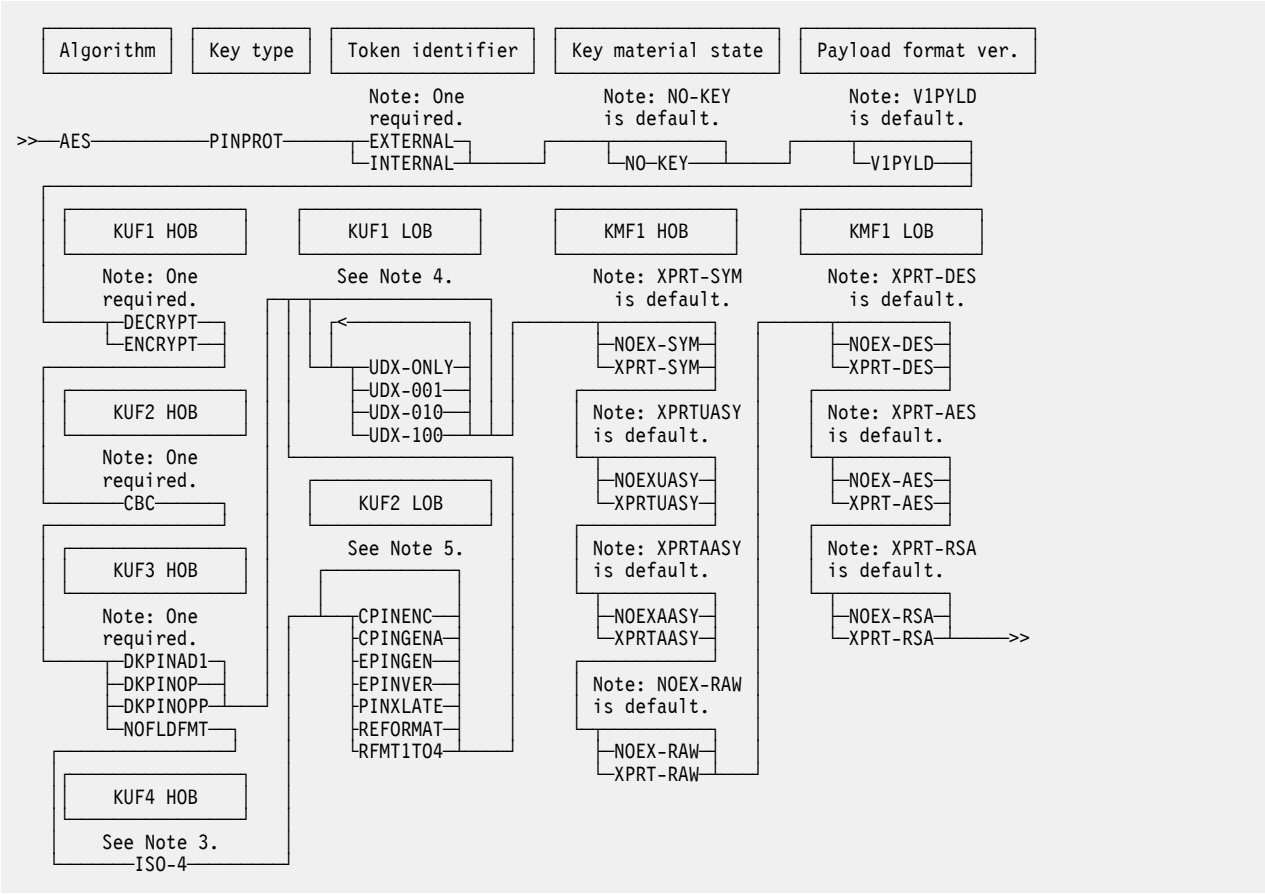


Figure 32. Key Token Build2 keyword combinations for AES PINPROT keys

Notes:

1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. If and only if common control keyword NOFLDFMT is specified, one keyword must be selected from this group. If NOFLDFMT is not specified, no KUF4 is created.
4. Choose any number of keywords in this group. No keywords in the group are defaults.
5. If and only if common control keyword NOFLDFMT is specified, at least one keyword must be selected from this group. No keywords in this group are defaults.

Table 126. Allowable keyword combinations for PINPROT keys		
KUF3 HOB keywords	**KUF1 HOB keywords**	**KUF2 LOB keywords (at least one must be specified, any number may be specified)**
DKPINAD1, DKPINOP, or DKPINOPP	DECRYPT or ENCRYPT	
NOFLDFMT	DECRYPT	CPINGENA, EPINVER, PINXLATE, REFORMAT, RFMT4TO1
NOFLDFMT	ENCRYPT	CPINENC, EPINGEN, PINXLATE, REFORMAT, RFMT1TO4

Table 127. Rule array keywords for AES PINPROT keys	
Keyword	**Meaning**
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
PINPROT	Key can be used for encrypting and decrypting PIN blocks.
Encryption operation (one, required). Key-usage field 1, high-order byte.	
DECRYPT	Specifies that this key can be used to decipher data (inbound). The key cannot be used to encipher data.
ENCRYPT	Specifies that this key can be used to encipher data (outbound). The key cannot be used to decipher data.
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.

Table 127. Rule array keywords for AES PINPROT keys (continued)	
Keyword	**Meaning**
Encryption mode (one, required). Key-usage field 2, high-order byte.	
CBC	Specifies that this key can be used for cipher block chaining.
PIN services control (one or more, required when the common control keyword NOFLDFMT is specified). Key-usage field 2, low-order byte. See Table 127 on page 327 for valid combinations.	
CPINENC	Key can be used with the Clear PIN Encrypt callable service. Only valid with ENCRYPT keyword.
CPINGENA	Key can be used with the Clear PIN Generate Alternate callable service. Only valid with DECRYPT keyword.
EPINGEN	Key can be used with the Encrypted PIN Generate callable service. Only valid with ENCRYPT keyword.
EPINVER	Key can be used with the Encrypted PIN Verify callable service. Only valid with DECRYPT keyword.
PINXLATE	Key can be used with the Encrypted PIN Translate2 callable service for a translation operation.
REFORMAT	Key can be used with the Encrypted PIN Translate callable service for a reformat operation.
RFMT1TO4	Key can be used to restrictively reformat an ISO-1 encrypted PIN to an ISO-4 encrypted PIN. Only valid with ENCRYPT keyword.
RFMT4TO1	Key can be used to restrictively reformat an ISO-4 encrypted PIN to an ISO-1 encrypted PIN. Only valid with the DECRYPT keyword.
Common control (one, required). Key-usage field 3, high-order byte. Use of a common control keyword DKPINOP, DKPINOPP, or DKPINAD1 causes key-usage field 3, low-order byte (field format identifier at token offset 050) to be set to X'01' (DK enabled). Keyword NOFLDFMT causes key-usage field 3, low-order byte (field format identifier at token offset 050) to be set to X'00' (no common control specification).	
DKPINOP	Specifies that this key may be used as a general-purpose key. It may not be used as a special-purpose key.
DKPINOPP	Specifies that this key is to be used to encrypt a PBF-1 format pin block for the specific purpose of creating a PIN mailer.
DKPINAD1	Specifies that this key may be used to create or verify a pin block to allow changing the account number associate with a PIN.
NOFLDFMT	Specifies that this key has no field format identifier.
PIN block format (one, required if and only if common control keyword is NOFLDFMT). Key-usage field 4.	
ISO-4	Specifies that only ISO-4 PIN blocks can be wrapped by this key.
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.

Table 127. Rule array keywords for AES PINPROT keys (continued)	
Keyword	**Meaning**
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.

Figure 33 on page 329 shows all the valid keyword combinations and their defaults for AES key type KDKGENKY. For a description of these keywords, see Table 128 on page 330.

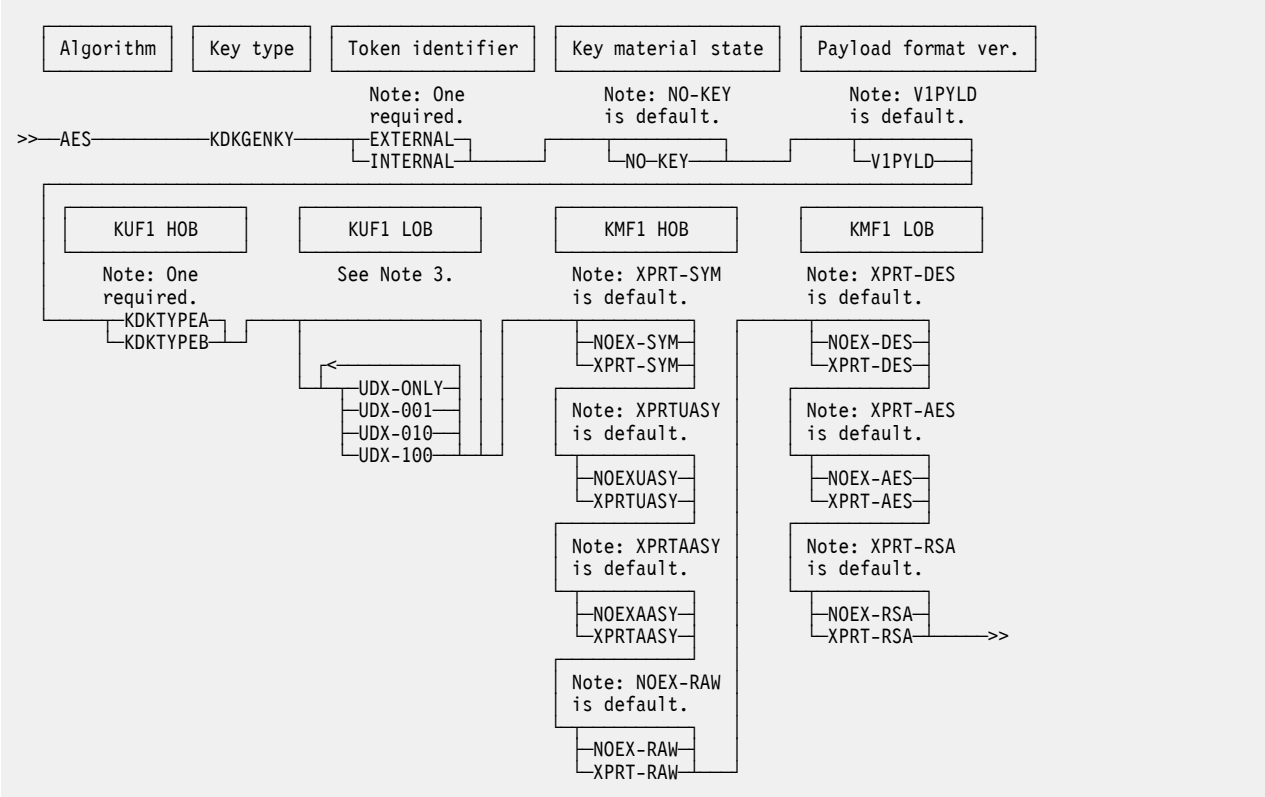


Figure 33. Key Token Build2 keyword combinations for AES KDKGENKY keys

Notes:

1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. Choose any number of keywords in this group. No keywords in the group are defaults.

Table 128. Rule array keywords for AES KDKGENKY keys

Keyword	Meaning
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
KDKGENKY	Key can be used for generating a diversified key.
Key diversification type (one required).	
KDKTYPEA	Entity A of communications partners A and B. One partner has active use of the key (for example, encipher or generate), while the other partner has passive use of the key (for example, decipher or verify).
KDKTYPEB	Entity B of communications partners A and B.
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.

Table 128. Rule array keywords for AES KDKGENKY keys (continued)	
Keyword	**Meaning**
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.

KDKGENKY key type usage notes

A KDKGENKY key is a key diversification key generating key used by the Diversify Directed Key (CSNBDDK) callable service to generate or derive a pair of AES directed keys, one at a time. A pair of directed keys has an active key and a complementary passive key. A KDKGENKY differs from a DKYGENKY in that it is used exclusively to generate or derive directed keys.

In addition to using a KDKGENKY generating key, CSNBDDK uses a key type vector (KTV) to determine which key type to generate or derive. CCA has eight KTVs defined (a pair for each of the four supported key types) that CSNBDDK can use. The KDKGENKY key and the KTV work together to determine which key type and direction (active or passive) of the diversified output key CSNBDDK generates or derives.

Based on the eight defined KTVs, the following CCA key types can be generated or derived using a KDKGENKY key:

CCA key type	Active key	Passive key
MAC (message authentication)	Key-usage keyword: GENONLY KDKGENKY KDKTYPEA and KTVM1 or KDKGENKY KDKTYPEB and KTVM2	Key-usage keyword: VERIFY KDKGENKY KDKTYPEB and KTVM1 or KDKGENKY KDKTYPEA and KTVM2
CIPHER (data encryption)	Key-usage keyword: ENCRYPT KDKGENKY KDKTYPEA and KTVC1 or KDKGENKY KDKTYPEB and KTVC2	Key-usage keyword: DECRYPT KDKGENKY KDKTYPEB and KTVC1 or KDKGENKY KDKTYPEA and KTVC2
PINPROT (PIN encryption)	Key-usage keyword: ENCRYPT KDKGENKY KDKTYPEA and KTVP1 or KDKGENKY KDKTYPEB and KTVP2	Key-usage keyword: DECRYPT KDKGENKY KDKTYPEB and KTVP1 or KDKGENKY KDKTYPEA and KTVP2
EXPORTER (key wrapping)	Key-usage keyword: EXPTT31D KDKGENKY KDKTYPEA and KTVW1 or KDKGENKY KDKTYPEB and KTVW2	Key-usage keyword: N/A
IMPORTER (key wrapping)	Key-usage keyword: N/A	Key-usage keyword: IMPTT31D KDKGENKY KDKTYPEB and KTVW1 or KDKGENKY KDKTYPEA and KTVW2
Note: Entity A and Entity B must both use the same KTV to produce the correct active/passive key pair. The KTV is used as the initialization vector during the diversification process.		

A KDKGENKY key has defined a key-usage field 1 high-order byte (KUF1 HOB) that specifies diversification key entity type of Entity A or Entity B. One of the key type vectors for a key type specifies that Entity A is to generate or derive an active key and Entity B is to derive or generate a passive key.

When an active or passive directed key is generated or derived by CSNBDDK, it inherits the key usage attributes that are defined by the related key usage fields of the KDKGENKY generating key token. The key usage attributes for the type of key to diversify can be specified by using the *service_data* parameter.

Table 129 on page 333 defines all the keywords and their combinations that can be specified in the *service_data* parameter for an AES KDKGENKY key:

Table 129. Allowable keywords for AES KDKGENKY keys

Type of key to diversify	KUF1 HOB	KUF2 HOB	KUF2 LOB	KUF3 HOB	KUF3 LOB	KUF4 HOB	KUF4 LOB
D-MAC Active	GENONLY	CMAC	N/A	DKPINAD1 DKPINAD2 DKPINOP (see Note 2)	N/A	N/A	N/A
D-MAC Passive	VERIFY						
D-CIPHER Active	ENCRYPT	CBC	N/A	N/A	N/A	N/A	N/A
D-CIPHER Passive	DECRYPT						
D-PINPROT Active	ENCRYPT	CBC	CPINENC EPINGEN PINXLATE REFORMAT RFMT1TO4 (see Note 3)	NOFLDFMT	N/A	ISO-4	N/A
D-PINPROT Passive	DECRYPT		CPINGENA EPINVER PINXLATE REFORMAT (see Note 3)				
D-EXP Active	EXPTT31D	VARDRV-D	KEK-RAW (see Note 4)	WR-AES WR-DES (see Note 5)	N/A	WR-CARD WR-CVAR WR-DATA WR-KEK WR-PIN WRDERIVE (see Note 6)	N/A
D-IMP Passive	IMPTT31D						

Notes:

1. UDX usage keywords are not allowed as service data. Instead, UDX usage is inherited based on the UDX keywords specified in the rule array.
2. One optional, no default. If a keyword is specified, the same keyword must be specified for both the active key and the passive key.
3. At least one keyword must be selected from this group. No keywords in this group are defaults.
4. Keyword is optional. There is no default.
5. One or both keywords must be specified. It is an error if only WR-AES is specified for one of the active/passive keys and WR-AES is not specified for the other key. Likewise, it is an error if only WR-DES is specified for one of the active/passive keys and WR-DES is not specified for the other key.
6. WR-CARD, WR-DATA, WR-KEK, WR-PIN, and WRDERIVE are defaults unless one or more keywords in this group are specified.

Figure 34 on page 334 shows all the valid keyword combinations and their defaults for AES key type PINPRW. For a description of these keywords, see Table 130 on page 334.

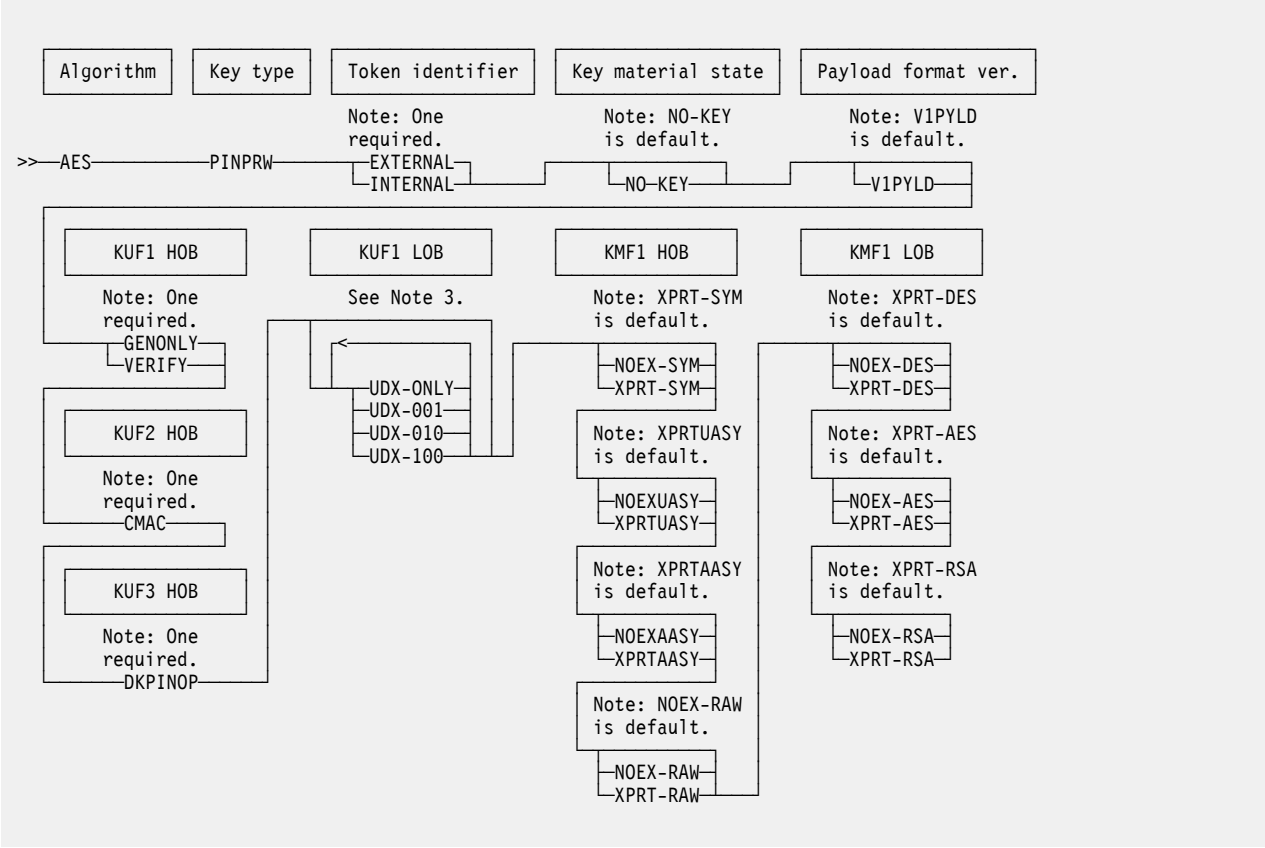


Figure 34. Key Token Build2 keyword combinations for AES PINPRW keys

Notes:

1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. Choose any number of keywords in this group. No keywords in the group are defaults.

Table 130. Rule array keywords for AES PINPRW keys

Keyword	Meaning
Key-token header section	
Token identifier (one required).	
EXTERNAL	Build a key token that is not to be used locally.
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	

Table 130. Rule array keywords for AES PINPRW keys (continued)	
Keyword	**Meaning**
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
PINPRW	Key can be used for generating and verifying PIN reference words.
Generate control (one required). Key-usage field 1, high-order byte.	
GENONLY	Specifies that this key can only be used to generate a PRW. It can not be used to verify a PRW.
VERIFY	Specifies that this key cannot be used to generate a PRW. It can only be used to verify a PRW.
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
Mode control (one, required). Key-usage field 2, high-order byte.	
CMAC	MAC calculation mode is block cipher-based MAC algorithm.
Common control (one, required). Key-usage field 3, high-order byte. Use of a common control keyword causes key-usage field 3, low-order byte (field format identifier at token offset 050) to be set to X'01' (DK enabled).	
DKPINOP	Specifies that this key may be used as a general-purpose key. It may not be used as a special-purpose key.
Symmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
RAW-key export control (one, optional). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.
DES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.

Table 130. Rule array keywords for AES PINPRW keys (continued)	
Keyword	**Meaning**
XPRT-DES	Permits the export of the key using DES key. This is the default.
AES-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
RSA-key export control (one, optional). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.

Figure 35 on page 336 shows all the valid keyword combinations and their defaults for AES key type SECMSG. For a description of these keywords, see Table 131 on page 337.

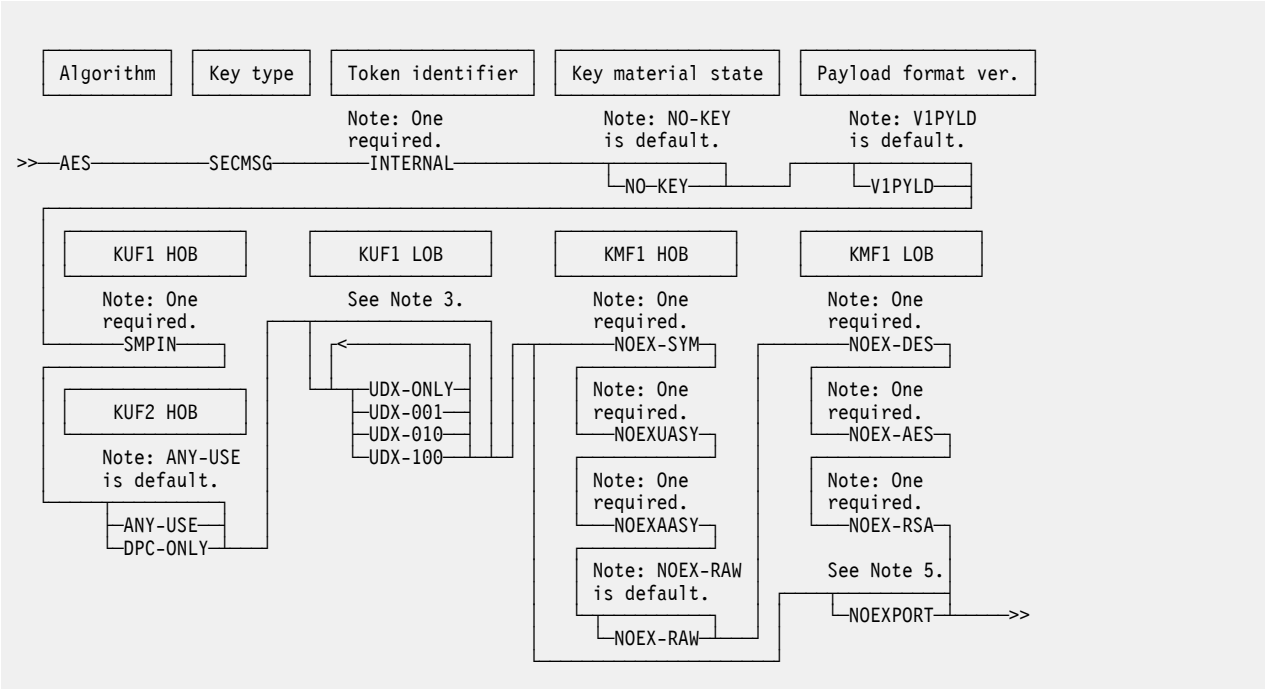


Figure 35. Key Token Build2 keyword combinations for AES SECMSG keys

Notes:

1. An AES SECMSG key is always derived. The derived key is the result of a key derivation function (KDF) applied to a fixed diversified key generating key (DKYGENKY) and derivation data. The final derived key is used as a session key and is typically used to encipher and decipher PIN information between devices. An AES SECMSG key can only be wrapped by an AES master key and cannot be stored in an external key-token.
2. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
3. Choose any number of keywords in this group. No keywords in the group are defaults.
4. NOEX-RAW is defined for future use and its meaning is currently undefined.
5. There is no default. Specifying NOEXPORT is equivalent to specifying all of the export control keywords (NOEX-SYM, NOEXUASY, NOEXAASY, NOEX-RAW, NOEX-DES, NOEX-AES, and NOEX-RSA). Do not specify any export control keywords together with NOEXPORT.

Table 131. Rule array keywords for AES SECMSG keys	
Keyword	**Meaning**
Key-token header section	
Token identifier (one required).	
INTERNAL	Build a key token that is to be used locally.
Wrapping-information section	
Key status (one, optional).	
NO-KEY	Build a key token that does not contain a key value. This is the default.
Payload format version (one, optional). Identifies format of the payload.	
V1PYLD	Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.
Associated data section	
Algorithm type (one required).	
AES	Key can be used for AES algorithm.
Key type (one required).	
SECMSG	Key can be used as an EMV secure messaging key for encrypting PINs or for encrypting keys.
Generate control (one required). Secure message encryption enablement (one required). Key-usage field 1, high-order byte.	
SMPIN	Enable the encryption of PINs in an EMV secure message.
User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.	
UDX-ONLY	Key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
Service restriction (one, optional). Key-usage field 2, high-order byte.	
ANY-USE	Any service can use this key. This is the default.
DPC-ONLY	Only CSNBDPC can use this key.
General export control (one, optional). Equivalent to specifying all export control keywords (NOEX-SYM, NOEXUASY, NOEXAASY, NOEX-RAW, NOEX-DES, NOEX-AES, and NOEX-RSA). Not valid with any other export control keyword. There is no default. Key-management field 1, high-order byte and low-order byte.	
NOEXPORT	Prohibits the export of this key in all cases. Equivalent to specifying NOEX-SYM, NOEXUASY, NOEXAASY, NOEX-RAW, NOEX-DES, NOEX-AES, and NOEX-RSA.
Symmetric-key export control (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, high-order byte.	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
Unauthenticated asymmetric-key export control (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, high-order byte.	

Table 131. Rule array keywords for AES SECMSG keys (continued)	
Keyword	**Meaning**
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
Authenticated asymmetric-key export control (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, high-order byte.	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
RAW-key export control (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, high-order byte.	
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
DES-key export control (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, low-order byte.	
NOEX-DES	Prohibits the export of the key using DES key.
AES-key export control (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, low-order byte	
NOEX-AES	Prohibits the export of the key using AES key.
RSA-key export control (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, low-order byte.	
NOEX-RSA	Prohibits the export of the key using RSA key.

service_data parameter

To use the *service_data* parameter to override the default key usage for any or all of the four active and four passive keys, follow these steps:

1. Start by reviewing the default key usage attributes for each key type along with the key usage requirements defined in the KTVs. For example, key usage restriction 1 of the two defined CIPHER KTVs requires that the CIPHER keys generate or derive a CBC mode key only.
2. For each key that is to be derived, make a list of the keywords required by CSNBKTB2 to build a skeleton key-token that has the desired key-usage attributes.

   Note: Do not specify any key management keywords because they are not included as part of the related key usage information in the generating key.
3. Use CSNBKTB2 to verify that each list of keywords can successfully produce a skeleton key-token. For each successful call to CSNBKTB2, consider the key type and key usage keywords specified in the *rule_array* as a group of keywords.

   Note: A pair of active/passive keys that has a mismatch of key-usage field counts is not usable. For each type of key to diversify, currently the KUF count of the active key must equal the KUF count of the passive key.
4. For each group of keywords, place the key type keyword at the beginning. Keywords AES and INTERNAL are excluded from the groups because they are not considered part of the key usage keywords to be included in the *verb_data* variable.
5. Count the number of remaining 8-byte keywords in all the groups, then multiply this total by 8. Set the *verb_data_length* variable to the result. The *verb_data_length* value must be a multiple of 8.
6. At this point, each group of keywords has the key type as the first keyword. Convert the key type keyword to a type of key to diversify keyword as follows:

Key type *keyword*	**Type of key to diversify keyword**
MAC	D-MAC

Key type *keyword*	Type of key to diversify keyword
CIPHER	D-CIPHER
PINPROT	D-PPROT
EXPORTER	D-EXP
IMPORTER	D-IMP

You will have 2, 4, 6, or 8 keyword groups, for example:

```
Keyword group 1:  D-PPROT ENCRYPT CBC NOFLDFMT  ISO-4 CPINENC
Keyword group 2:  D-PPROT DECRYPT CBC NOFLDFMT  ISO-4 EPINVER
```

7. Concatenate your ordered and converted groups of keywords in no particular order into the *verb_data* variable, for example (substitute space characters for the periods):

```
D-MAC...GENONLY.CMAC....
D-MAC...VERIFY..CMAC....
D-CIPHERENCRYPT.CBC.....
D-CIPHERDECRYPT.CBC.....
D-PPROT.ENCRYPT.CBC.....CPINENC.EPINGEN.PINXLATEREFORMATNOFLDFMTISO-4...
D-PPROT.DECRYPT.CBC.....CPINGENAEPINVER.PINXLATEREFORMATNOFLDFMTISO-4...
D-EXP...EXPTT31DVARDRV-DWR-AES..WR-CARD.WR-DATA.WR-KEK..WR-PIN..WRDERIVE
D-IMP...IMPTT31DVARDRV-DWR-AES..WR-CARD.WR-DATA.WR-KEK..WR-PIN..WRDERIVE
```

Consider the above eight lines as one contiguous string of 8-byte keywords in the *verb_data* variable. Rules for the *verb_data* variable of a KDKGENKY key are as follows:

- Each line in the example above represents a group of verb data keywords, either for an active key or a passive key. There must be 2, 4, 6, or 8 groups of keywords related to the type of key to diversify (related keywords) in the *verb_data* variable for a KDKGENKY key.
- Keyword groups can be concatenated in any order, but a group can only be included once. For example, there cannot be more than one D-CIPHER active group of keywords, or more than one D-CIPHER passive group of keywords.
- Each group specified must start with the KDKGENKY type of key to diversify keyword, that is, D-MAC for MAC, D-CIPHER for CIPHER, D-PPROT for PINPROT, D-EXP for EXPORTER, or D-IMP for IMPORTER. Each of these key type keywords acts as a delimiter, which enables the keywords in the *verb_data* variable to be parsed into their respective groups for conversion into their related KUF fields.
- Each keyword group must include a uni-directional keyword (not bi-directional) that indicates whether the key is active or passive. An active key can only be active, and a passive key can only be passive.

Type of key to diversify	Direction usage keyword required by active key	Direction usage keyword required by passive key
D-MAC	GENONLY (GENERATE not allowed)	VERIFY
D-CIPHER	ENCRYPT	DECRYPT
D-PPROT	ENCRYPT	DECRYPT
D-EXP	EXPTT31D	N/A
D-IMP	N/A	IMPTT31D

- Each keyword group must include the usage required by the key type vector for that key type:

Type of key to diversify	Usage keyword required by KTV for active key	Usage keyword required by KTV for passive key
D-MAC	CMAC	CMAC

Type of key to diversify	Usage keyword required by KTV for active key	Usage keyword required by KTV for passive key
D-CIPHER	CBC (ANY-MODE not allowed)	CBC
D-PPROT	ISO-4	ISO-4
D-EXP	VARDRV	N/A
D-IMP	N/A	VARDRV

CSNBKTB2 converts the type of key to diversify and its active or passive related key-usage keywords into an AES KDKGENKY active/passive related key-usage field block as defined in Table 132 on page 340. A fixed-length active/passive related KUF block is for a particular key type to be derived or generated, and consists of an 8-byte header, followed by 8 bytes of related active key usage fields for the type of key to diversify, followed by 8 bytes of related passive key usage fields for the type of key to diversify.

Each of these related KUF blocks gets included as part of the key usage fields of the KDKGENKY key.

Table 132. AES DKYGENKY and AES KDKGENKY active/passive related key-usage field block

Offset (bytes)	Length (bytes)	Description
KUF block header		
0	2	Key type of the key to be derived or generated (values match key type values defined in key type vector): **Value** **Meaning** **X'0000'** MAC (MAC) **X'0001'** Data encryption (CIPHER) **X'0003'** PIN encryption (PINPROT) **X'0004'** Key wrapping (EXPORTER or IMPORTER) All other values are reserved and undefined.
2	2	Underlying algorithm (values match underlying algorithm defined in key type vector): **Value** **Meaning** **X'0002'** AES All other values are reserved and undefined.

Table 132. AES DKYGENKY and AES KDKGENKY active/passive related key-usage field block (continued)

Offset (bytes)	Length (bytes)	Description
4	1	Block count of active related key usage fields, akuf (2 – 4). Count is based on type of active key to be derived or generated (offset 0) and underlying algorithm (offset 2): **Value at offset 0** **CCA key type *akuf*** **X'0000'** MAC 2 or 3 **X'0001'** CIPHER 2 **X'0003'** PINPROT 4 **X'0004'** EXPORTER 4
5	1	Block count of passive related key usage fields, pkuf (2 – 4). Count is based on type of passive key to be derived or generated (offset 0) and underlying algorithm (offset 2): **Value at offset 0** **CCA key type *akuf*** **X'0000'** MAC 2 or 3 **X'0001'** CIPHER 2 **X'0003'** PINPROT 4 **X'0004'** IMPORTER 4
6	1	KUF block format version: **Value** **Meaning** **X'01'** Version 1 All other values are reserved and undefined.
7	1	Reserved, binary zeros.
Active key related KUF fields for the type of key to diversify.		
8	2	Key-usage field 1, high-order byte (KUF1 HOB) and low-order byte (KUF1 LOB) of active related key for the type of key to diversify. This field is the same as the one defined for the key identified in the block header.
10	2	Key-usage field 2, high-order byte (KUF2 HOB) and low-order byte (KUF2 LOB) of active related key for the type of key to diversify. This field is the same as the one defined for the key identified in the block header.

Table 132. AES DKYGENKY and AES KDKGENKY active/passive related key-usage field block (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
12	2	For akuf < 3, reserved, binary zero. Otherwise: Key-usage field 3, high-order byte (KUF3 HOB) and low-order byte (KUF3 LOB) of active related key for the type of key to diversify. This field is the same as the one defined for the key identified in the block header.
14	2	For akuf < 4, reserved, binary zero. Otherwise: Key-usage field 4, high-order byte (KUF4 HOB) and low-order byte (KUF4 LOB) of active related key for the type of key to diversify. This field is the same as the one defined for the key identified in the block header.
Passive key related KUF fields for the type of key to diversify.		
16	2	Key-usage field 1, high-order byte (KUF1 HOB) and low-order byte (KUF1 LOB) of passive related key for the type of key to diversify. This field is the same as the one defined for the key identified in the block header.
18	2	Key-usage field 2, high-order byte (KUF2 HOB) and low-order byte (KUF2 LOB) of passive related key for the type of key to diversify. This field is the same as the one defined for the key identified in the block header.
20	2	For pkuf < 3, reserved, binary zero. Otherwise: Key-usage field 3, high-order byte (KUF3 HOB) and low-order byte (KUF3 LOB) of passive related key for the type of key to diversify. This field is the same as the one defined for the key identified in the block header.
22	2	For pkuf < 4, reserved, binary zero. Otherwise: Key-usage field 4, high-order byte (KUF4 HOB) and low-order byte (KUF4 LOB) of passive related key for the type of key to diversify. This field is the same as the one defined for the key identified in the block header.

An AES KDKGENKY key-token is defined to have one key-usage field (KUF) followed by KUFs related to the keys to be derived or generated. These related KUFs are grouped into one to four active/passive related KUF blocks, one for each key type. The active and passive KUFs are used to define the usage of an active or passive key when it is derived or generated by CSNBDDK.

The format of the key usage fields of an AES KDKGENKY are shown in Table 133 on page 342:

Table 133. AES KDKGENKY key usage fields format													
KDKGENKY normal KUF		**Related KUFs starting with KUF2 to end of KUFs**											
KUF1 (KDKTYPEA or KDKTYPEB, UDX)		KUF block related to active and passive MAC keys to be derived or derived			KUF block related to active and passive data encryption (CIPHER) keys to be derived or generated			KUF block related to active and passive PIN encryption (PINPROT) keys to be derived or generated			KUF block related to active and passive key wrapping (EXPORTER and IMPORTER) keys to be derived or generated		
HOB	LOB	Block header	Active KUFs	Passive KUFs	Block header	Active KUFs	Passive KUFs	Block header	Active KUFs	Passive KUFs	Block header	Active KUFs	Passive KUFs

When an AES DKYGENKY key-token has a type of key to diversify of D-KDKGKY (KUF1 HOB valued to X'09'), it is defined to have two key-usage fields followed by one to four active/passive related KUF blocks starting with KUF3. The related KUF fields starting at KUF1 in Table 133 on page 342 are equivalent to the KUF fields starting at KUF3 in Table 134 on page 343Table 15:

Table 134. AES DKYGENKY key usage fields format

DKYGENKY normal KUFs		**DKYGENKY related KUFs starting with KUF3 to end of KUFs**													
		KUF1 related to KDKGENKY key		KUFs related to MAC, CIPHER, PINPROT, and key wrapping keys to be derived or generated starting with KUF2 to end of KUFs											
KUF1	KUF2	KUF3 – start of related KUFs		MAC active/passive related KUF block			Data encryption (CIPHER) active/passive related KUF block			PIN encryption (PINPROT) active/passive related KUF block			Key wrapping (EXPORTER and IMPORTER) active/passive related KUF block		
HOB/LOB	HOB/LOB	HOB	LOB	Block header	Active KUFs	Passive KUFs	Block header	Active KUFs	Passive KUFs	Block header	Active KUFs	Passive KUFs	Block header	Active KUFs	Passive KUFs

Required hardware

No cryptographic hardware is required by this callable service.

Key Translate (CSNBKTR and CSNEKTR)

Note: This service has been deprecated. New applications should use the Key Translate2 service that is described in "Key Translate2 (CSNBKTR2 and CSNEKTR2)" on page 346 instead of this service.

The Key Translate callable service uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

Note: All key labels must be unique.

The callable service name for AMODE(64) invocation is CSNEKTR.

Format

```
CALL CSNBKTR(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             input_key_token,
             input_KEK_key_identifier,
             output_KEK_key_identifier,
             output_key_token )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

input_key_token

Direction	Type
Input	Integer

A 64-byte string variable containing an external key token. The external key token contains the key to be re-enciphered (translated).

input_KEK_key_identifier

Direction	Type
Input/Output	String

A 64-byte string variable containing the internal key token or the key label of an internal key token record in the CKDS. The internal key token contains the key-encrypting key used to decipher the key. The internal key token must contain a control vector that specifies an importer or IKEYXLAT key type. The control vector for an importer key must have the XLATE bit set to 1.

output_KEK_key_identifier

Direction	Type
Input/Output	String

A 64-byte string variable containing the internal key token or the key label of an internal key token record in the CKDS. The internal key token contains the key-encrypting key used to encipher the key. The internal key token must contain a control vector that specifies an exporter or OKEYXLAT key type. The control vector for an exporter key must have the XLATE bit set to 1.

output_key_token

Direction	Type
Output	String

A 64-byte string variable containing an external key token. The external key token contains the re-enciphered key.

The *output_key_token* will be wrapped in the same manner as the *input_key_token*.

The DES key wrapping methods available are described in “CCA key wrapping” on page 20.

Restrictions

Triple length DATA key tokens are not supported.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

The **Key Translate** access control point controls the function of this service.

If the output key-encrypting key identifier is a weaker key than the key being translated, then:

- The service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- The service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the output key-encrypting key identifier key is a 16-byte key.

When the **Disallow translation from DES wrapping to weaker DES wrapping** access control point is enabled, this service fails if the input key-encrypting key identifier is strong than the output key-encrypting key identifier.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 135. Key Translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).

Table 135. Key Translate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Key Translate2 (CSNBKTR2 and CSNEKTR2)

The Key Translate2 callable service translates the *input_key_token* parameter in one of several ways:

- Changes an external DES or variable-length symmetric key token from encipherment under one key-encrypting key to another.
- Changes the wrapping method of an external DES key token.
- Converts an operational AES DATA token (version X'04') to an operational AES CIPHER token (version X'05') or converts an operational AES CIPHER token (version X'05') to an operational AES DATA token (version X'04').
- Converts a key token using the AESKW wrapping method from a variable length payload to a fixed length payload.
- Checks the compliance of a key token.
- Converts a key token from a non-compliant-tagged key token to a compliant-tagged key token.

To reencipher a key token, specify the TRANSLAT rule array keyword (the default), the external key token, and the input and output key-encrypting keys. If the *input_key_token* is a DES key token, you can also specify which key wrapping method to use. If no wrapping method is specified, the system default wrapping method will be used.

To change the wrapping method of an external DES key token, specify the REFORMAT rule array keyword, the Key Wrapping Method to use, the external key token and the input key-encrypting key. If no wrapping method is specified, the system default wrapping method will be used. Note that the *output_KEK_identifier* will be ignored.

To convert an operational AES DATA token (version X'04') to an operational AES CIPHER token (version X'05') or vice versa, specify the REFORMAT rule array keyword, the operational key token as *input_key_token*, and either a NULL token or skeleton token as *output_key_token*. Note that both the *input_KEK_identifier* and the *output_KEK_identifier* will be ignored as the corresponding lengths must be 0.

To convert an internal or external variable-length AES key token (version X'05') from a variable-length payload to a fixed-length payload, specify the V1PYLD rule array keyword. The fixed-length payload will obfuscate the key length. This keyword is only valid for the CIPHER, EXPORTER and IMPORTER key types.

To convert an internal or external variable-length AES key token (version X'05') from a fixed-length payload to a variable-length payload, specify the V0PYLD rule array keyword. This keyword is only valid for the CIPHER, EXPORTER and IMPORTER key types.

To check the compliance of a key token, specify the COMP-CHK rule array keyword.

To convert a key token from a non-compliant-tagged key token to a compliant-tagged key token, specify the COMP-TAG rule array keyword.

Notes:

- All key labels must be unique.
- The CSNBKTR2 service provides functions for CCA key tokens. The CSNBT31X service (see "TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)" on page 1007) provides similar functions for X9.143 (TR-31) key blocks.

The callable service name for AMODE(64) invocation is CSNEKTR2.

Format

```
CALL CSNBKTR2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              input_key_length,
              input_key_token,
              input_KEK_length,
              input_KEK_identifier,
              output_KEK_length,
              output_KEK_identifier,
              output_key_length,
              output_key_token )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is defined in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The count must be between 0 and 4, inclusive.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Keyword	Meaning
Encipherment (One optional)	
COMP-CHK	Check if the *input_key_token* can have the compliant tag.
COMP-TAG	Convert the *input_key_token* into a compliant-tagged token.
REFORMAT	Reformat the *input_key_token*. • When *input_key_token* is a DES key token, reformat with the Key Wrapping Method specified. • When *input_key_token* is an operational AES key token, either reformat an AES DATA key (version X‘04’) to an AES CIPHER key (version X‘05’) or the reverse (version X’05’ to version X’04’).
TRANSLAT	Translate the *input_key_token* from encipherment under the *input_KEK_identifier* to encipherment under the *output_KEK_identifier*. This is the default.
V1PYLD	Reencipher an input variable-length AES key token (version X‘05’) to a payload version 1 (fixed-length) key token. This keyword is only valid for the CIPHER, EXPORTER and IMPORTER key types.
V0PYLD	Reencipher an input variable-length AES key token (version X‘05’) to a payload version 0 (variable-length) key token. This keyword is only valid for the CIPHER, EXPORTER and IMPORTER key types.
Key Wrapping Method (optional, valid only if input_key_token is an external DES key token)	
USECONFG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default. The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the *z/OS Cryptographic Services ICSF System Programmer's Guide*.
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.

Keyword	Meaning
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This method requires ENH-ONLY.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens.
Translation Control (optional)	
ENH-ONLY	Restrict rewrapping of the *key_identifier*. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method. This is the default when the wrapping method is WRAPENH2 or WRAPENH3.
Algorithm (One required, if the V0PYLD or V1PYLD keyword is specified)	
AES	Specifies that the input key is an AES key. Where used, the key-encrypting keys will be AES transport keys.
DES	Specifies that the input key is a DES key. Where used, the key-encrypting keys will be DES transport keys. This is the default.
HMAC	Specifies that the input key is an HMAC key. Where used, the key-encrypting keys will be AES transport keys.

input_key_length

Direction	Type
Input	Integer

The length of the *input_key_token* in bytes. The maximum value allowed is 900.

input_key_token

Direction	Type
Input/Output	String

A variable length string variable containing the key token to be translated or reformatted.

If the REFORMAT keyword is specified and the input_key_token is an AES CIPHER key (version X‘05’), the key must have the following characteristics:

- Key-usage field 1 allows the key to be used for encryption and decryption and has no UDX bits set (UDX bits are not supported in version ‘04’X AES tokens).
- Key-usage field 2 allows the key to be used for Cipher Block Chaining (CBC) mode or Electronic Code Book (ECB) mode.
- Key-management field 1 allows export using symmetric, unauthenticated asymmetric, and authenticated asymmetric transport keys, and allows export using DES, AES, and RSA transport keys.
- Key-management field 2 indicates that the key is complete.

If the COMP-TAG keyword is specified, this parameter must be an internal DES key token or an internal version 05 AES key token.

If the COMP-CHK keyword is specified, this parameter must be an internal DES key token, an internal version 05 AES key token, or the label of either.

If either of the COMP-CHK or COMP-TAG keywords is specified and *input_key_token* was encrypted under the old master key, the token is returned encrypted under the current master key.

If the REFORMAT and AES keywords are specified and *input_key_token* was encrypted under the old master key, the token is returned encrypted under the current master key.

input_KEK_length

Direction	Type
Input	Integer

The length of the *input_KEK_identifier* in bytes.

When the *input_KEK_identifier* is a label, the value must be 64.

When the *input_KEK_identifier* is a token, the value must be between the actual length of the token and 9992.

When the REFORMAT, V0PYLD, or V1PYLD keywords are specified and the *input_key_token* parameter contains internal key or the label of an operation key, this value must be 0.

When the COMP-TAG or COMP-CHK keyword is specified, this value must be 0.

input_KEK_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to unwrap the input key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage

If *input_KEK_length* is 0, this parameter is ignored.

If the TRANSLAT keyword is specified and the *input_key_token* is an external DES key:

- For CCA keys, the key identifier is a 64-byte DES key token of key type IMPORTER with the XLATE key usage must be enabled in the control vector or key type IKEYXLAT.
- For X9.143 (TR-31) keys, the key identifier is a variable-length key block of a DES key-encrypting key: key usage K0, algorithm T, and mode of use D.

When the TRANSLAT keyword is specified and the *input_key_token* is an external AES key:

- For CCA keys, the key identifier is a variable-length AES key token of key type IMPORTER with the TRANSLAT key usage attribute enabled.
- For X9.143 (TR-31) keys, the key identifier is a variable-length key block of an AES key-encrypting key: key usage K0, algorithm A, and mode of use D.

When the V0PYLD or V1PYLD keyword is specified and the *input_key_token* is an external variable-length CCA key token:

- For CCA keys, the *input_KEK_identifier*is an internal CCA variable-length key token containing an IMPORTER key-encrypting key. The IMPORTER key must have the TRANSLAT bit on in key-usage field 1 of the token.
- For X9.143 (TR-31) keys, the key identifier is a variable-length key block of an AES key-encrypting key: key usage K0, algorithm A, and mode of use D.

If the REFORMAT keyword is specified and *input_key_token* is an external DES CCA key token:

- For CCA keys, the key identifier may be a CCA IMPORTER, IKEYXLAT, EXPORTER, or OKEYXLAT key type.
- For X9.143 (TR-31) keys, the key identifier is a variable-length key block of a DES key-encrypting key: key usage K0, algorithm T, and mode of use D or E.

If an internal token or block was supplied and was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

output_KEK_length

Direction	Type
Input	Integer

The length of the *output_KEK_identifier* in bytes.

When the *output_KEK_identifier* is a label, the value must be 64.

When the *output_KEK_identifier* is a token, the value must be between the actual length of the token and 9992.

If the REFORMAT, V0PYLD, or V1PYLD keyword is specified, this value must be 0.

If the COMP-CHK or COMP-TAG keyword is specified, this parameter must be 0.

output_KEK_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to wrap the output key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

If *output_KEK_length* is 0, this parameter is ignored.

If the *output_key_token* contains an external key, the parameter contains the wrapping key:

- For CCA DES keys, the key identifier is a 64-byte DES key token of key type EXPORTER with the XLATE key usage enabled in the control vector or key type OKEYXLAT.
- For CCA AES and HMAC keys, the wrapping key must contain an AES EXPORTER with the TRANSLAT key usage attribute enabled.
- For TR-31 keys, the wrapping key must be K0 key usage. The algorithm is A or T matching the algorithm of the output key and the mode of use is E.

If the token or block was supplied and was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

output_key_length

Direction	Type
Input/Output	Integer

On input, the length of the output area provided for the *output_key_token*. This must be between 64 and 900 bytes and provide sufficient space for the output key. On output, the parameter is updated with the length of the token copied to the *output_key_token*.

If the COMP-CHK keyword is specified, this parameter must be 0.

output_key_token

Direction	Type
Input/Output	String

If the REFORMAT keyword is specified and the *input_key_token* is an AES DATA key (version X‘04’), *output_key_token* must contain an AES CIPHER key (version X‘05’) on input. The algorithm rule array keyword must specify AES and the token must have the following characteristics:

- Algorithm is AES.

- Key type CIPHER.
- Key-usage field 2 either allows the key to be used for Cipher Block Chaining (CBC) mode or allows the key to be used for Electronic Code Book (ECB) mode.
- (Optional) The CPACF export key management bit may be enabled. The CPACF export status is copied to the output key token.

Otherwise, this field is ignored on input.

On output, a variable length string variable containing the key token that was translated or reformatted.

If the REFORMAT keyword is specified and the *input_key_token* is an AES DATA key (version X'04'), on output, *output_key_token* will be updated with the following characteristics:

- Key-usage field 1 allows the key to be used for encryption and decryption.
- Key-management field 1 allows export using symmetric, unauthenticated asymmetric, and authenticated asymmetric transport keys, and allows export using DES, AES, and RSA transport keys.
- Key-management field 2 indicates that the key is complete.

Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

This table lists the access control points in the domain role that control the function for this service.

Table 136. Key Translate2 Access Control Points

Access Control point	Function control
Key Translate2	Allows the Key Translate2 service to be functional.
Key Translate2 – Allow use of REFORMAT	Allows a key token to be rewrapped using one key-encrypting key.
Key Translate2 – Allow wrapping method override keywords	Allows a key wrapping method keyword to specify a wrapping method that is not the default method.
Key Translate2 – COMP-CHK	Allows the COMP-CHK keyword to be used.
Key Translate2 – COMP-TAG	Allows the COMP-TAG keyword to be used.
Key Translate2 – Translate fixed to variable payload	Allows a key token with a fixed-length payload to be re-enciphered with a variable-length payload (V0PYLD).

When the **Key Translate2 - Disallow AES ver 5 to ver 4 conversion** access control point is enabled, a version 5 AES key token (variable-length token) cannot be converted to a version 4 token.

When the **Disallow translation from AES wrapping to weaker AES wrapping** access control point is enabled, this service fails if the input key-encrypting key identifier is stronger than the output key-encrypting key identifier.

When the **Disallow translation from DES wrapping to weaker DES wrapping** access control point is enabled, this service fails if the input key-encrypting key identifier is stronger than the output key-encrypting key identifier.

When the **Disallow translation from AES wrapping to DES wrapping** access control point is enabled, this service fails if the input key-encrypting key identifier is an AES key and the output key-encrypting key identifier is a DES key.

If the output key-encrypting key identifier is a weaker key than the key being translated, then:

- The service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- The service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the input key is a triple-length DATA key and the output key-encrypting key identifier key is a 16-byte key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 137. Key Translate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords COMP-CHK and COMP-TAG are not supported. Compliant-tagged key tokens are not supported. Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 137. Key Translate2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Keywords COMP-CHK and COMP-TAG are not supported. Compliant-tagged key tokens are not supported. Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	The COMP-TAG keyword requires a coprocessor in compliance migration mode. Keywords COMP-CHK and COMP-TAG with a version 05 AES token require the July 2019 or later licensed internal code (LIC). Using the COMP-TAG keyword to migrate a DES KDF 01 token to a compliant-tagged token, requires the July 2019 or later licensed internal code (LIC). Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged AES key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Keywords COMP-CHK and COMP-TAG are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 137. Key Translate2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Multiple Clear Key Import (CSNBCKM and CSNECKM)

The Multiple Clear Key Import callable service imports a clear AES or DES key, enciphers the key under the corresponding master key, and returns the enciphered key in an internal key token. The enciphered key's type is DATA. The enciphered key is in operational form.

The callable service name for AMODE(64) invocation is CSNECKM.

Format

```
CALL CSNBCKM(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              clear_key_length,
              clear_key,
              key_identifier_length,
              key_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 0, 1, 2, or 3. If the *rule_array_count* is 0, the default keywords are used.

rule_array

Direction	Type
Input	String

Keywords that supply control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.Refer to Table 138 on page 356 for a list of keywords.

Table 138. Keywords for Multiple Clear Key Import Rule Array Control Information

Keyword	Meaning
Algorithm (optional)	
AES	The output key identifier is to be an AES token.
DES	The output key identifier is to be a DES token. This is the default.
Key Wrapping Method (optional). Valid for DES keys only.	
USECONFG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword. The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the *z/OS Cryptographic Services ICSF System Programmer's Guide*.
WRAP-ENH	Use enhanced key wrapping method with SHA-1, which is compliant with the ANSI X9.24 standard.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method with SHA-256. Valid only with triple-length keys. This is the default for triple-length keys.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method with SHA-256 and CMAC authentication code.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens.
Translation Control (optional)	

Table 138. Keywords for Multiple Clear Key Import Rule Array Control Information (continued)	
Keyword	**Meaning**
ENH-ONLY	Restrict rewrapping of the *key_identifier* token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method. This is the default when the wrapping method is WRAPENH2 or WRAPENH3.

clear_key_length

Direction	Type
Input	Integer

The *clear_key_length* specifies the length of the clear key value to import in bytes. For DES keys, this length must be 8-, 16-, or 24-bytes. For AES keys, this length must be 16-, 24-, or 32-bytes.

clear_key

Direction	Type
Input	String

The *clear_key* specifies the clear key value to import.

key_identifier_length

Direction	Type
Input/Output	Integer

The byte length of the *key_identifier* parameter. This must be exactly 64 bytes.

key_identifier

Direction	Type
Output	String

A 64-byte string that is to receive an internal AES or DES key token.

The output *key_identifier* will use the default method unless a rule array keyword overriding the default is specified.

The DES key wrapping methods available are described in “CCA key wrapping” on page 20.

Usage notes

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 139. Required access control points for Multiple Clear Key Import	
Key algorithm	**Access control point**
DES	Clear Key Import/Multiple Clear Key Import – DES
AES	Multiple Clear Key Import/Multiple Secure Key Import – AES

When the key wrapping method keyword specifies a wrapping method that is not the default method, the **Multiple Clear Key Import - Allow wrapping override keywords** access control point must be enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 140. Multiple Clear Key Import required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express7 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Multiple Secure Key Import (CSNBSKM and CSNESKM)

Use this service to encipher a single-length, double-length, or triple-length DES key under the system master key or an importer key-encrypting key. The clear DES key can then be imported as any of the possible key types.

In addition to DES keys, this service imports a clear AES key, enciphers the AES key under the AES master key, and returns the enciphered key in an internal token. The enciphered key's type is DATA. The enciphered key is in operational form.

The callable service can execute only when ICSF is in special secure mode, which is described in "Special secure mode" on page 10.

The callable service name for AMODE(64) invocation is CSNESKM.

Format

```
CALL CSNBSKM(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              clear_key_length,
              clear_key,
              key_type,
              key_form,
              key_encrypting_key_identifier,
              imported_key_identifier_length,
              imported_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 0, 1, 2, 3, 4, or 5. If the *rule_array_count* is 0, the default keywords are used.

rule_array

Direction	Type
Input	String

Keywords that supply control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The keywords are shown in Table 141 on page 360.

The first keyword is the algorithm. If no algorithm is specified, DES is used.

The second keyword is optional and specifies that the output key-encrypting key token be marked as an NOCV-KEK.

The third keyword is optional, and specifies whether the original key wrapping method or the enhanced key wrapping method (which is compliant with the ANSI X9.24 standard) should be used. Note that triple-length keys with a control vector are wrapped with the WRAPENH2 method.

The fourth keyword enables an application to specify that the *imported_key_identifier* output token cannot be rewrapped using the original wrapping method after it has been wrapped using the enhanced method.

Table 141. Keywords for Multiple Secure Key Import Rule Array Control Information

Keyword	Meaning
Algorithm (optional)	
AES	The output key identifier is to be a AES token.
DES	The output key identifier is to be a DES token. This is the default.
DES control vector (optional, valid with algorithm DES, not valid with key type TOKEN)	
UNIQUE	Specifies that the control vector for DES keys is marked as having unique key parts. The clear key value must have two or three unique key values. **Note:** When this keyword is specified with a triple-length DATA key, the key token has the DATA control vector with the guaranteed unique key form bit on.
DATA control vector (optional, valid with triple-length DES DATA keys only)	
DATA-CV	Specifies that the triple-length DES DATA key has the default control vector. The key type must be DATA.
NOCV Choice (optional, valid with DES IMPORTER and EXPORTER only)	
NOCV-KEK	The output token is to be marked as an NOCV-KEK. This keyword is valid when the key form is OP and key type is IMPORTER or EXPORTER.
Key Wrapping Method (optional, valid with the DES algorithm only) **Note:** Triple-length DES keys will always be wrapped with the enhanced method using SHA-256 and the ENH-ONLY control vector bit will be enabled.	

Table 141. Keywords for Multiple Secure Key Import Rule Array Control Information (continued)	
Keyword	**Meaning**
USECONFG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword. The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set.
WRAP-ENH	Use enhanced key wrapping method using SHA-1, which is compliant with the ANSI X9.24 standard.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This is the default for triple-length keys.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.
Translation Control (optional, valid with the DES algorithm only)	
ENH-ONLY	Restrict rewrapping of the *imported_key_identifier* token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method. This is the default when the wrapping method is WRAPENH2 or WRAPENH3.

clear_key_length

Direction	**Type**
Input	Integer

The *clear_key_length* specifies the length of the clear key value to import in bytes. For AES keys, this length must be 16-, 24-, or 32-bytes. For DES keys, this length must be 8-, 16- or 24-bytes.

clear_key

Direction	**Type**
Input	String

The *clear_key* specifies the AES or DES clear key value to import.

When the UNIQUE keyword is specified for DES keys, the 8-byte values supplied must be unique.

key_type

Direction	**Type**
Input	8 Character String

The type of key you want to encipher under the master key or an importer key. Specify an 8-byte field that must contain a keyword from this list or the keyword TOKEN. For types with fewer than 8 characters, the type should be padded on the right with blanks. If the key type is TOKEN, ICSF determines the key type from the control vector (CV) field in the internal key token provided in the *imported_key_identifier* parameter. When key_type is TOKEN, ICSF does not check for the length of the key but uses the *clear_key_length* parameter to determine the length of the key.

The DES key types are: CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR, DATA, DATAM, DATAMV, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IMP-PKA, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER. DATA is the only key type for AES.

key_form

Direction	Type
Input	4 Character String

The key form you want to generate. Enter a 4-byte keyword specifying whether the key should be enciphered under the master key (OP) or the importer key-encrypting key (IM). The keyword must be left-justified and padded with blanks. Valid DES keyword values are OP for encryption under the master key or IM for encryption under the importer key-encrypting key. If you specify IM, you must specify an importer key-encrypting key in the *key_encrypting_key_identifier* parameter. For a *key_type* of IMP-PKA, this service supports only the OP *key_form*.

The only valid AES keyword value is OP.

key_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to wrap the imported key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte DES key token of key type IMPORTER.

For X9.143 keys, the identifier is a variable-length key block of a TDES key-encrypting key usage K0, algorithm T, and mode of use D.

This parameter is ignored when importing an AES secure key.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

imported_key_identifier_length

Direction	Type
Input/Output	Integer

The byte length of the *imported_key_identifier* parameter. This must be at least 64.

imported_key_identifier

Direction	Type
Input/Output	String

On input, this parameter contains a 64-byte key token.

- When the *key_type* parameter is TOKEN, a key token that ICSF will use to determine the key type and key length:
  - A skeleton token of the key type to be imported. When the WRAPENH3 method is selected, a skeleton key token is required.
  - An internal or external token of the key type to be imported.
- Otherwise, a null token.

On output, this parameter receives the imported key token:

- Internal DES or AES key token for an operational key form, or
- External DES key tokens containing a key enciphered under the *key_encrypting_key_identifier* parameter.

The output *imported_key_identifier* will use the default method unless a rule array keyword overriding the default is specified.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Creation of a DES NOCV key-encrypting key is only available for IMPORTERs and EXPORTERs with the default control vector.

For key types CIPHERXI, CIPHERXL, and CIPHERXO, the key-encrypting key in the key_encrypting_key_identifier parameter must have a control vector with the key halves guaranteed unique flag on in the key form bits. An existing key-encrypting key can have its control vector updated using the restrict key attribute callable service.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 142. Required access control points for Multiple Secure Key Import

Key Algorithm and Key Form	Access control point
DES OP	Secure Key Import - DES, OP
DES IM	Secure Key Import - DES, IM
AES OP	Multiple Clear Key Import/Multiple Secure Key Import - AES

When the key wrapping method keyword specifies a wrapping method that is not the default method, the **Multiple Secure Key Import - Allow wrapping override keywords** control must be enabled.

To use a NOCV key-encrypting key with the Multiple Secure Key Import service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

If the key-encrypting key identifier is a weaker key than the key being imported, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key or the key-encrypting key is a double-length key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 143. Multiple Secure Key Import required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in the output key token is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). The CPACF export bit (XPRTCPAC) in output DES CIPHER key tokens requires a CEX6C with the P41458.002 or later MCL. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 143. Multiple Secure Key Import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	The CPACF export bit (XPRTCPAC) in the output key token is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

PKA Decrypt (CSNDPKD and CSNFPKD)

Use this service to decrypt (unwrap) a formatted key value. The service unwraps the key, parses it, and returns the parsed value to the application in the clear. PKCS 1.2, RSAES-OAEP, and ZERO-PAD formatting is supported. For PKCS 1.2, the decrypted data is examined to ensure it meets RSA DSI PKCS #1 block type 2 format specifications.

For PKA private keys, this service allows the use of clear or encrypted RSA or CRYSTALS-Kyber private keys. If an external clear key token is used, the master keys are not required to be installed in any cryptographic coprocessor and PKA callable services does not have to be enabled. Requests are routed to a Cryptographic Accelerator if available when an RSA clear key token is used. ZERO-PAD is only supported for external RSA clear private keys or CRYSTALS-Kyber private keys.

This service also supports the use of secure PKCS #11 RSA private keys, which requires an active Enterprise PKCS #11 coprocessor. PKCS 1.2 formatting is supported.

The callable service name for AMODE(64) invocation is CSNFPKD.

Format

```
CALL CSNDPKD(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              PKA_enciphered_keyvalue_length,
              PKA_enciphered_keyvalue,
```

```
                data_structure_length,
                data_structure,
                key_identifier_length,
                key_identifier,
                target_keyvalue_length,
                target_keyvalue)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1.

rule_array

Direction	Type
Input	String

The keyword that provides control information to the callable service. The keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 144. Keywords for PKA Decrypt	
Keyword	**Meaning**
Recovery Method (required) specifies the method to use to recover the key value.	
PKCS-1.2	RSA PKCS #1 V1.5 block type 02 will be used to recover the key value.
PKCSOAEP	Specifies to recover the data formatted using the RSAES-OAEP encoding scheme defined in the RSA PKCS #1 v2.0 standard. This keyword is not valid when using a secure PKCS #11 private key.
PKOAEP2	Specifies that the key is formatted as defined in the RSA PKCS #1 v2.1 standard for the RSAES-OAEP encryption/decryption scheme.
ZERO-PAD	The input *PKA_enciphered_keyvalue* is decrypted using the private key. The entire result (including leading zeros) will be returned in the *target_keyvalue* field. For PKA keys, the *key_identifier* must be an external RSA or CRYSTALS-Kyber token or the label of a token. This keyword is not valid when using a secure PKCS #11 private key. Required when the *PKA_key_identifier* is a CRYSTALS-Kyber private key.
Hash Method (one required when PKCSOAEP or PKOAEP2 is specified. Otherwise, not allowed.)	
SHA-1	Specifies to use the SHA-1 method to calculate the OAEP message digest.
SHA-224	Specifies to use the SHA-224 method to calculate the OAEP message digest. Only supported with PKOAEP2.
SHA-256	Specifies to use the SHA-256 method to calculate the OAEP message digest.
SHA-384	Specifies to use the SHA-384 method to calculate the OAEP message digest. Only supported with PKOAEP2.
SHA-512	Specifies to use the SHA-512 method to calculate the OAEP message digest. Only supported with PKOAEP2.

PKA_enciphered_keyvalue_length

Direction	Type
Input	Integer

The length of the *PKA_enciphered_keyvalue* parameter in bytes. The maximum size that can be generated when the *key_identifier* is an RSA key is 512 bytes. The length should be the same as the modulus length of the *key_identifier*.

When the *key_identifier* is a CRYSTALS-Kyber private key, the maximum size is 1568 bytes.

PKA_enciphered_keyvalue

Direction	Type
Input	String

This field contains the key value protected under a public key. This byte-length string is left-justified within the *PKA_enciphered_keyvalue* parameter.

data_structure_length

Direction	Type
Input	Integer

The value must be 0.

data_structure

Direction	Type
Input	String

This field is currently ignored.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter. When the *key_identifier* is a key label, this field specifies the length of the label. The maximum size that you can specify is 8000 bytes.

key_identifier

Direction	Type
Input	String

For PKA keys, a token or label of an internal RSA or CRYSTALS-Kyber private key or an external private key token containing a clear private key.

For RSA keys, the key may be in modulus-exponent or Chinese Remainder Theorem format.

For secure PKCS #11 keys, this is the 44-byte handle of the private key, prefixed with an EBCDIC equal sign character (‘=’ or x’7E’), and padded on the right with spaces for a total length of 64 bytes. The corresponding public key was used to wrap the key value.

The CRYSTALS-Kyber private key usage attributes must allow dataEncipherment.

target_keyvalue_length

Direction	Type
Input/Output	Integer

The length of the *target_keyvalue* parameter. The maximum size that can be generated is 512 bytes.

On return, this field is updated with the actual length of *target_keyvalue*.

If ZERO-PAD is specified when the *key_identifier* is an RSA private key, this length will be the same as the RSA modulus byte length.

When the *key_identifier* is a CRYSTALS-Kyber private key, the returned length will be 32 bytes.

target_keyvalue

Direction	Type
Output	String

This field will contain the decrypted, deformatted key value. If ZERO-PAD is specified, the decrypted key value, including leading zeros, will be returned.

Restrictions

The exponent of the RSA public key must be odd.

Crypto Express accelerators do clear key RSA operations. Requests with encrypted keys are routed to a coprocessor.

The OAEP standard (PKCS #1) defines overhead = (2 * hLen) + 2 Bytes. hLen is the encoding hash algorithm output length in Bytes. This gives additional overhead:

- 42 Bytes for SHA-1
- 56 Bytes for SHA-224
- 66 Bytes for SHA-256
- 98 Byes for SHA-384
- 130 Bytes for SHA-512

RSA key size restrictions:

- The RSA key used must have a modulus size greater than or equal to the total PKOAEP2 message bit length, calculated with the data above as (source data size + total overhead).
- Minimum source data length is zero bytes, giving total message sizes (and therefore minimum RSA key sizes):
  - 0 + 42 = 42 Bytes (336 bits) for SHA-1 OAEP
  - 0 + 58 = 58 Bytes (464 bits) for SHA-224 OAEP
  - 0 + 66 = 66 Bytes (528 bits) for SHA-256 OAEP
  - 0 + 98 = 98 Bytes (784 bits) for SHA-384 OAEP
  - 0 + 130 = 130 Bytes (1040 bits) for SHA-512 OAEP
- The Maximum RSA key size is 4096 bits (512 bytes); therefore, the maximum message size is key size - overhead:
  - 512 – 42 = 470 Bytes (3760 bits) for SHA-1 OAEP
  - 512 – 58 = 454 Bytes (3632 bits) for SHA-224 OAEP
  - 512 – 66 = 446 Bytes (3568 bits) for SHA-256 OAEP
  - 512 – 98 = 414 Bytes (3312 bits) for SHA-384 OAEP
  - 512 – 130 = 382 Bytes (3056 bits) for SHA-512 OAEP

Authorization

To use this service with a secure PKCS #11 private key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access) to the containing PKCS #11 token.

To use this service with a secure PKCS #11 private key that is a private object, the caller must have USER (READ) authority (user access) to the containing PKCS #11 token.

See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS, PKDS, or TKDS.

PKA RSA private key must be enabled for key management functions. Secure PKCS #11 private keys must be enabled for decryption.

For PKA keys, the hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.

Access control points

For PKA keys, the **PKA Decrypt** access control point controls the function of this service.

Use of a CRYSTALS-Kyber private key in the *key_identifier* parameter requires the **PKA Decrypt - Allow CRYSTALS-Kyber keys** access control point.

There are access control points to disable a formatting rule. All of these controls are disabled in the domain role. Enabling these access control points will cause the request for the keyword to fail.

Table 145. PKA Decrypt access controls

Access control point	Rule array keyword
PKA Decrypt - Disallow PKCS-1.2	PKCS-1.2
PKA Decrypt - Disallow PKCSOAEP	PKCSOAEP
PKA Decrypt - Disallow PKOAEP2	PKOAEP2
PKA Decrypt - Disallow ZEROPAD	ZEROPAD

Note: Access control checking will not be performed when the request is routed to an accelerator.

For secure PKCS #11 private keys, see 'PKCS #11 Access Control Points' in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the access control points of the Enterprise PKCS #11 coprocessor.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 146. PKA Decrypt required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords PKCSOAEP, SHA-1, and SHA-256 require the July 2015 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express5 Accelerator	Compliant-tagged key tokens are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express5 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key. Keywords ZEROPAD, PKCSOAEP, SHA-1, and SHA-256 are not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.

Table 146. PKA Decrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Keywords PKCSOAEP, SHA-1, and SHA-256 require the July 2015 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express5 Accelerator Crypto Express6 Accelerator	Compliant-tagged key tokens are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express5 Enterprise PKCS #11 coprocessor Crypto Express6 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key. Keywords ZEROPAD, PKCSOAEP, SHA-1, and SHA-256 are not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.

Table 146. PKA Decrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express5 Accelerator Crypto Express6 Accelerator Crypto Express7 Accelerator	Compliant-tagged key tokens are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express5 Enterprise PKCS #11 coprocessor Crypto Express6 Enterprise PKCS #11 coprocessor Crypto Express7 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key. Compliant-tagged key tokens are not supported. Keywords ZEROPAD, PKCSOAEP, SHA-1, and SHA-256 are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.

Table 146. PKA Decrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Crypto Express7 CCA Coprocessor	CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express8 CCA Coprocessor	CRYSTALS-Kyber private keys require CCA release 8.0 or later licensed internal code (LIC). Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 require CCA release 8.1 or later Licensed Internal Code (LIC).
	Crypto Express6 Accelerator Crypto Express7 Accelerator Crypto Express8 Accelerator	Compliant-tagged key tokens are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express6 Enterprise PKCS #11 coprocessor Crypto Express7 Enterprise PKCS #11 coprocessor Crypto Express8 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key. Compliant-tagged key tokens are not supported. Keywords ZEROPAD, PKCSOAEP, SHA-1, and SHA-256 are not supported. CRYSTALS-Kyber private keys are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.

PKA Encrypt (CSNDPKE and CSNFPKE)

This callable service encrypts a supplied clear key value under an RSA or CRYSTALS-Kyber public key. The rule array keyword specifies the format of the key prior to encryption.

The callable service name for AMODE(64) invocation is CSNFPKE.

Format

```
CALL CSNDPKE(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              keyvalue_length,
              keyvalue,
              sym_key_identifier_length,
              sym_key_identifier,
              PKA_key_identifier_length,
              PKA_key_identifier,
              PKA_enciphered_keyvalue_length,
              PKA_enciphered_keyvalue)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value can be 1, 2, 3, 4, or 5.

rule_array

Direction	Type
Input	String

A keyword that provides control information to the callable service. The keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 147. Keywords for PKA Encrypt

Keyword	Meaning
Formatting Method (required) specifies the method to use to format the key value prior to encryption.	
PKCS-1.2	RSA PKCS #1 V1.5 block type 02 format will be used to format the supplied key value.

Table 147. Keywords for PKA Encrypt (continued)

Keyword	Meaning
ZERO-PAD	The key value will be padded on the left with binary zeros to the length of the PKA key modulus. The exponent of the RSA public key must be odd. Required when the *PKA_key_identifier* is a CRYSTALS-Kyber public key.
MRP	The key value will be padded on the left with binary zeros to the length of the PKA key modulus. The RSA public key may have an even or odd exponent.
PKCSOAEP	Specifies to format the data using the RSAES-OAEP encoding scheme defined in the RSA PKCS #1 V2.0 standard. The formatted key is encrypted by the RSA public-key provided as the transport key and returned as an opaque data buffer.
PKOAEP2	Specifies that the key is formatted as defined in the RSA PKCS #1 v2.1 standard for the RSAES-OAEP encryption/decryption scheme.
Hash Method (one required when PKCSOAEP or PKOAEP2 is specified. Otherwise, not allowed.)	
SHA-1	Specifies to use the SHA-1 method to calculate the OAEP message digest.
SHA-224	Specifies to use the SHA-224 method to calculate the OAEP message digest. Only supported with PKOAEP2.
SHA-256	Specifies to use the SHA-256 method to calculate the OAEP message digest.
SHA-384	Specifies to use the SHA-384 method to calculate the OAEP message digest. Only supported with PKOAEP2.
SHA-512	Specifies to use the SHA-512 method to calculate the OAEP message digest. Only supported with PKOAEP2.
Key Rule (optional)	
KEYIDENT	This indicates that the value in the *keyvalue* field is the label of a clear key token in the CKDS. The *keyvalue_length* must be 64. Not supported when the *PKA_key_identifier* is a CRYSTALS-Kyber public key.
RANDOM	This indicates to generate a random 32-byte value and return this value, formatted according to the formatting method keyword and encrypted. The random 32-byte value can be returned in one of two ways: 1. Encrypted under the provided AES CIPHER key provided in *sym_key_identifier*, using the AES encryption IV provided as input through the *keyvalue* parameter. This encrypted value is returned as output to the *keyvalue* parameter. 2. Encrypted under the CRYSTALS-Kyber public key provided in the *PKA_key_identifier* parameter, returned in the *PKA_enciphered_keyvalue* parameter. Only valid with ZERO-PAD and when the *PKA_key_identifier* contains a CRYSTALS-Kyber public key.
Certificate validation method (one required when the input is an X.509 certificate. Otherwise, must not be specified.)	

Table 147. Keywords for PKA Encrypt (continued)	
Keyword	**Meaning**
RFC-2459	Validate the certificate using the semantics of RFC-2459.
RFC-3280	Validate the certificate using the semantics of RFC-3280.
RFC-5280	Validate the certificate using the semantics of RFC-5280.
RFC-ANY	Attempt to validate the certificate by first using the semantics of RFC-2459, then the semantics of RFC-3280, and finally, the semantics of RFC-5280.
Public Key Infrastructure Usage (one optional when the input is an X.509 certificate. Otherwise, must not be specified.)	
PKI-CHK	Specifies that the X.509 certificate is to be validated against the trust chain of the PKI hosted in the adapter. This requires that the CA credentials have been installed into all coprocessors using the Trusted Key Entry workstation. This is the default.
PKI-NONE	Specifies that the X.509 certificate is not to be validated against the trust chain of the PKI hosted in the adapter. This is suitable if the certificate has been validated using host-based PKI services or if using a self-signed certificate.

keyvalue_length

Direction	**Type**
Input/Output	Integer

The length of the *keyvalue* parameter. The maximum field size is 512 bytes. The actual maximum size depends on the RSA modulus length of *PKA_key_identifier* and the formatting method you specify in the *rule_array* parameter. When key rule KEYIDENT is specified, then the *keyvalue_length* parameter is required to be 64 bytes.

When the *PKA_key_identifier* is a CRYSTALS-Kyber public key and the RANDOM keyword has not been specified, the *keyvalue_length* parameter is required to be 32 bytes.

When the RANDOM keyword is specified and *sym_key_identifier* contains an AES CIPHER key, the *keyvalue_length* parameter must be 32 bytes or larger. On return, this field will be updated with the actual length of the *keyvalue* parameter.

When *PKA_key_identifier* is a CRYSTALS-Kyber public key, the RANDOM keyword has been specified, and *sym_key_identifier* does not contain a key, this parameter is required to be set to zero and is not updated.

keyvalue

Direction	**Type**
Input	String

This field contains the supplied clear key value to be encrypted under the *PKA_key_identifier*. When key rule KEYIDENT is specified, the *keyvalue* parameter is assumed to contain a label for a valid CKDS clear key token.

When the *PKA_key_identifier* is a CRYSTALS-Kyber public key, the RANDOM keyword has been specified, and *sym_key_identifier* contains an AES CIPHER key, a random 32-byte value is generated and returned in an encrypted form.

On input, this parameter should contain the 16-byte initialization vector for the AES encryption left justified in the buffer. On output, the *keyvalue* parameter will be updated to contain the random 32-byte value enciphered by the AES CIPHER key passed in the *sym_key_identifier* parameter.

When the *PKA_key_identifier* is a CRYSTALS-Kyber public key, the RANDOM keyword has been specified, and the *sym_key_identifier* does not contain an AES CIPHER key, this parameter is not updated.

sym_key_identifier_length

Direction	Type
Input	Integer

The length of the *sym_key_identifier* parameter in bytes.

When the PKA_key_identifier is a CRYSTALS-Kyber public key, the RANDOM keyword has been specified, and the sym_key_identifier is needed to encrypt the output value:

- When the *sym_key_identifier* contains a label, the length must be 64.
- When the *sym_key_identifier* contains a key token, the length must be between the actual length of the token and 9992.

Otherwise, this value must be 0.

sym_key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to encrypt the output value. The key identifier is a variable-length key token or key block or the label of a token or block in key storage.

When *sym_key_identifier_length* is zero, this parameter is ignored.

For CCA keys, the identifier is a variable-length AES key token of key type CIPHER with key usage attributes ENCRYPT and CBC enabled.

For X9.143 (TR-31) keys, the identifier is a variable-length key block of an AES data-encrypting key usage D0, algorithm A, and mode of use B or E.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

PKA_key_identifier_length

Direction	Type
Input	Integer

The length of the *PKA_key_identifier* parameter. When the *PKA_key_identifier* is a key label, this field specifies the length of the label. The maximum size that you can specify is 8000 bytes.

PKA_key_identifier

Direction	Type
Input	String

The token or label of an RSA public or private key token, or the X.509 certificate containing the RSA public key, or the token or label of a CRYSTALS-Kyber public key token to be used to encrypt the supplied key value.

Certificates may be PEM-formatted EBCDIC text or DER-encoded. The certificate may either have no RSA key usage attribute or it must have at least one of the following usages: keyEncipherment or dataEncipherment.

CRYSTALS-Kyber public key usage attribute must allow dataEncipherment.

PKA_enciphered_keyvalue_length

Direction	Type
Input/Output	Integer

The length of the *PKA_enciphered_keyvalue* parameter in bytes. The maximum size that can be generated when the *PKA_key_identifier* is an RSA key is 512 bytes. This length should be the same as the modulus length of the *PKA_key_identifier*.

When the *PKA_key_identifier* is a CRYSTALS-Kyber public key, the *PKA_enciphered_keyvalue_length* is the same as the size of the public key material. For a Crystals-Kyber(1024) key, this is 1568 bytes.

On return, this field is updated with the actual length of *PKA_enciphered_keyvalue*.

PKA_enciphered_keyvalue

Direction	Type
Output	String

This field contains the key value protected under an RSA or CRYSTALS-Kyber public key. This byte-length string is left-justified within the *PKA_enciphered_keyvalue* parameter.

Restrictions

The exponent for RSA public keys must be odd.

RSA keys 512-bit to 2048-bit may have a public exponent of 3, 5, 17, 257, 65537, or random. Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC).

For 2049-bit to 4096-bit RSA keys:

- The public exponent may have a value of 3, 5, 17, 257, 65537, or random.
- Support for a random public exponent requires zEC12, zBC12, and later systems with a CEX4C or later coprocessor with September 2013 or later licensed internal code (LIC).
- Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC).

The OAEP standard (PKCS #1) defines overhead = (2 * hLen) + 2 Bytes. hLen is the encoding hash algorithm output length in Bytes. This gives additional overhead:

- 42 Bytes for SHA-1
- 56 Bytes for SHA-224
- 66 Bytes for SHA-256
- 98 Bytes for SHA-384
- 130 Bytes for SHA-512

RSA key size restrictions:

- The RSA key used must have a modulus size greater than or equal to the total PKOAEP2 message bit length, calculated with the data above as (source data size + total overhead).
- Minimum source data length is zero bytes, giving total message sizes (and therefore minimum RSA key sizes):
  - 0 + 42 = 42 Bytes (336 bits) for SHA-1 OAEP
  - 0 + 58 = 58 Bytes (464 bits) for SHA-224 OAEP

  - 0 + 66 = 66 Bytes (528 bits) for SHA-256 OAEP
  - 0 + 98 = 98 Bytes (784 bits) for SHA-384 OAEP
  - 0 + 130 = 130 Bytes (1040 bits) for SHA-512 OAEP
- The Maximum RSA key size is 4096 bits (512 bytes); therefore, the maximum message size is key size - overhead:
  - 512 – 42 = 470 Bytes (3760 bits) for SHA-1 OAEP
  - 512 – 58 = 454 Bytes (3632 bits) for SHA-224 OAEP
  - 512 – 66 = 446 Bytes (3568 bits) for SHA-256 OAEP
  - 512 – 98 = 414 Bytes (3312 bits) for SHA-384 OAEP
  - 512 – 130 = 382 Bytes (3056 bits) for SHA-512 OAEP

Usage notes

- SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.
- For RSA DSI PKCS #1 formatting, the key value length must be at least 11 bytes less than the modulus length of the RSA key.
- The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.
- The key value to be encrypted must be smaller than the modulus in the *PKA_key_identifier*.

Access control points

The **PKA Encrypt** access control point controls the function of this service.

Use of a CRYSTALS-Kyber public key in the *PKA_key_identifier* parameter requires the **PKA Encrypt - Allow CRYSTALS-Kyber keys** access control point.

There are access control points to disable a formatting rule. All of these controls are disabled in the domain role. Enabling these access control points will cause the request for the keyword to fail.

Table 148. PKA Encrypt access controls

Access control point	**Rule array keyword**
PKA Encrypt - Disallow MRP	MRP
PKA Encrypt - Disallow PKCS-1.2	PKCS-1.2
PKA Encrypt - Disallow PKCSOAEP	PKCSOAEP
PKA Encrypt - Disallow PKOAEP2	PKOAEP2
PKA Encrypt - Disallow ZEROPAD	ZEROPAD

Note: Access control checking will not be performed when the request is routed to an accelerator.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 149. PKA Encrypt required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Routed to a CEX5A if one is available (PKCSOAEP, ZERO-PAD, and MRP only). Keywords PKCSOAEP, SHA-1, and SHA-256 require the July 2015 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express5 Accelerator	PKCS-1.2 keyword not supported. Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.

Table 149. PKA Encrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Routed to a CEX5A or CEX6A if one is available (PKCSOAEP, ZERO-PAD, and MRP only). Keywords PKCSOAEP, SHA-1, and SHA-256 require the July 2015 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express6 CCA Coprocessor	Routed to a CEX5A or CEX6A if one is available (PKCSOAEP, ZERO-PAD, and MRP only). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE require the July 2019 or later licensed internal code (LIC). X.509 certificates require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express5 Accelerator Crypto Express6 Accelerator	PKCS-1.2 keyword not supported. Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.

Table 149. PKA Encrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Routed to a CEX5A, CEX6A, or CEX7A if one is available (PKCSOAEP, ZERO-PAD, and MRP only). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Routed to a CEX5A, CEX6A, or CEX7A if one is available (PKCSOAEP, ZERO-PAD, and MRP only). CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express5 Accelerator Crypto Express6 Accelerator Crypto Express7 Accelerator	PKCS-1.2 keyword not supported. Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.

Table 149. PKA Encrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Crypto Express7 CCA Coprocessor	Routed to a CEX5A, CEX6A, or CEX7A if one is available (PKCSOAEP, ZERO-PAD, and MRP only). CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.
	Crypto Express8 CCA Coprocessor	Routed to a CEX5A, CEX6A, or CEX7A if one is available (PKCSOAEP, ZERO-PAD, and MRP only). CRYSTALS-Kyber public keys and the RANDOM keyword require CCA release 8.0 or later licensed internal code (LIC). Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 require CCA release 8.1 or later Licensed Internal Code (LIC).
	Crypto Express6 Accelerator Crypto Express7 Accelerator Crypto Express8 Accelerator	Keywords PKCS-1.2, RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber public keys and the RANDOM keyword are not supported. Rules SHA-224, SHA-384, SHA-512, and PKOAEP2 are not supported.

Prohibit Export (CSNBPEX and CSNEPEX)

Note: This service has been deprecated. New applications should use the Restrict Key Attribute service that is described in “Restrict Key Attribute (CSNBRKA and CSNERKA)” on page 402 instead of this service.

Use this service to modify an exportable internal DES key token so that it cannot be exported.

The callable service name for AMODE(64) invocation is CSNEPEX.

Format

```
CALL CSNBPEX(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_identifier

Direction	Type
Input/Output	String

A 64-byte string variable containing the internal key token to be modified. The returned *key_identifier* will be encrypted under the current master key.

The output *key_identifier* will be wrapped in the same manner as the input *key_identifier*.

The DES key wrapping methods available are described in “CCA key wrapping” on page 20.

Restrictions

DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.

Triple-length DES keys are not supported.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control point

The **Prohibit Export** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 150. Prohibit Export required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Prohibit Export Extended (CSNBPEXX and CSNEPEXX)

Note: This service has been deprecated. New applications should use the Restrict Key Attribute service that is described in "Restrict Key Attribute (CSNBRKA and CSNERKA)" on page 402 instead of this service.

Use the Prohibit Export Extended callable service to change the external token of a cryptographic key in exportable DES key token form so that it can be imported at the receiver node and is non-exportable from that node. You cannot prohibit export of DATA keys.

The inputs are an external token of the key to change in the *source_key_token* parameter and the label or internal token of the exporter key-encrypting key in the *KEK_key_identifier* parameter.

This service is a variation of the Prohibit Export service (CSNBPEX and CSNEPEX), which supports changing an *internal* token.

The callable service name for AMODE(64) invocation is CSNEPEXX.

Format

```
CALL CSNBPEXX(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              source_key_token,
              KEK_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

source_key_token

Direction	Type
Input/Output	String

A 64-byte string of an external token of a key to change. It is in exportable form.

The output *source_key_token* will be wrapped in the same manner as the input *source_key_token*.

The DES key wrapping methods available are described in “CCA key wrapping” on page 20.

KEK_key_identifier

Direction	Type
Input/Output	String

A 64-byte string of an internal token or label of the exporter KEK used to encrypt the key contained in the external token specified in the previous parameter.

Restrictions

External DATAM and DATAMV keys are not supported.

Triple-length DES keys are not supported.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control point

The **Prohibit Export Extended** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 151. Prohibit Export Extended required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)

The callable service uses either the Crypto Express adapter or the CP assist for Cryptographic Functions instructions to generate random numbers.

There are two forms of the Random Number Generate callable service. One version returns an 8-byte random number. The second version allows the caller to specify the length of the random number.

The callable service names for AMODE(64) invocation are CSNERNG and CSNERNGL.

Format

```
CALL CSNBRNG(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             form,
             random_number )
```

```
CALL CSNBRNGL(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             random_number_length,
             random_number )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

form

Direction	Type
Input	Character String

The 8-byte keyword for the CSNBRNG service that defines the characteristics of the random number should be left-justify and pad on the right with blanks. The keywords are listed in Table 152 on page 390.

Table 152. Keywords for the Form Parameter

Keyword	Meaning
EVEN	Generate a 64-bit random number with even parity in each byte.
ODD	Generate a 64-bit random number with odd parity in each byte.
RANDOM	Generate a 64-bit random number.

Parity is calculated on the 7 high-order bits in each byte and is presented in the low-order bit in the byte.

rule_array_count

Direction	Type
Input	Integer

The number of keywords for the CSNBRNGL service you are supplying in the *rule_array* parameter. The value must be 1 or 2.

rule_array

Direction	Type
Input	String

The keyword for the CSNBRNGL service that provides control information to the callable service. The recovery method is the method to use to recover the symmetric key. The keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 153. Keywords for Random Number Generate Control Information

Keyword	Meaning
Requested service (one, required).	
EVEN	Specifies that each generated random byte is adjusted for even parity.
ODD	Specifies that each generated random byte is adjusted for odd parity.
RANDOM	Specifies that each generated random byte is not adjusted for parity.
RT-KRD	Specifies that the generated random number is returned formatted as a TR-34 Key Receiving Device Random Number Token (RT-KRD). The token requires 21 additional bytes for encoding and overhead. The random number is not adjusted for parity. Note the maximum size of the token that is usable with the service that the token is planned to be used with. If the maximum size is 200 bytes, the maximum random number size is 179 bytes.

Table 153. Keywords for Random Number Generate Control Information (continued)	
Keyword	**Meaning**
Encryption Process (one, optional). Not valid with the RT-KRD keyword.	
TDES-CBC	Specifies to return the random number encrypted using the DES key specified in the *key_identifier* parameter. **Note:** A CCA Crypto Express coprocessor must be active to get encrypted output.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter in bytes.

When the rule array keyword TDES-CBC is specified and when:

- The *key_identifier* parameter contains a label, the value must be 64.
- The *key_identifier* parameter contains a key token, the value must be between the actual length of the key token and 9992.

Otherwise, the value must be 0.

key_identifier

Direction	Type
Input	String

The identifier of the key to encrypt the random number. The key identifier is an operational token or the key label of an operational token in key storage.

When the TDES-CBC keyword is specified:

- For CCA keys, the identifier is a 64-byte DES key token of key type CIPHER or ENCIPHER and the key must be a double-length or triple-length key.
- For X9.143 keys, the identifier is a key block of a DES data-encrypting key usage D0, algorithm T, and mode of use B or E.

When the *key_identifier_length* parameter is 0, this parameter is ignored.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

random_number_length

Direction	Type
Input/Output	Integer

This parameter contains the desired length of the *random_number* that is returned by the CSNBRNGL callable service. The minimum value is 1 byte; the maximum value is 8192 bytes.

When the requested service keyword is TDES-CBC, the value must be a multiple of 8. The maximum value is 1024.

When the requested service keyword is RT-KRD:

- On input, this value is the number of bytes of the random number requested plus 21 bytes for the DER encoding of the token. Note the maximum size of the token that is usable with the service that the token is planned to be used with. If the maximum size is 200 bytes, the maximum random number size is 179 bytes.
- On output, the value will be the actual size of the token returned.

random_number

Direction	Type
Output	String

The generated number returned by the CSNBRNG callable service is stored in an 8-byte variable.

The generated number returned by the CSNBRNGL callable service is stored in a variable that is at least *random_number_length* bytes long.

When the requested service keyword is RT-KRD, the TR-34 Key Receiving Device Random Number Token is returned.

Usage notes

If the CSF.CSFSERV.AUTH.CSFRNG.DISABLE SAF resource profile is defined in the XFACILIT SAF resource class, no SAF authorization checks will be performed against the CSFSERV class when using this service. If CSF.CSFSERV.AUTH.CSFRNG.DISABLE is not defined, the SAF authorization check will be performed. Disabling the SAF check may improve the performance of your application.

Access control points

The CSNBRNG service requires that the **Key Generate – SINGLE-R** access control point is enabled. The CSNBRNGL service is not controlled by any access control.

The use of the TDES-CBC rule array keyword requires the **Random Number Generate Long – TDES-CBC** access control point is enabled.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 154. Random Number Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions	Rule array keyword TDES-CBC is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions	Rule array keyword TDES-CBC is not supported. X9.143 key blocks are not supported.

Table 154. Random Number Generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions	Rule array keyword TDES-CBC is not supported. X9.143 key blocks are not supported.
	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	Rule array keyword TDES-CBC is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	The rule array keyword TDES-CBC requires the CCA release 7.4 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	Rule array keyword TDES-CBC is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keyword TDES-CBC is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Remote Key Export (CSNDRKX and CSNFRKX)

This callable service uses the trusted block to generate or export DES keys for local use and for distribution to an ATM or other remote device. RKX uses a special structure to hold encrypted symmetric keys in a way that binds them to the trusted block and allows sequences of RKX calls to be bound together as if they were an atomic operation.

Rule array keywords may be specified to indicate whether to wrap an output DES CCA key token using the default wrapping mode or to use a specific method.

The callable service name for AMODE(64) invocation is CSNFRKX.

Format

```
CALL CSNDRKX(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              trusted_block_length,
              trusted_block_identifier,
              certificate_length,
              certificate,
              certificate_parms_length,
              certificate_parms,
              transport_key_length,
              transport_key_identifier,
              rule_id_length,
```

```
                rule_id,
                importer_key_length,
                importer_key_identifier,
                source_key_length,
                source_key_identifier,
                asym_encrypted_key_length,
                asym_encrypted_key,
                sym_encrypted_key_length,
                sym_encrypted_key,
                extra_data_length,
                extra_data,
                key_check_parameters_length,
                key_check_parameters,
                key_check_length,
                key_check_value)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the specific results of processing. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. This number must be 0, 1 or 2.

rule_array

Direction	Type
Input	Character string

The rule_array parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The *rule_array* keywords are:

Table 155. rule_array keywords

Keyword	Meaning
Key Wrapping Method (*Optional*)	
USECONFG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This is the default for triple-length keys.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.
WRAP-ECB	Specifies that the original wrapping method is to be used.
Translation Control (*Optional, valid only for enhanced wrapping*)	
ENH-ONLY	Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the original method. This restricts translation to the original method. This is the default when the wrapping method is WRAPENH2 or WRAPENH3.

trusted_block_length

Direction	Type
Input	Integer

Specifies the number of bytes in the trusted_block_identifier parameter. The maximum length is 3500 bytes.

trusted_block_identifier

Direction	Type
Input	String

Specifies a trusted block label or trusted block token of an internal/complete trusted block constructed by the service, which is used to validate the public key certificate (certificate) and to define the rules for key generation and export.

certificate_length

Direction	Type
Input	Integer

Specifies the number of bytes in the certificate parameter. The maximum is 5000 bytes.

If the certificate_length is zero and the trusted block's Asymmetric Encrypted Output Key Format indicates no asymmetric key output, this service will not attempt to use or validate the certificate in any way. Consequently, the output parameter asym_encrypted_key_length will contain zero and output parameter asym_encrypted_key will not be changed from its input content.

If the certificate length is zero and the trusted block's Asymmetric Encrypted Output Key Format indicates PKCS1.2 output format or RSAOAEP output format, this service will exit with an error.

If the certificate_length is non-zero and the trusted block's Asymmetric Encrypted Output Key Format indicates no asymmetric key output, this service will fail.

certificate

Direction	Type
Input	String

Contains a public-key certificate. The certificate must contain the public key modulus and exponent in binary_form, as well as a digital signature. The signature in the certificate will be verified using the root public key that is in the trusted block supplied in trusted_block_identifier parameter.

certificate_parms_length

Direction	Type
Input	Integer

Contains the number of bytes in the certificate_parms parameter. The length must be 36 bytes.

certificate_parms

Direction	Type
Input	String

Contains a structure provided by the caller used for identifying the location and length of values within the certificate in parameter certificate. For each of the values used by RKX, the structure contains offsets from the start of the certificate and length in bytes. It is the responsibility of the calling application program to provide these values. This method gives the greatest flexibility to support different certificate formats. The structure has this layout:

Table 156. Structure of values used by RKX

Offset (bytes)	Length (bytes)	Description
0	4	Offset of modulus
4	4	Length of modulus
8	4	Offset of public exponent
12	4	Length of public exponent
16	4	Offset of digital signature
20	4	Length of digital signature

Table 156. Structure of values used by RKX (continued)

Offset (bytes)	Length (bytes)	Description
24	1	Identifier for the hash algorithm used • 01 - SHA-1 • 05 - SHA-256
25	1	Identifier for the digital hash formatting method • 01 - PKCS-1.0 • 02 - PKCS-1.1 • 03 - X9.31 • 04 - ISO-9796 • 05 - ZERO-PAD
26	2	Reserved - must be filled with 0x00 bytes
28	4	Offset of first byte of certificate data hashed to compute the digital signature
32	4	Length of the certificate data hashed to compute the digital signature

The modulus, exponent, and signature values are right-justified and padded on the left with binary zeros if necessary.

transport_key_length

Direction	Type
Input	Integer

Contains the number of bytes in the transport_key_identifier parameter.

transport_key_identifier

Direction	Type
Input	String

Contains a label of an internal key token, or an RKX token for a Key Encrypting Key (KEK) that is used to encrypt a key exported by the RKX service. A transport key will not be used to encrypt a generated key.

For flag bit0=1 (export existing key) within Rule section and parameter rule_id = Rule section ruleID, the transport_key_identifier encrypts the exported version of the key received in parameter source_key_identifier. The service can distinguish between the internal key token or RKX key token by virtue of the version number at offset 0x04 contained in the key token and the flag value at offset 0x00 as follows:

Table 157. Transport_key_identifer used by RKX

Flag Byte Offset 00	Version Number Offset 04	Description
0X01	0X00	Internal DES key token version 0

Table 157. Transport_key_identifer used by RKX (continued)

Flag Byte Offset 00	Version Number Offset 04	Description
0X02	0X10	RKX Key token (Flag byte 0x02 indicates external key token)

rule_id_length

Direction	Type
Input	Integer

Contains the number of bytes in the rule_id parameter. The value must be 8.

rule_id

Direction	Type
Input	String

Specifies the rule in the trusted block that will be used to control key generation or export. The trusted block can contain multiple rules, each of which is identified by a rule ID value.

importer_key_length

Direction	Type
Input	Integer

Contains the number of bytes in the importer_key_identifier parameter. It must be zero if the Generate/Export flag in the rule section (section 0x12) of the Trusted Block is a zero, indicating a new key is to be generated.

importer_key_identifier

Direction	Type
Input	String

Contains a key token or key label for the IMPORTER key-encrypting key that is used to decipher the key passed in parameter source_key_identifier. It is unused if either RKX is being used to generate a key, or if the source_key_identifier is an RKX key token or internal DES key token.

source_key_length

Direction	Type
Input	Integer

Contains the number of bytes in the source_key_identifier parameter. The parameter must be 0 if the trusted block Rule section ruleID = rule_id parameter and the flag bit0 = 0 (Generate new key).

The parameter must be 64 if the trusted block Rule section has a flag bit0 = 1 (Export existing key).

source_key_identifier

Direction	Type
Input	String

Contains a label of a single or double length external or internal key token or an RKX key token for a key to be exported. It must be empty (source_key_length=0) if RKX is used to generate a new key. The service examines the key token to determine which form has been provided.

Table 158. Examination of key token for source_key_identifier

Flag Byte Offset 00	Version Number Offset 04	Description
0X01	0X00	Internal DES key token version 0
0X02	0X00	External DES key token version 0
0X02	0X01	External DES key token version 1
0X02	0X10	RKX Key token (Flag byte 0x02 indicates external key token)

asym_encrypted_key_length

Direction	Type
Input/Output	Integer

The length of the asym_encrypted_key parameter. On input, it is the length of the storage to receive the output. On output, it is the length of the data returned in the asym_encrypted_key parameter. The maximum length is 512 bytes.

asym_encrypted_key

Direction	Type
Output	String

The contents of this field is ignored on input. A string buffer RKX will use to return a generated or exported key that is encrypted under the public (asymmetric) key passed in parameter certificate. An error will be returned if the caller's buffer is too small to hold the value that would be returned.

sym_encrypted_key_length

Direction	Type
Input/Output	Integer

On input, the sym_encrypted_key_length parameter is an integer variable containing the number of bytes in the sym_encrypted_key field. On output, that value in sym_encrypted_key_length is replaced with the length of the key returned in sym_encrypted_key field.

sym_encrypted_key

Direction	Type
Output	String

Sym_encrypted_key is the string buffer RKX uses to return a generated or exported key that is encrypted under the key-encrypting key passed in the *transport_key_identifier* parameter. The value

returned will be 64 bytes. An error will be returned if the caller's buffer is smaller than 64 bytes because it is too small to hold the value that would be returned. The *sym_encrypted_key* may be an RKX key token or a key token depending upon the value of the Symmetric Encrypted Output Key Format value of the Rule section within the *trusted_block_identifier* parameter.

The *sym_encrypted_key* will be wrapped in the same manner as the *source_key_identifier*.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

extra_data_length

Direction	Type
Input	Integer

Contains the number of bytes of data in the extra_data parameter. It must be zero if the output format for the RSA-encrypted key in asym_encrypted_key is anything but RSAOEAP. The maximum size is 1024 bytes.

extra_data

Direction	Type
Input	String

Can be used in the OAEP key wrapping process. Extra_data is optional and is only applicable when the output format for the RSA-encrypted key returned in asym_encrypted_key is RSAOAEP.

Note: RSAOAEP format is specified in the rule in the trusted block.

key_check_parameters_length

Direction	Type
Input	Integer

Contains the number of bytes in the key_check_parameters parameter. Currently, none of the defined key check algorithms require any key check parameters, so this field must specify 0.

key_check_parameters

Direction	Type
Input	String

Contains parameters that are required to calculate a key check value parameter, which will be returned in key_check_value. Currently, none of the defined key check algorithms require any key check parameters, but you must still specify this parameter.

key_check_length

Direction	Type
Input/Output	Integer

On input this parameter contains the number of bytes in the key_check_value parameter. On output, the value is replaced with the length of the key check value returned in the key_check_value parameter. The length depends on the key-check algorithm identifier. See Table 656 on page 1592.

key_check_value

Direction	Type
Output	String

Used by RKX to return a key check value that calculates on the generated or exported key. Values in the rule specified with rule_id can specify a key check algorithm that should be used to calculate this output value.

Usage notes

If ICSF is configured to audit the lifecycle of tokens (for example, AUDITKEYLIFECKDS(TOKEN(YES),...) is specified) and a token is passed as input to be exported, a request is made to the Crypto Express Coprocessor to generate the key fingerprint that will be used for auditing the exported key.

If ICSF is configured to audit the lifecycle of tokens (for example, AUDITKEYLIFECKDS(TOKEN(YES),...) is specified), a request is made to the Crypto Express Coprocessor to generate the key fingerprint to be used for auditing the generated key.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

The **Remote Key Export - Gen or export a non-CCA node key** access control point controls the function of this service.

The service also requires that the **Key Generate – SINGLE-R** access control point is enabled to replicate a single-length source key (either from a fixed-length DES key-token or an RKX key-token). If authorized, key replication occurs if all of the following are true:

- The key token returned using the *sym_encrypted_key_identifier* parameter is a fixed-length DES key-token, as defined in the rule section identified by the *rule_id* parameter.
- The rule section identified by the *rule_id* parameter has a common export key parameters subsection defined and the control vector in the subsection is 16 bytes in length with key-form bits of B'010' for the left half and B'001' for the right half (see “Key form bits, 'fff'” on page 1608.
- The token identified by the *source_key_identifier* parameter is single length and is either a fixed-length DES key-token or an RKX key-token.

To use a NOCV IMPORTER key-encrypting key with the Remote Key Export service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

To use a NOCV EXPORTER key-encrypting key with the Remote Key Export service, the **NOCV KEK usage for export-related functions** access control point must be enabled in addition to one or both of the access control points listed.

If the key-encrypting key identifier is a weaker key than the key being exported, then

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

To use the default key-wrapping configuration and rule array keywords, the **Remote Key Export - Include RKX in default wrap config** access control point must be enabled.

- If enabled and no keywords are specified, the wrapping of an output DES CCA key token is based on the default configuration setting.
- If disabled, the key-wrapping method of the source key token determines the wrapping of the output DES CCA key token.

When the key wrapping method keyword specifies a wrapping method that is not the default method, the **Remote Key Export - Allow wrapping override keywords** access control point must be enabled.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 159. Remote Key Export required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Triple-length DES keys are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Triple-length DES keys are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express7 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Restrict Key Attribute (CSNBRKA and CSNERKA)

Use the Restrict Key Attribute callable service to modify an attribute of an internal or external CCA symmetric key-token.

The callable service name for AMODE(64) is CSNERKA.

Format

```
CALL CSNBRKA(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              key_encrypting_key_identifier_length,
              key_encrypting_key_identifier,
              opt_parameter1_length,
              opt_parameter1,
              opt_parameter2_length,
              opt_parameter2 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be between 1 and 10, inclusive.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 160. Keywords for Restrict Key Attribute Control Information

Keyword	Meaning
Key Identifier Type (Required)	
AES	Specifies the key identifier is an AES key token or X9.143 (TR-31) key block.
DES	Specifies the key identifier is a DES key token or X9.143 (TR-31) key block.
HMAC	Specifies the key identifier is an HMAC key token or X9.143 (TR-31) key block.
Attribute to Restrict (Optional)	
NOEXPORT	Prohibits the key from being exported by any verb. The use of this keyword always causes each available export control attribute to be lowered. If no attribute to restrict keywords are used, this is the default. Variable-length symmetric key-token: This keyword is equivalent to providing all of the keywords listed under Export Control for AES or HMAC (NOEX-AES, NOEX-DES, NOEX-RAW, NOEX-RSA, NOEX-SYM, NOEXAASY, and NOEXUASY). This is the default if no AES or HMAC attribute restriction keywords are used. DES key tokens: Use this keyword to set CV bit 17 = B'0' (NO-XPORT) and CV bit 57 = B'1' (NOT31XPT). This is the default if no DES attribute restriction keywords are used. X9.143 (TR-31) key blocks: When the current Exportability setting of ’S’ or ’E’, the Exportability will be changed to ’N’. For other Exportability settings, no change is made.
NOEXNX24	Prohibits the key in a X9.143 (TR-31) key block from being exported under a KEK that is not compliant with ANSI X9.24 parts 1 and 2. For a key with current Exportability setting of 'S', the Exportability will be changed to 'E'. For other Exportability settings, no change is made. Valid only when key identifier refers to an operational X9.143 (TR-31) key block.
For AES or HMAC keys *(Optional, one or more keywords may be specified)*	
Export control for AES and HMAC *(one or more, optional)*	
NOEX-AES	Specifies to prohibit export using an AES key.
NOEX-DES	Specifies to prohibit export using a DES key.
NOEX-RAW	Specifies to prohibit export in RAW format.

Table 160. Keywords for Restrict Key Attribute Control Information (continued)	
Keyword	**Meaning**
NOEX-RSA	Specifies to prohibit export using an RSA key.
NOEX-SYM	Prohibits the key from being exported using a symmetric key.
NOEXAASY	Prohibits the key from being exported using an authenticated asymmetric key (for example, an RSA key in a trusted block token).
NOEXUASY	Prohibits the key from being exported using an unauthenticated asymmetric key.
Key usage restriction for AES and HMAC (optional)	
C-XLATE	Specifies that the CIPHER key can only be used for cipher text translate operations. This is only valid with AES CIPHER keys.
For DES keys (Optional, one or two keywords)	
Export control for DES (one, optional)	
CCAXPORT	For DES key tokens, set bit 17 of the CV to 0 to prohibit any export of the key.
NOT31XPT	For DES key tokens, set bit 57 of the CV to 1 to prohibit TR-31 export of the key.
Key restriction for DES (optional)	
DOUBLE-O	For DES key tokens, change the control vector of a double-length key that has unique key halves (ignoring parity) to indicate that the key does not have replicated key halves. Key type must be EXPORTER, IKEYXLAT, IMPORTER, or OKEYXLAT. If key type is EXPORTER or IMPORTER, the key must not be marked as a NOCV key-encrypting key. **Note:** A double-length key with replicated key halves has the effective strength of a single-length key. If the key token supplied in the key_identifier parameter has replicated key halves, this keyword will cause the service to fail.
Input Transport Key (Optional)	
IKEK-AES	Specifies the KEK is an AES transport key. This is the default for Token Types AES and HMAC, and is not allowed with Token Type DES.
IKEK-DES	Specifies the KEK is a DES transport key. This is the default for Token Type DES.
IKEK-PKA	Specifies the KEK is a PKA transport key. This is not allowed with Token Type DES.

key_identifier_length

Direction	Type
Input/Output	Integer

The length of the *key_identifier* parameter in bytes.

When the *key_identifier* is a label, the value must be 64.

When the *key_identifier* is a key token or key block, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The key for which the export control is to be updated. The parameter contains an internal or external key token or key block or the 64-byte CKDS label of an internal token or block. If a label is specified, the key token will be updated in the CKDS and not returned by this service.

If the key identifier supplied was a symmetric key token or block encrypted under the old master key, the token or block will be returned encrypted under the current master key.

key_encrypting_key_identifier_length

Direction	Type
Input	Integer

The length of the *key_encrypting_key_identifier* parameter. When *key_identifier* is an internal token, the value must be zero.

- If *key_encrypting_key_identifier* is a label for either the CKDS (IKEK-AES or IKEK-DES rules) or PKDS (IKEK-PKA rule), the value must be 64.
- If *key_encrypting_key_identifier* is an AES KEK, the value must be between the actual length of the token and 725.
- If *key_encrypting_key_identifier* is a DES KEK, the value must be 64.
- When the *key_encrypting_key_identifier* contains an X9.143 (TR-31) key block, the value must be between the actual length of the token and 9992.
- If *key_encrypting_key_identifier* is an RSA KEK, the maximum length is 3500.

key_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to unwrap the input key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

When *key_encrypting_key_identifier_length* is zero, this parameter is ignored.

For AES and DES key-encrypting keys:

For CCA keys
The identifier is variable-length key token of an AES or DES IMPORTER or EXPORTER.

For X9.143 (TR-31) keys
The identifier is a variable-length key block of an AES or DES key-encrypting key: key usage K0 or K1, algorithm A or T, and mode of use D or E.

If the key identifier supplied was an AES or DES key token or block encrypted under the old master key, the token or block will be returned encrypted under the current master key.

opt_parameter1_length

Direction	Type
Input	Integer

The byte length of the *opt_parameter1* parameter. The value must be zero.

opt_parameter1

Direction	Type
Input	String

This parameter is ignored.

opt_parameter2_length

Direction	Type
Input	Integer

The byte length of the *opt_parameter2* parameter. The value must be zero.

opt_parameter2

Direction	Type
Input	String

This parameter is ignored.

Access control points

The access control points in the domain role that control the function of this service are:

Table 161. Required access control points for Restrict Key Attribute

Access control	Rule array keyword
Restrict Key Attribute - Export Control	AES or HMAC
Restrict Key Attribute - Permit setting the TR-31 export bit	DES

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 162. Restrict Key Attribute required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.

Table 162. Restrict Key Attribute required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Secure Key Import (CSNBSKI and CSNESKI)

Note: This service has been deprecated. New applications should use the Multiple Secure Key Import service that is described in "Multiple Secure Key Import (CSNBSKM and CSNESKM)" on page 358 instead of this service. The Multiple Secure Key Import service supports single-length, double-length, and triple-length DES keys for all key types and AES keys for all key types.

Use the Secure Key Import callable service to encipher a single-length or double-length clear key under the DES master key or under an importer key-encrypting key. The clear key can then be imported as any of the possible key types. This service does not adjust key parity.

The callable service can execute only when ICSF is in special secure mode, which is described in "Special secure mode" on page 10.

The callable service name for AMODE(64) invocation is CSNESKI.

Format

```
CALL CSNBSKI(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              clear_key,
              key_type,
```

```
            key_form,
            importer_key_identifier,
            key_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

clear_key

Direction	Type
Input	String

The clear key to be enciphered. Specify a 16-byte string (clear key value). For single-length keys, the value must be left-justified and padded with zeros. For effective single-length keys, the value of the right half must equal the value of the left half. For double-length keys, specify the left and right key values.

Note: For key types that can be single or double-length, a single length encrypted key will be generated if a *clear_key* value of zeros is supplied.

key_type

Direction	Type
Input	Character String

The type of key you want to encipher under the master key or an importer key. Specify an 8-byte field that must contain a keyword from this list or the keyword TOKEN. If the key type is TOKEN, ICSF determines the key type from the CV in the *key_identifier* parameter.

Key type values for the Secure Key Import callable service are: CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR, DATA, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IMP-PKA, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER.

key_form

Direction	Type
Input	Character String

The key form you want to generate. Enter a 4-byte keyword specifying whether the key should be enciphered under the master key (OP) or the importer key-encrypting key (IM). The keyword must be left-justified and padded with blanks. Valid keyword values are OP for encryption under the master key or IM for encryption under the importer key-encrypting key. If you specify IM, you must specify an importer key-encrypting key in the *importer_key_identifier* parameter. For a *key_type* of IMP-PKA, this service supports only the OP *key_form*.

importer_key_identifier

Direction	Type
Input/Output	Character String

The importer key-encrypting key under which you want to encrypt the clear key. Specify either a 64-byte string of the internal key format or a key label. If you specify IM for the *key_form* parameter, the *importer_key_identifier* parameter is required.

key_identifier

Direction	Type
Input/Output	String

On input, this parameter contains a 64-byte key token.

- When the *key_type* parameter is TOKEN, a key token that ICSF will use to determine the key type:
  - A skeleton token of the key type to be imported. When the WRAPENH3 method is selected, a skeleton key token is required.
  - An internal token of the key type to be imported.
- Otherwise, a null token.

On output, this parameter receives the imported key token:

- Internal DES or AES key token for an operational key form, or
- External DES key tokens containing a key enciphered under the *key_encrypting_key_identifier* parameter.

When the imported *key_type* is IMPORTER or EXPORTER and the *key_form* is OP, and the application passes a valid internal key token for an IMPORTER or EXPORTER key in this parameter, the NOCV bit is propagated to the imported key token.

The Secure Key Import service does not adjust key parity.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

Restrictions

Triple-length DES keys are not supported.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 163. Required access control points for Secure Key Import

Key Form	Access control point
OP	Secure Key Import - DES, OP
IM	Secure Key Import - DES, IM

To use a NOCV key-encrypting key with the Secure Key Import service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

If the key-encrypting key identifier is a weaker key than the key being imported, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 164. Secure Key Import required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	

Table 164. Secure Key Import required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Secure Key Import2 (CSNBSKI2 and CSNESKI2)

Use this service to encipher a variable-length symmetric key under the AES master key or an AES IMPORTER KEK, depending on the Key Form rule provided. This service returns variable-length CCA key tokens and uses the AESKW wrapping method.

Some key types are not directly supported by this service because there is no default key usage value. Also, some key usage flags are not supported by this service. These key types can be created by using the TOKEN keyword and a skeleton token from the Key Token Build2 service.

The following AES key types require TOKEN to be used: DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW.

The callable service can execute only when ICSF is in special secure mode, which is described in the 'Special Secure Mode' topic in *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

The callable service name for AMODE(64) is CSNESKI2.

Format

```
CALL CSNBSKI2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             clear_key_bit_length,
             clear_key,
             key_name_length,
             key_name,
             user_associated_data_length,
             user_associated_data,
             key_encrypting_key_identifier_length,
             key_encrypting_key_identifier,
             target_key_identifier_length,
             target_key_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 3 or 4.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 165. Keywords for Secure Key Import2 Control Information

Keyword	Meaning
Token algorithm (One Required)	
HMAC	The target key identifier is to be an HMAC key.
AES	The target key identifier is to be an AES key.

Table 165. Keywords for Secure Key Import2 Control Information (continued)	
Keyword	**Meaning**
Key Form (One Required)	
OP	Specifies the key should be enciphered under the master key.
IM	Specifies the key should be enciphered under the key-encrypting key.
Key Type (One Required)	
CIPHER	The key type of the output token will be CIPHER. Only valid for AES algorithm.
EXPORTER	The key type of the output token will be EXPORTER. Only valid for AES algorithm.
IMPORTER	The key type of the output token will be IMPORTER. Only valid for AES algorithm.
MAC	MAC generation key.
MACVER	MAC verify key. Only valid for HMAC algorithm.
TOKEN	The key type will be determined from the key token supplied in the *target_key_identifier* parameter. ICSF does not check for the length of the key but uses the *clear_key_bit_length* parameter to determine the length of the key.
Payload version (One, optional) Note: This keyword overrides payload format version of any corresponding skeleton token.	
V0PYLD	The generated token will have the old variable length payload format. This is the default for AES CIPHER, EXPORTER, IMPORTER key types and is only valid with those key types.
V1PYLD	The generated token will have the new fixed length payload format. This is the default for AES MAC, PINPROT, PINCALC, PINPRW, and DKYGENKY key types. Not valid with the HMAC MAC key type.

clear_key_bit_length

Direction	Type
Input	Integer

The length of the value supplied in the *clear_key* parameter in bits. Valid lengths are 80 to 2048 for HMAC keys, and 128, 192, or 256 for AES keys.

clear_key

Direction	Type
Input	String

The value of the key to be imported. The value should be left justified and padded on the right with zeros to a byte boundary if the *clear_key_bit_length* is not a multiple of 8.

key_name_length

Direction	Type
Input	Integer

The length of the *key_name* parameter. Valid values are 0 and 64.

key_name

Direction	Type
Input	String

A 64-byte key store label to be stored in the associated data structure of the token.

user_associated_data_length

Direction	Type
Input	Integer

The length of the user-associated data. The valid values are 0 to 255 bytes.

user_associated_data

Direction	Type
Input	String

User-associated data to be stored in the associated data structure.

key_encrypting_key_identifier_length

Direction	Type
Input	Integer

The byte length of the *key_encrypting_key_identifier* parameter.

When Key Form is OP, the value must be 0.

When the Key Form is IM and:

- When the *key_encrypting_key_identifier* parameter contains a key token, the value must be between the actual length of the token and 9992.
- When the *key_encrypting_key_identifier* parameter contains a key label, the value must be 64.

key_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to wrap the imported key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

When the Key Form rule is OP, this parameter is ignored.

When the Key Form rule is IM, this parameter contains:

- For CCA keys, the identifier is a variable-length AES key token of key type IMPORTER with key management attributes enable to allow the key to wrap an AES or HMAC key.
- For X9.143 keys, the identifier is a AES key block of a key-encrypting key: key usage K0, algorithm A, and mode of use D.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master ke.y

target_key_identifier_length

Direction	Type
Input/Output	Integer

On input, the byte length of the buffer for the *target_key_identifier* parameter. The buffer must be large enough to receive the target key token. The maximum value is 900 bytes.

On output, the parameter will hold the actual length of the target key token.

target_key_identifier

Direction	Type
Input/Output	String

The output key token. On input, this parameter is ignored except when the Key Type keyword is TOKEN. If you specify the TOKEN keyword, then this field contains a valid token of the key type you want to import. On output, when Key Form is OP, this will be an internal variable-length symmetric token. When Key Form is IM, this will be an external variable-length symmetric token. See *rule_array* for a list of valid key types.

Usage notes

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 166. Required access control points for Secure Key Import2

Key Form	Access control point
OP	Secure Key Import2 – OP
IM	Secure Key Import2 – IM

When the **Prohibit weak wrapping - Transport keys** access control point is enabled, a key token wrapped with a weaker key will not be imported. When the **Warn when weak wrap - Transport keys** access control point is enabled, the reason code will indicate when the wrapping key is weaker than the key being imported.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 167. Secure Key Import2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Symmetric Key Export (CSNDSYX and CSNFSYX)

Use the Symmetric Key Export callable service to transfer an operational AES, DES, or HMAC key in a CCA key token from encryption under a master key to encryption under an RSA public key or AES EXPORTER key.

For an RSA-enciphered output key, the key is returned as an opaque data buffer or in an external variable-length symmetric key-token. If the key is returned as an opaque data buffer, the Symmetric Key Import service can be used along with the associated RSA private-key to import the key back into an operational symmetric key-token. If the key is returned in an external variable-length symmetric key-token, the Symmetric Key Import2 service can be used along with the associated RSA private-key to import the key.

For an AES-enciphered output key, the key is returned in an external variable-length symmetric key-token. The Symmetric Key Import2 service can be used along with its associated AES IMPORTER key-encrypting key to import the key. The usage attributes of the IMPORTER key must allow IMPORT.

Table 168 on page 418 and Table 169 on page 418 show which formatting methods can be used for each type of key token and a description of the enciphered key returned.

Table 168. CSNDSYX key formatting for fixed length AES and DES key tokens

Operational source key-tokenOperational source key-token	Key-formatting method keyword			
	AESKWCV	**PKCS-1.2**	**PKCSOAEP**	**ZERO-PAD**
AES DATA	Not supported	The output key is returned as an opaque data buffer after being formatted using the RSAES-PKCS1-v1_5 encryption / decryption scheme of the RSA PKCS #1 v2.0 standard and enciphered using the RSA public-key provided as a transport key.	The output key is returned as an opaque data buffer after being formatted using the RSAES-OAEP encryption / decryption scheme of the RSA PKCS #1 v2.0 standard and enciphered using the RSA public-key provided as a transport key.	The output key is returned as an opaque data buffer after the key is right-aligned, padded on the left to the necessary block length with bits valued to zero, and enciphered using the RSA public-key provided as a transport key.
DES DATA	The output key is returned in an external variable-length DES key-token with control vector after being enciphered using the AES EXPORTER key provided as the transport key.			
DES key types other than DATA	The output key is returned in an external variable-length DES key-token with control vector after being enciphered using the AES EXPORTER key provided as the transport key.	Not supported.	Not supported.	Not supported.

Table 169. CSNDSYX key formatting for variable length AES and HMAC key tokens

Operational source key-token	Key-formatting method keyword		
	AESKW	**PKOAEP2**	**CKM-RAKW**
AES	The output key is returned in an external variable-length AES key-token after being enciphered using he AES EXPORTER key provided as the transport key.	The output key is returned in an external variable length AES key token after being formatted using the RSAES-OAEP encryption / decryption scheme of the RSA PKCS #1 v2.1 standard and enciphered using the RSA public-key provided as a transport key.	The output key is returned as output structure corresponding to the output from the PKCS#11 mechanism CKM_RSA_AES_KEY_WRAP and enciphered using the RSA public-key provided as a transport key.
HMAC	Same as the variable-length AES source key-token, except that the output key is returned in an external variable-length HMAC key-token.	Same as the variable-length AES source key-token, except that the output key is returned in an external variable-length HMAC key-token.	Not supported.

Notes:

1. For keywords PKCS-1.2, PKCSOAEP, and PKOAEP2, see "Formatting hashes and keys in public-key cryptography" on page 1650.
2. The RSA PKCS #1 v2.0 standard for the RSAES-PKCS1-v1_5 encryption/decryption scheme is formerly known as block-type 02 format.
3. PKCSOAEP and PKOAEP2 are the only key formatting methods that use a hash method. PKCSOAEP and PKOAEP2 can specify either SHA-1 or SHA-256. PKOAEP2 can also specify SHA-384 or SHA-512.

The callable service name for AMODE(64) is CSNFSYX.

Format

```
CALL CSNDSYX(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              source_key_identifier_length,
              source_key_identifier,
              transporter_key_identifier_length,
              transporter_key_identifier,
              enciphered_key_length,
              enciphered_key)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. Value may be 1, 2, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Table 170 on page 420 lists the keywords. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 170. Keywords for Symmetric Key Export Control Information

Keyword	Meaning
Token Algorithm (One keyword, optional)	
AES	The key being exported is an AES key. If *source_key_identifier* is a variable-length symmetric key token or label, only the CKM-RAKW, PKOAEP2 and AESKW key formatting methods are supported.
DES	The key being exported is a DES key. This is the default.
HMAC	The key being exported is an HMAC key. Only the PKOAEP2 and AESKW key formatting methods are supported.
Key Formatting method (One required)	
AESKW	Specifies that the key is to be formatted using AESKW and placed in an external variable length CCA token. The *transport_key_identifier* must be an AES EXPORTER. This rule is not valid with the DES Algorithm keyword or with AES DATA (version X'04') keys.
AESKWCV	Specifies that the key is to be formatted using AESKW and placed in a symmetric variable length CCA token of type DESUSECV. The *transport_key_identifier* must be an AES EXPORTER key. The DES control vector and other significant token information will be in the associated data section of the variable length key token. Only valid with the DES token algorithm.
CKM-RAKW	Specifies to return the key in an external AES wrapped PKCS#11 object. The variable-length symmetric key-token will be returned in an output structure corresponding to the output from PKCS#11 mechanism CKM_RSA_AES_KEY_WRAP. Valid only with the AES algorithm.
PKCSOAEP	Specifies to format the key according to the method in RSA DSI PKCS #1V2 OAEP. The default hash method is SHA-1. Use the SHA-256 keyword for the SHA-256 hash method.
PKCS-1.2	Specifies to format the key according to the method found in RSA DSI PKCS #1 block type 02 to recover the symmetric key.

Table 170. Keywords for Symmetric Key Export Control Information (continued)	
Keyword	**Meaning**
PKOAEP2	Specifies to format the key according to the method found in RSA DSI PKCS #1 v2.1 RSAES-OAEP documentation. Not valid with DES algorithm or with AES DATA (version X'04') keys. A hash method is required.
ZERO-PAD	The clear key is right-justified in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length).
Hash Method (One, optional for PKCSOAEP, required for PKOAEP2. Not valid with any other Key Formatting method)	
SHA-1	Specifies to use the SHA-1 hash method to calculate the OAEP message hash. This is the default for PKCSOAEP.
SHA-256	Specifies to use the SHA-256 hash method to calculate the OAEP message hash.
SHA-384	Specifies to use the SHA-384 hash method to calculate the OAEP message hash. Not valid with PKCSOAEP.
SHA-512	Specifies to use the SHA-512 hash method to calculate the OAEP message hash. Not valid with PKCSOAEP.
Certificate validation method (One required when the input is an X.509 certificate. Otherwise, must not be specified.)	
RFC-2459	Validate the certificate using the semantics of RFC-2459.
RFC-3280	Validate the certificate using the semantics of RFC-3280.
RFC-5280	Validate the certificate using the semantics of RFC-5280.
RFC-ANY	Attempt to validate the certificate by first using the semantics of RFC-2459, then the semantics of RFC-3280, and finally, the semantics of RFC-5280.
Public Key Infrastructure Usage (One optional when the input is an X.509 certificate. Otherwise, must not be specified.)	
PKI-CHK	Specifies that the X.509 certificate is to be validated against the trust chain of the PKI hosted in the adapter. This requires that the CA credentials have been installed into all coprocessors using the Trusted Key Entry workstation. This is the default.
PKI-NONE	Specifies that the X.509 certificate is not to be validated against the trust chain of the PKI hosted in the adapter. This is suitable if the certificate has been validated using host-based PKI services or if using a self-signed certificate.

source_key_identifier_length

Direction	**Type**
Input	Integer

The length in bytes of the *source_key_identifier* parameter. The minimum size is 64 bytes. If the *source_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

source_key_identifier

Direction	Type
Input/Output	String

The key to be exported and wrapped by the *transport_key_identifier*. The key identifier is an operational token or the key label of an operational token in key storage.

The key in the key identifier must match the algorithm in the *rule_array*. DES is the default algorithm.

For formatting method rules PKCSOAEP, PKCS-1.2, and ZERO-PAD, the source key is an AES or DES DATA key in a fixed-length key token.

For rule AESKWCV, the source key is a DES key of any type in a fixed-length key token.

For rules AESKW and PKOAEP2, the source key is an AES or HMAC key of any type in a variable-length key token.

For rule CKM-RAKW, the source key is an AES CIPHER key in a variable-length key token.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

transporter_key_identifier_length

Direction	Type
Input	Integer

The length of the *transporter_key_identifier* parameter.

When the *transporter_key_identifier* parameter contains a label, the value is 64.

Otherwise, the value must be between the actual length of the token and maximum allowed. When the parameter contains:

- An RSA key token, the maximum value is 3500.
- A CCA AES key token, the maximum value is 725.
- A TR-31 key block, the maximum value is 9992.

transporter_key_identifier

Direction	Type
Input	String

The identifier of the key to wrap the source key in a formatted data buffer or external key token. The key identifier is an operational key token or key block, the key label of an operational token or block in key storage, or an X.509 certificate containing the public key.

When the AESKW or AESKWCV key formatting method is specified, the identifier must be an AES key-encrypting key:

- For CCA keys, the key is an AES EXPORTER with the EXPORT attribute enabled in the key-usage field. The key usage wrap algorithm attribute must match the algorithm of the source key. The key usage wrap class attribute must match the class of the source key.
- Fox X9.143 (TR-31) keys, the key usage must be K0, the algorithm A, and the mode of use E.

Otherwise, this parameter must be the token or label of an RSA public or private key token, or the X.509 certificate containing the RSA public key.

Certificates may be PEM-formatted EBCDIC text or DER-encoded. The certificate may either have no key usage attribute, or it must have the following usage: keyEncipherment.

When the identifier is an AES EXPORTER and the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

enciphered_key_length

Direction	Type
Input/Output	Integer

The length of the *enciphered_key* parameter. This is updated with the actual length of the *enciphered_key* generated. The maximum size you can specify in this parameter is 900 bytes, although the actual key length may be further restricted by your hardware configuration (as shown in Table 173 on page 424).

enciphered_key

Direction	Type
Output	String

The exported key in the specified format wrapped by the RSA public or AES EXPORTER key specified in the *transporter_key_identifier* field.

Usage notes

If ICSF is configured to audit the lifecycle of tokens (for example, AUDITKEYLIFECKDS(TOKEN(YES),...) is specified) and a token is passed as input to be exported, a request is made to the Crypto Express Coprocessor to generate the key fingerprint to be used for auditing the exported key.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If an RSA public key is specified as the *transporter_key_identifier*, the hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.

When wrapping an AES key with an RSA public key, the RSA key used must have a modulus size greater than or equal to the total PKOAEP2 message bit length (key size + total overhead).

Table 171. Minimum RSA modulus strength required to contain a PKOAEP2 block when exporting an AES key

AES key size	Total message sizes (and therefore minimum RSA key size) when the Hash Method is:			
	SHA-1	SHA-256	SHA-384	SHA-512
128 bits	736 bits	928 bits	1184 bits	1440 bits
192 bits	800 bits	992 bits	1248 bits	1504 bits
256 bits	800 bits	1056 bits	1312 bits	1568 bits

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 172. Required access control points for Symmetric Key Export		
Key formatting method	**Token Algorithm**	**Access control point**
PKCSOAEP	AES	Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2
	DES	Symmetric Key Export - DES, PKCS-1.2
PKCS-1.2	AES	Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2
	DES	Symmetric Key Export - DES, PKCS-1.2
ZERO-PAD	AES	Symmetric Key Export - AES, ZERO-PAD
	DES	Symmetric Key Export - DES, ZERO-PAD
PKOAEP2	HMAC	Symmetric Key Export - HMAC, PKOAEP2
	AES	Symmetric Key Export - AES, PKOAEP2
AESKW	AES or HMAC	Symmetric Key Export - AESKW
AESKWCV	DES	Symmetric Key Export - AESKWCV
CKM-RAKW	AES	Symmetric Key Export - CKM-RAKW

If the transport key identifier is a weaker key than the key being exported, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 173. Symmetric Key Export required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. Rule array keyword CKM-RAKW is not supported. X9.143 key blocks are not supported.

Table 173. Symmetric Key Export required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. Rule array keyword CKM-RAKW is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE require the July 2019 or later licensed internal code (LIC). X.509 certificates require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Rule array keyword CKM-RAKW is not supported. X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Rule array keyword CKM-RAKW is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keyword CKM-RAKW is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keyword CKM-RAKW requires the CCA release 7.4 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	Rule array keyword CKM-RAKW is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Symmetric Key Export with Data (CSNDSXD and CSNFSXD)

Export a symmetric key, along with some application supplied data, encrypted using an RSA key. The clear key data will be copied into the provided data field at offset data_offset then encrypted using the PKCS-1.5 block type 2 formatting algorithm.

The callable service name for AMODE(64) is CSNDSXD.

Format

```
CALL CSNDSXD(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             source_key_identifier_length,
             source_key_identifier,
             data_length,
             data_offset,
             data,
             RSA_public_key_identifier_length,
             RSA_public_key_identifier,
             RSA_enciphered_key_length,
             RSA_enciphered_key)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 2.

rule_array

Direction	Type
Input	String

The keywords that provide control information to the callable service. The following table provides a list. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 174. Keywords for Symmetric Key Export with Data (CSNDSXD)

Keyword	Meaning
Algorithm (one required)	
AES	The key specified in source_key_identifier is an AES key.
DES	The key specified in source_key_identifier is a DES key.
Key Formatting method (one required)	
PKCS-EXT	Copy the clear key data (length determined by the key length in the source key token) into the provided data field at offset *data_offset* then encrypt using the PKCS-1.5 block type 2 formatting algorithm.

source_key_identifier_length

Direction	Type
Input	Integer

The length of the *source_key_identifier* parameter.

When the *source_key_identifier* parameter contains a label, the value is 64. Otherwise, the value must be between the actual length of the token and 9992.

source_key_identifier

Direction	Type
Input	String

The identifier of the key to be exported. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

The key token or key block must allow the key to be exported. For CCA DES keys, bit 17 of the control vector must be equal to '1'b (XPORT-OK). For CCA AES keys in the variable-length token, the key management usage must allow export by RSA keys and by unauthenticated asymmetric keys. The TR-31 keys, the exportability cannot be ‘N’ (none).

When the 'Allow Symmetric Key Export with Data Special' access control is disabled,

- CCA DES keys with a control vector of DATAC or DKYGENKY with subtype DKYL0 are allowed to be exported.
- CCA AES CIPHER or DATA keys are allowed to be exported.
- TR-31 keys that are allowed to be exported:

- key usage B3, algorithm T and mode of use X.
- key usage D0, algorithm T or A, and mode of use B, D, or E.

When the 'Allow Symmetric Key Export with Data Special' access control is enabled,

- CCA keys: any type of DES or AES key may be exported.
- TR-31 keys: Key usage *, algorithm A or T, mode of use *.

If the token or key block supplied was encrypted under the old master key, the token or key block will be returned encrypted under the current master key.

data_length

Direction	Type
Input	Integer

The length of *data* in bytes. The maximum value is the length of the modulus (in bytes) of the RSA_public_key_identifier minus 11. The overall maximum value is 501.

data_offset

Direction	Type
Input	Integer

The offset from the start of data at which the clear DES or AES key is to be copied. The maximum value is *data_length* - key length of clear source key.

data

Direction	Type
Input	String

The clear data. The deciphered key from *source_key_identifier* is copied into this data at the specified *offset*, and then encrypted with the key from the *RSA_public_key_identifier*.

RSA_public_key_identifier_length

Direction	Type
Input	Integer

The length of the *RSA_public_key_identifier* field in bytes. This value is 64 when a label is supplied. When the key identifier is a key token, the value is the length of the token. The maximum value is 3500.

RSA_public_key_identifier

Direction	Type
Input	String

A PKA96 RSA internal or external key-token with the RSA public key of the remote node that is to import the exported key.

RSA_enciphered_key_length

Direction	Type
Output	Integer

The length of the *RSA_enciphered_key* field in bytes. On output, the variable is updated with the actual length of the *RSA_enciphered_key* parameter. The maximum length is 512.

RSA_enciphered_key

Direction	Type
Output	String

The exported RSA-enciphered key.

Usage notes

If ICSF is configured to audit the lifecycle of tokens (for example, AUDITKEYLIFECKDS(TOKEN(YES),...) is specified) and a token is passed as input to be exported, a request is made to the Crypto Express Coprocessor to generate the key fingerprint to be used for auditing the exported key.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 175. Required access control points for Symmetric Key Export with Data

Access control point	Restrictions
Symmetric Key Export with Data	None
Symmetric Key Export with Data - special	Allow source keys that are not DATAC or DKYGENKY with subtype DKYL0

In addition, the service requires the following access control points to be enabled based on the key-formatting method and the token algorithm.

Table 176. Required access control points based on the key-formatting method and the token algorithm

Key-formatting method	Token algorithm	Access control points
PKCS-EXT	AES	**Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2**
	DES	**Symmetric Key Export - DES, PKCS-1.2**

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 177. Symmetric Key Export with Data required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 177. Symmetric Key Export with Data required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Symmetric Key Generate (CSNDSYG and CSNFSYG)

Use the Symmetric Key Generate callable service to generate an AES DATA or DES DATA key and return the key in two forms: enciphered under the master key and encrypted under an RSA public key.

You can import the RSA public key encrypted form by using the symmetric key import service at the receiving node.

Also use the Symmetric Key Generate callable service to generate any DES importer or exporter key-encrypting key encrypted under an RSA public key according to the PKA92 formatting structure. See “PKA92 key format and encryption process” on page 1649 for more details about PKA92 formatting.

The callable service name for AMODE(64) invocation is CSNFSYG.

Format

```
CALL CSNDSYG(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_encrypting_key_identifier,
             RSA_public_key_identifier_length,
             RSA_public_key_identifier,
             local_enciphered_key_token_length,
             local_enciphered_key_token,
             RSA_enciphered_key_length,
             RSA_enciphered_key)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1, 2, 3, 4, 5, 6, or 7.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Table 178 on page 431 lists the keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 178. Keywords for Symmetric Key Generate Control Information

Keyword	Description	Algorithm
Algorithm (one keyword, optional)		

Table 178. Keywords for Symmetric Key Generate Control Information (continued)

Keyword	Description	Algorithm
AES	The key being generated is a secure AES key.	AES
DES	The key being generated is a DES key. This is the default.	DES
Key formatting method (one keyword required)		
PKA92	Specifies the key-encrypting key is to be encrypted under a PKA96 RSA public key according to the PKA92 formatting structure.	DES
PKCSOAEP	Specifies using the method found in RSA DSI PKCS #1V2 OAEP. The default hash method is SHA-1. Use the SHA-256 keyword for the SHA-256 hash method.	AES or DES
PKCS-1.2	Specifies the method found in RSA DSI PKCS #1 block type 02.	AES or DES
ZERO-PAD	The clear key is right-justified in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length).	AES or DES
Key Length (optional - for use with PKA92)		
SINGLE-R	For key-encrypting keys, this specifies that the left half and right half of the generated key will have identical values. This makes the key operate identically to a single-length key with the same value. Without this keyword, the left and right halves of the key-encrypting key will each be generated randomly and independently.	DES
Key Length (optional - for use with PKCSOAEP, PKCS-1.2, or ZERO-PAD)		
SINGLE, KEYLN8	Specifies that the generated key should be 8 bytes in length.	DES
DOUBLE	Specifies that the generated key should be 16 bytes in length.	DES
TRIPLE	Specifies that the generated key should be 24 bytes in length.	DES
KEYLN16	Specifies that the generated key should be 16 bytes in length.	AES or DES
KEYLN24	Specifies that the generated key should be 24 bytes in length.	AES or DES
KEYLN32	Specifies that the generated key should be 32 bytes in length.	AES
Encipherment method for the local enciphered copy of the key (optional - for use with PKCSOAEP, PKCS-1.2, or ZERO-PAD		

Table 178. Keywords for Symmetric Key Generate Control Information (continued)

Keyword	Description	Algorithm
OP	Enciphers the key with the master key. The DES master key is used with DES keys and the AES master key is used with AES keys.	AES or DES
EX	Enciphers the key with the EXPORTER key that is provided through the *key_encrypting_key_identifier* parameter.	DES
IM	Enciphers the key with the IMPORTER key-encrypting key specified with the *key_encrypting_key_identifier* parameter.	DES
Key Wrapping Method (optional)		
USECONFG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword. The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set.	DES
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.	DES
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This is the default for triple-length keys	DES
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.	DES
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.	DES
Translation Control (optional)		
ENH-ONLY	Restrict rewrapping of the *target_key_identifier* token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method. This is the default when the wrapping method is WRAPENH2 or WRAPENH3.	DES
Hash Method (optional - only valid with PKCSOAEP)		
SHA-1	Specifies to use the SHA-1 hash method to calculate the OAEP message hash. This is the default.	AES or DES

Table 178. Keywords for Symmetric Key Generate Control Information (continued)		
Keyword	**Description**	**Algorithm**
SHA-256	Specifies to use the SHA-256 hash method to calculate the OAEP message hash.	AES or DES
Certificate validation method (One required when the input is an X.509 certificate. Otherwise, must not be specified.)		
RFC-2459	Validate the certificate using the semantics of RFC-2459.	
RFC-3280	Validate the certificate using the semantics of RFC-3280.	
RFC-5280	Validate the certificate using the semantics of RFC-5280.	
RFC-ANY	Attempt to validate the certificate by first using the semantics of RFC-2459, then the semantics of RFC-3280, and finally, the semantics of RFC-5280.	
Public Key Infrastructure Usage (One optional when the input is an X.509 certificate. Otherwise, must not be specified.)		
PKI-CHK	Specifies that the X.509 certificate is to be validated against the trust chain of the PKI hosted in the adapter. This requires that the CA credentials have been installed into all coprocessors using the Trusted Key Entry workstation. This is the default.	
PKI-NONE	Specifies that the X.509 certificate is not to be validated against the trust chain of the PKI hosted in the adapter. This is suitable if the certificate has been validated using host-based PKI services or if using a self-signed certificate.	

key_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to wrap the generated key. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage. The maximum length of the key token or block is 9992 bytes.

- When the *rule_array* specifies IM, the key identifier is a:
  - 64-byte CCA DES key token of key type IMPORTER.
  - Variable-length TR-31 DES key block of an importer key-encrypting key: key usage K0, algorithm T, and mode of use D.
- When the *rule_array* specifies EX, the key identifier is a:
  - 64-byte CCA DES key token of key type EXPORTER.

  - Variable-length TR-31 DES key block of an exporter key-encrypting key. Key usage K0, algorithm T, and mode of use E.
- Otherwise, the parameter is ignored.

When the token or key block supplied was encrypted under the old master key, the token or key block will be returned encrypted under the current master key.

RSA_public_key_identifier_length

Direction	Type
Input	Integer

The length of the *RSA_public_key_identifier* parameter. If the *RSA_public_key_identifier* parameter is a label, this parameter specifies the length of the label. The maximum size is 3500 bytes.

RSA_public_key_identifier

Direction	Type
Input	String

The token or label of an RSA public or private key token, or the X.509 certificate containing the RSA public key to be used for protecting the generated symmetric key.

Certificates may be PEM-formatted EBCDIC text or DER-encoded. The certificate may either have no key usage attribute or it must have the following usage: keyEncipherment.

local_enciphered_key_token_length

Direction	Type
Input/Output	Integer

The length in bytes of the *local_enciphered_key_token*.

On input, this is size of the buffer to receive the generated key token. The minimum length is 16 bytes and the maximum length is 9992 bytes.

On output, this field is updated with the actual length of the token that is generated.

local_enciphered_key_token

Direction	Type
Input/Output	String

This parameter contains the generated key token in the form of an internal or external token, depending on *rule_array* specification.

On input, these are the requirements for the key identifier:

Table 179. requirements for the key identifier

Rule array keyword	Desired output key identifier	
	CCA key token	TR-31 key block
PKA92	64-byte internal operational or skeleton token of a DES IMPORTER or EXPORTER key.	N/A

Table 179. requirements for the key identifier (continued)

Rule array keyword	Desired output key identifier	
	CCA key token	TR-31 key block
DES with PKCSOAEP PKCS-1.2 ZEROPAD	64-byte null.	Skeleton block of a DES data-encrypting key (key usage D0, algorithm T, and mode use B, D or E).
AES with PKCSOAEP PKCS-1.2 ZEROPAD	64-byte null.	Skeleton block of an AES data-encrypting key (key usage D0, algorithm A, and mode use B, D or E).

To generate a compliant-tagged output key, specify a compliant-tagged key. To generate a compliant-tagged key block, the IBM optional block "10" with the Compliance Tag attribute enabled must be included with the skeleton key block.

When the WRAPENH3 method is selected, a skeleton key token is required. A secure internal key token wrapped with the WRAPENH3 method obfuscates the key length.

The DES key wrapping methods available are described in "CCA key wrapping" on page 20.

RSA_enciphered_key_length

Direction	Type
Input/Output	Integer

The length of the *RSA_enciphered_key* parameter. This service updates this field with the actual length of the *RSA_enciphered_key* it generates. The maximum size is 512 bytes.

RSA_enciphered_key

Direction	Type
Input/Output	String

This field contains the RSA enciphered key, which is protected by the public key specified in the *RSA_public_key_identifier* field. When using the PKA92 keyword, specify a compliant-tagged key on input to generate a compliant-tagged output key.

Usage notes

If ICSF is configured to audit the lifecycle of tokens (for example, AUDITKEYLIFECKDS(TOKEN(YES),...) is specified), a request is made to the Crypto Express Coprocessor to generate the key fingerprint to be used for auditing the exported key.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.

Specification of PKA92 with an input NOCV key-encrypting key token is not supported.

Use the PKA92 key-formatting method to generate a key-encrypting key. The service enciphers one key copy using the key encipherment technique employed in the IBM Transaction Security System (TSS)

4753, 4755, and AS/400 cryptographic product PKA92 implementations (see "PKA92 key format and encryption process" on page 1649). The control vector for the RSA-enciphered copy of the key is taken from an internal (operational) DES key token that must be present on input in the *RSA_enciphered_key* variable. Only key-encrypting keys that conform to the rules for an OPEX case under the key generate service are permitted. The control vector for the local key is taken from a DES key token that must be present on input in the *local_enciphered_key_token* variable. The control vector for one key copy must be from the EXPORTER class while the control vector for the other key copy must be from the IMPORTER class.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 180. Required access control points for Symmetric Key Generate

Key algorithm	Key formatting rule	Access control point
DES	PKCS-1.2	Symmetric Key Generate - DES, PKCS-1.2
DES	ZERO-PAD	Symmetric Key Generate - DES, ZERO-PAD
DES	PKA92	Symmetric Key Generate - DES, PKA92
AES	PKCSOAEP, PKCS-1.2	Symmetric Key Generate - AES, PKCSOAEP, PKCS-1.2
AES	ZERO-PAD	Symmetric Key Generate - AES, ZERO-PAD

When the key wrapping method keyword specifies a wrapping method that is not the default method, the **Symmetric Key Generate - Allow wrapping override keywords** access control point must be enabled.

If the RSA key identifier is a weaker key than the key being generated, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 181. Symmetric Key Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE require the July 2019 or later licensed internal code (LIC). X.509 certificates require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 181. Symmetric Key Generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE are not supported. X.509 certificates are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Symmetric Key Import (CSNDSYI and CSNFSYI)

Use the Symmetric Key Import callable service to import a symmetric AES DATA or DES DATA key enciphered under an RSA public key. It returns the key in operational form, enciphered under the master key.

This service also supports the import of a PKA92-formatted DES key-encrypting key encrypted under a PKA96 RSA public key.

The callable service name for AMODE(64) is CSNFSYI.

Format

```
CALL CSNDSYI(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             RSA_enciphered_key_length,
             RSA_enciphered_key,
             RSA_private_key_identifier_length,
             RSA_private_key_identifier,
             target_key_identifier_length,
             target_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 1, 2, 3, 4, or 5.

rule_array

Direction	Type
Input	String

The keywords that provide control information to the callable service. Table 182 on page 440 provides a list. The recovery method is the method to use to recover the symmetric key. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 182. Keywords for Symmetric Key Import Control Information

Keyword	Meaning
Algorithm (one keyword, optional)	

Table 182. Keywords for Symmetric Key Import Control Information (continued)	
Keyword	**Meaning**
AES	The key being imported is an AES key.
DES	The key being imported is a DES key. This is the default.
Recovery Method (required)	
PKA92	Supported by the DES algorithm. Specifies the key-encrypting key is encrypted under a PKA96 RSA public key according to the PKA92 formatting structure.
PKCSOAEP	Specifies to use the method found in RSA DSI PKCS #1V2 OAEP. Supported by the DES and AES algorithms. The default hash method is SHA-1. Use the SHA-256 keyword for the SHA-256 hash method.
PKCS-1.2	Specifies to use the method found in RSA DSI PKCS #1 block type 02. Supported by the DES and AES algorithms.
ZERO-PAD	The clear key is right-justified in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length). Supported by the DES and AES algorithms.
Key Wrapping Method (optional)	
USECONFG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword. The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set.
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This is the default for triple-length keys.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.
Translation Control (optional)	
ENH-ONLY	Restrict rewrapping of the *target_key_identifier* token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method. This is the default when the wrapping method is WRAPENH2 or WRAPENH3.
Hash Method (optional - only valid with PKCSOAEP)	
SHA-1	Specifies to use the SHA-1 hash method to calculate the OAEP message hash. This is the default.
SHA-256	Specifies to use the SHA-256 hash method to calculate the OAEP message hash.

RSA_enciphered_key_length

Direction	Type
Input	Integer

The length of the *RSA_enciphered_key* parameter. The maximum size is 512 bytes.

RSA_enciphered_key

Direction	Type
Input	String

The key to import, protected under an RSA public key. The encrypted key is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the encrypted key. This string is left-justified within the *RSA_enciphered_key* parameter.

RSA_private_key_identifier_length

Direction	Type
Input	Integer

The length of the *RSA_private_key_identifier* parameter. When the *RSA_private_key_identifier* parameter is a key label, this field specifies the length of the label. The maximum size is 3500 bytes.

RSA_private_key_identifier

Direction	Type
Input	String

An internal RSA private key token or label whose corresponding public key protects the symmetric key.

target_key_identifier_length

Direction	Type
Input/Output	Integer

The length of the *target_key_identifier* parameter. This field is updated with the actual length of the *target_key_identifier* that is generated. The size must be 64 bytes.

target_key_identifier

Direction	Type
Output	String

This field contains the internal token of the imported symmetric key. Except for PKA92 processing, this service produces a DATA key token with a key of the same length as that contained in the imported token.

When the *RSA_private_key_identifier* is compliant-tagged, the key is imported as a compliant-tagged key token.

Restrictions

The exponent of the RSA public key must be odd.

Usage notes

If ICSF is configured to audit the lifecycle of tokens (for example, AUDITKEYLIFECKDS(TOKEN(YES),...) is specified), a request is made to the Crypto Express Coprocessor to generate the key fingerprint to be used for auditing the imported key.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit. The service will fail with return code 12 and reason code 11020.

Specification of PKA92 with an input NOCV key-encrypting key token is not supported.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 183. Required access control points for Symmetric Key Import

Key algorithm	Key formatting rule	Access control point
DES	PKCS-1.2	Symmetric Key Import - DES, PKCS-1.2
DES	PKA92 KEK	Symmetric Key Import - DES, PKA92 KEK
DES	ZERO-PAD	Symmetric Key Import - DES, ZERO-PAD
AES	PKCSOAEP, PKCS-1.2	Symmetric Key Import - AES, PKCSOAEP, PKCS-1.2
AES	ZERO-PAD	Symmetric Key Import - AES, ZERO-PAD

When the key wrapping method keyword specifies a wrapping method that is not the default method, the **Symmetric Key Import - Allow wrapping override keywords** access control point must be enabled.

When the **Symmetric Key Import2 - disallow weak import** access control is enabled, a key token wrapped with a weaker key will not be imported. When the **Warn when weak wrap - Transport keys** access control is enabled, the reason code will indicate when the wrapping key is weaker than the key being imported.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 184. Symmetric Key Import required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
	Crypto Express7 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC).
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)

Use the Symmetric Key Import2 callable service to import an HMAC, AES or DES key enciphered under an RSA public key or AES EXPORTER key. It returns the key in operational form, enciphered under the master key.

This service returns a variable-length CCA key token wrapped using the mode configured as the default wrapping mode or specified by a rule array keyword.

The callable service name for AMODE(64) is CSNFSYI2.

Format

```
CALL CSNDSYI2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             enciphered_key_length,
             enciphered_key,
             transport_key_identifier_length,
             transport_key_identifier,
             key_name_length,
             key_name,
             target_key_identifier_length,
             target_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 2, 3 or 4.

rule_array

Direction	Type
Input	String

The keywords that provide control information to the callable service. The following table provides a list. The recovery method is the method to use to recover the symmetric key. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 185. Keywords for Symmetric Key Import2 Control Information

Keyword	Meaning
Token Algorithm (One required)	
AES	The key being imported is an AES key.
DES	The key being imported is a DES key.
HMAC	The key being imported is an HMAC key.
Recovery Method (Required)	
AESKW	Specifies the enciphered key has been wrapped with the AESKW formatting method.
AESKWCV	Specifies the enciphered key has been wrapped with the AESKWCV formatting method with a key type of DESUSECV.
PKOAEP2	Specifies to use the method found in RSA DSI PKCS #1 v2.1 RSAES-OAEP documentation.
Key Wrapping Method (Optional, valid only for DES algorithm.	
USECONFG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This is the default for triple-length keys.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.
WRAP-ECB	Specifies that the original wrapping method is to be used.
Translation Control (Optional, valid only for enhanced wrapping)	
ENH-ONLY	Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the original method. This restricts translation to the original method. This is the default when the wrapping method is WRAPENH2 or WRAPENH3.

enciphered_key_length

Direction	Type
Input	Integer

The length of the *enciphered_key* parameter. The maximum size is 900 bytes.

enciphered_key

Direction	Type
Input	String

This is the key to be imported. The key is in an external variable-length symmetric key token. The key can be either:

- An HMAC or AES key with an RSA-enciphered payload (PKOAEP2).
- An AES and HMAC key with an AES-enciphered payload (AESKW).
- A DES key with key type DESUSECV with an AES-enciphered payload (AESKWCV).

transport_key_identifier_length

Direction	Type
Input	Integer

The length of the *transport_key_identifier* parameter.

When the *transport_key_identifier* parameter contains:

- A key label, this field must be 64.
- An RSA private key, the value must be between the actual length of the key token and 3500.
- An AES operational key token, the value must be between the actual length of the key token and 725.
- A TR-31 operational key block, the value must be between the actual length of the key token and 9992.

transport_key_identifier

Direction	Type
Input	String

The identifier of the key-encrypting key to unwrap the source key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

When the AESKW or AESKWCV key formatting method is specified, this parameter must contain an AES key-encrypting key:

- For CCA keys, the key is an AES IMPORTER with the IMPORT attribute enabled in the key-usage field.
- For TR-31 keys, the key is an AES importer: key usage is K0, algorithm is A, and mode of use is D.

Otherwise, this parameter must be an RSA internal private key.

If the token or key block supplied is an AES IMPORTER and was encrypted under the old master key, the token or key block will be returned encrypted under the current master key.

key_name_length

Direction	Type
Input	Integer

The length of the key_name parameter for *target_key_identifier*. Valid values are 0 and 64. For the DES token algorithm, key_name_length must be 0.

key_name

Direction	Type
Input	String

A 64-byte key store label to be stored in the associated data structure of *target_key_identifier*.

target_key_identifier_length

Direction	Type
Input/Output	Integer

On input, the byte length of the buffer for the *target_key_identifier* parameter. The buffer must be large enough to receive the target key token. The maximum value is 725 bytes.

On output, the parameter will hold the actual length of the target key token.

target_key_identifier

Direction	Type
Output	String

This parameter contains the internal token of the imported symmetric key.

When the *transport_key_identifier* is compliant-tagged, the key is imported as a compliant-tagged key token.

Restrictions

The exponent of the RSA public key must be odd.

Usage notes

If ICSF is configured to audit the lifecycle of tokens (for example, AUDITKEYLIFECKDS(TOKEN(YES),...) is specified), a request is made to the Crypto Express Coprocessor to generate the key fingerprint to be used for auditing the imported key.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This is the message layout used to encode the key material exported with the new PKOAEP2 formatting method.

Table 186. PKCS#1 OAEP encoded message layout (PKOAEP2)

Field	Size	Value
Hash field	32 Bytes	SHA-256 hash of associated data section in the source key identifier
Key Bit Length	2 Bytes	variable
Key Material	Byte length of the key material (rounded up to the nearest byte)	variable

Hash field
The associated data for the HMAC variable length token is hashed using SHA-256. Specifically referring to vartoken.h, this is the "VarAssocData_t AD" section of the VarKeyTkn_t structure, for the full length indicated in the 'SectLn' field of the VarAssocData_t.

Key Bit Length
A 2 Byte key bit length field.

Key Material
The key material is padded to the nearest byte with '0' bits.

Access control points

This table lists the access control points in the domain role that control the function for this service.

Table 187. Symmetric Key Import2 Access Control Points

Key formatting method	Token Algorithm	Access control point
PKOAEP2	HMAC	Symmetric Key Import2 - HMAC, PKOAEP2
	AES	Symmetric Key Import2 - AES, PKOAEP2
AESKW	HMAC, AES	Symmetric Key Import2 - AESKW
AESKWCV	DES	Symmetric Key Import2 - AESKWCV

When the **Symmetric Key Import2 - disallow weak import** access control point is enabled, a key token wrapped with a weaker key will not be imported. When the **Warn when weak wrap - Transport keys** access control point is enabled, the reason code will indicate when the wrapping key is weaker than the key being imported.

If the token algorithm is DES and the key wrapping method keyword specifies a wrapping method that is not the default method, the **Symmetric Key Import2 – Allow wrapping override keywords** access control point must be enabled.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 188. Symmetric Key Import2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 188. Symmetric Key Import2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Trusted Block Create (CSNDTBC and CSNFTBC)

This callable service is used to create a trusted block in a two step process. The block will be in external form, encrypted under an IMP-PKA transport key. This means that the MAC key contained within the trusted block will be encrypted under the IMP-PKA key.

The callable service name for AMODE(64) invocation is CSNFTBC.

Format

```
CALL CSNDTBC(
             return_code,
```

```
                reason_code,
                exit_data_length,
                exit_data,
                rule_array_count,
                rule_array,
                input_block_length,
                input_block_identifier,
                transport_key_identifier,
                trusted_block_length,
                trusted_block_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the specific results of processing. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. This number must be 1.

rule_array

Direction	Type
Input	String

Specifies a string variable containing an array of keywords. The keywords are 8 bytes long and must be left-justified and right padded with blanks

This table lists the rule_array keywords for this callable service.

Table 189. Rule_array keywords for Trusted Block Create (CSNDTBC)

Keyword	Meaning
Operational Keywords - One Required	
INACTIVE	Create the trusted block, but in inactive form. The MAC key is randomly generated, encrypted with the transport key, and inserted into the block. The ACTIVE flag is set to False (0), and the MAC is calculated over the block and inserted in the appropriate field. The resulting block is fully formed and protected, but it is not usable in any other CCA services. Use of the INACTIVE keyword is authorized by the 0x030F access control point.
ACTIVATE	This makes the trusted block usable in CCA services. Use of the ACTIVATE keyword is authorized by the 0x0310 access control point.

input_block_length

Direction	Type
Input/Output	String

Specifies the number of bytes of data in the input_block_identifier parameter. The maximum length is 3500 bytes.

input_block_identifier

Direction	Type
Input	String

Specifies a trusted block label or complete trusted block token, which will be updated by the service and returned in trusted_block_identifier. The length is indicated by input_block_length. Its content depends on the rule array keywords supplied to the service.

When rule_array is INACTIVE the block is complete but typically does not have MAC protection. If MAC protection is present due to recycling an existing trusted block, then the MAC key and MAC value will be overlaid by the new MAC key and MAC value. The input_block_identifier includes all fields of the trusted block token, but the MAC key and MAC will be filled in by the service. The Active flag will be set to False (0) in the block returned in trusted_block_identifier.

When the rule_array is ACTIVATE the block is complete, including the MAC protection which is validated during execution of the service. The Active flag must be False (0) on input. On output, the block will be returned in trusted_block_identifier provided the identifier is a token, with the Active flag changed to True (1), and the MAC value recalculated using the same MAC key. If the trusted_block_identifier is a label, the block will be written to the PKDS.

transport_key_identifier

Direction	Type
Input	String

Specifies a key label or key token for an IMP-PKA key that is used to protect the trusted block.

trusted_block_length

Direction	Type
Input/Output	Integer

Specifies the number of bytes of data in trusted_block_identifier parameter. The maximum length is 3500 bytes.

trusted_block_identifier

Direction	Type
Output	String

Specifies a trusted block label or trusted block token for the trusted block constructed by the service. On input, the trusted_block_length contains the size of this buffer. On output, the trusted_block_length is updated with the actual byte length of the trusted block written to the buffer if the trusted_block_identifier is a token. The trusted block consists of the data supplied in input_block_identifier, but with the MAC protection and Active flag updated according to the rule array keyword that is provided. See Table 189 on page 452 for details on the actions. If the trusted_block_identifier is a label identifying a key record in key storage, the returned trusted block token will be written to the PKDS.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 190. Required access control points for Trusted Block Create

Rule array keyword	Access control point
INACTIVE	Trusted Block Create - Create Block in inactive form
ACTIVATE	Trusted Block Create - Activate an inactive block

To prevent a weaker transport key (key-encrypting key) from being used to encipher a triple-length MAC key into an external trusted block, enable the **Trusted Block Create - Disallow triple-length MAC** access control.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 191. Trusted Block Create required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC).

Table 191. Trusted Block Create required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Unique Key Derive (CSNBUKD and CSNEUKD)

Unique Key Derive (CSNBUKD and CSNEUKD) can perform the key derivation process for DES-DUKPT as defined in ANSI X9.24 2007 Part 1 or the key derivation process for AES-DUKPT as defined in ANSI X9.24 2017 Part 3.

The process derives keys from two values: The base derivation key and the derivation data:

- The base derivation key is the key from which the others are derived. For DES-DUKPT, this must be a DES KEYGENKY key with the UKPT bit (bit 18) set to 1 in the Control Vector. For AES-DUKPT, this must be an AES DKYGENKY key with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1.
- The derivation data is used to make the derived key specific to a particular device and to a specific transaction from that device. The derivation data for DES-DUKPT, called the Current Key Serial Number (CKSN), is the 80-bit concatenation of the device's 59-bit Initial Key Serial Number value and the 21-bit value of the current encryption counter which the device increments for each new transaction. The derivation data for AES-DUKPT, as described in ANSI X9.24-3 (reference 1), is used to provide input parameters for the AES DUKPT key derivation function. The derivation data for AES-DUKPT is defined in Table 684 on page 1719.

The Initial Pin Encryption Key (IPEK) is derived from the base derivation key and the initial derivation data. Specify the K3IPEK rule array keyword to return the IPEK.

Rule array keywords determine the types and number of keys derived on a particular call. See the Rule Array parameter description for more information.

Output keys are wrapped using the mode configured as the default wrapping mode.

The DES key wrapping methods available are described in “CCA key wrapping” on page 20.

When the WRAPENH3 method is selected, a skeleton key token is required. A secure internal key token wrapped with the WRAPENH3 method obfuscates the key length.

Format

```
CALL CSNBUKD(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             base_derivation_key_identifier_length,
             base_derivation_key_identifier,
             derivation_data_length,
             derivation_data,
             generated_key_identifier1_length,
             generated_key_identifier1,
             generated_key_identifier2_length,
             generated_key_identifier2,
             generated_key_identifier3_length,
             generated_key_identifier3,
             transport_key_identifier_length,
             transport_key_identifier,
             reserved2_length,
             reserved2,
             reserved3_length,
             reserved3,
             reserved4_length,
             reserved4,
             reserved5_length,
             reserved5,
             reserved6_length,
             reserved6 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. Values are 1 through 7.

rule_array

Direction	Type
Input	String

The *rule_array* parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The *rule_array* keywords are:

Table 192. Keywords for Unique Key Derive

Keyword	Meaning
DUKPT algorithm (One, optional).	
DES	Specifies to derive keys using DES DUKPT algorithm as described in X9.24 2007 Part 1. This is the default.
A-DUKPT	Specifies to derive keys using AES DUKPT algorithm as described in X9.24 2017 Part 3.
Output key selection for generated_key_identifier1 (One, optional). Specify at least one output key selection keyword for **DES**. Not valid with the **K3IPEK**, **PIN-DATA**, or **A-DUKPT** keywords.	
K1DATA	The returned key type for this keyword is a DATA ENCRYPTION key. Output value *generated_key_identifier1* will be created and will be a data encryption key. The skeleton token provided in that parameter on input must be one of the permitted 'Data encryption key' types for this callable service. For valid values, see Table 194 on page 464.
Data encryption direction or initiation (One, optional, with K1DATA. Otherwise, not allowed).	
REQ-ENC	Specifies to derive a data encryption key to be used to send or process a request. See 'Data Encryption, request or both ways' under the DUKPT key usage description column in Table 195 on page 465.
RSP-ENC	Specifies to derive a data encryption key to be used to send or process a response. See 'Data Encryption, response' under the DUKPT key usage description column in Table 195 on page 465.
Output key selection for generated_key_identifier2 (One, optional). Specify at least one output key selection keyword for **DES**. Not valid with the **K3IPEK**, **PIN-DATA**, or **A-DUKPT** keywords.	
K2MAC	The returned key type for this keyword is a MAC key. Output value *generated_key_identifier2* will be created and will be a MAC key. The skeleton token provided in that parameter on input must be one of the permitted MAC key types for this callable service. For valid values, see Table 194 on page 464.
MAC direction or initiation (one, optional, with K2MAC. Otherwise, not allowed).	

Table 192. Keywords for Unique Key Derive (continued)	
Keyword	**Meaning**
REQ-MAC	Specifies to derive a MAC key to be used to send or process a request. See 'Message authentication, request or both ways' under the DUKPT key usage description column in Table 195 on page 465.
RSP-MAC	Specifies to derive a MAC key to be used to send or process a response. See 'Message authentication, response' under the DUKPT key usage description column in Table 195 on page 465.
Output key selection for generated_key_identifier3 (One, optional). Specify at least one output key selection keyword for **DES**. Not valid with the **K3IPEK** or **PIN-DATA**. No other output key selection keywords are allowed. For **A-DUKPT**, only **K3IPEK** is allowed.	
K3IPEK	The returned key for this keyword is the IPEK. When used in conjugation with the **DES** keyword, it specifies to use the *generated_key_identifier3* parameter to identify on input a null key token. On output, the initial PIN encryption key (IPEK) is returned TDES-wrapped using the key identified by the *transport_key_identifier* parameter. The IPEK is created by taking the base derivation key and encrypting the 59-bit initial key serial number contained within the derivation data. The key is returned either in an external CCA fixed-length DES key-token or an external TR-31 key block, depending on which token output type keyword is specified. When used in conjugation with the **A-DUKPT** keyword, it specifies to use the *generated_key_identifier3* parameter to identify on input a null key token. On output, the initial PIN encryption key (IPEK) is returned AES-wrapped using the key identified by the *transport_key_identifier* parameter. The IPEK is created by taking the information from derivation data. The key is returned in an external TR-31 key block. This keyword may not be combined with any other output key selection keyword.
K3PIN	The returned key type for this keyword is a PIN key. Output value *generated_key_identifier3* will be created and will be a PIN key. The skeleton token provided in that parameter on input must be one of the permitted PIN key types for this callable service. For valid values, see Table 194 on page 464. When the **A-DUKPT** keyword is specified, this keyword is not allowed.

Table 192. Keywords for Unique Key Derive (continued)	
Keyword	**Meaning**
PIN-DATA	The returned key type for this keyword is a PIN key, which is returned in a DATA key token. Output value *generated_key_identifier3* will be created and will be a DATA key. The skeleton token provided in that parameter on input must be one of the permitted 'PIN key with rule keyword PIN-DATA' key types for this callable service. For valid values, see Table 194 on page 464. To use this option: • Control Vector bit 61 (Not-CCA) will be set to a one. • Access Control Point Unique Key Derive - Allow PIN-DATA processing must be enabled. When the **A-DUKPT** keyword is specified, this keyword is not allowed.
Token output type (One, optional).	
TDES-TOK	Specifies that the output IPEK should be wrapped by the TDES transport key and returned in an external TDES token. Valid with the K3IPEK keyword only. When the **A-DUKPT** keyword is specified, this keyword is not allowed.
TR31-TOK	Specifies that the output IPEK should be wrapped by the TDES transport key if DES is specified or the AES transport key if A-DUKPT is specified and returned in a TR-31 key block. Valid with the K3IPEK keyword only.
Key wrapping method (One, optional). These keywords are only valid when CCA DES keys are derived.	
USECONFG	Specifies to wrap the key using the configuration setting for the default wrapping method. This is the default.
WRAP-ENH	Specifies to wrap the key using the enhanced wrapping method with SHA-1.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This is the default for triple-length keys.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.
WRAP-ECB	Specifies to wrap the key using the original wrapping method.

base_derivation_key_identifier_length

Direction	Type
Input	Integer

The length of the *base_derivation_key* parameter.

If the *base_derivation_key_identifier* field contains a label, the length must be 64 bytes.

Otherwise, the value must be between the actual length of the key token and 9992.

base_derivation_key_identifier

Direction	Type
Input/Output	String

The identifier of the base derivation key used to derive operational keys using the DUKPT algorithms defined in ANSI X9.24 2007 Part 1 and 3. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys:

- For DES, the identifier is a 64-byte key token of a DES key-derivation key of key type KEYGENKY with the UKPT attribute (bit 18) enabled in the control vector.
- For A-DUKPT, the identifier is a variable-length key token of an AES key-derivation key of key type DKYGENKY with the A-DUKPT key usage attribute enabled.

For TR-31 key, the identifier is a variable-length key block of a key-derivation key: key usage B0 and mode of use X. The algorithm is T for DES and A for A-DUKPT.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

derivation_data_length

Direction	Type
Input	Integer

The length of the *derivation_data* parameter. For DES, this value must be 10. For A-DUKPT, this value must be 20.

derivation_data

Direction	Type
Input	String

For DES, the derivation data is an 80-bit (10-byte) string that contains the Current Key Serial Number (CKSN) of the device concatenated with the 21-bit value of the current Encryption Counter which the device increments for each new transaction. For A-DUKPT, the derivation is a 20-byte string as defined in Table 684 on page 1719. For allowed derived working keys sizes, see Table 686 on page 1722.

generated_key_identifier1_length

Direction	Type
Input/Output	Integer

The length of the *generated_key_identifier1* parameter. The maximum length is 9992.

For DES, when K1DATA is specified:

- This value must be 64 when a fixed-length DES token or skeleton is provided.
- This value may be up to 9992 when a TR-31 key block or skeleton is provided.

For DES, when K1DATA is not specified, this value must be 0.

For A-DUKPT, on input, the length of the buffer for the *generated_key_identifier1* parameter in bytes. The maximum value is 9992. On output, the parameter will hold the actual length of the *generated_key_identifier1*. When A-DUKPT is specified with K3IPEK, this value must be 0.

generated_key_identifier1

Direction	Type
Input/Output	String

The buffer to receive the variable-length generated key.

For DES algorithm (DES keyword):

On input:

- For CCA keys, this must be a DES data-encryption 64-byte key token or a skeleton token of a DES data-encryption key, with one of the data-encryption control vectors as shown in Table 194 on page 464. To derive a compliant-tagged key token, a compliant-tagged skeleton token must be supplied.
- For TR-31 keys, this must be a skeleton key block with key usage D0, algorithm T, and mode of use B, D, or E as indicated by the Data encryption direction keyword. To generate a compliant-tagged key block, the IBM optional block "10" with the Compliance Tag attribute enabled must be included with the skeleton key block.

On output:
generated_key_identifier1 will contain the data encryption token or block with the derived data encryption key.

For AES algorithm (A-DUKPT keyword):

On input:

- For CCA keys, this must be a variable-length skeleton token for a DES, AES, or HMAC key. To derive a compliant-tagged key token, a compliant-tagged skeleton token must be supplied.
- For TR-31 keys, this must be a skeleton key block with key usage, algorithm, and mode of use matching the key defined in the derivation data structure.
  - When the key usage indicator in the derivation data is X'8000', only key usages B0 and B3 are allowed.
  - When the key usage indicator in the derivation data is X'8001', only key usage B1 is allowed.
  - To generate a compliant-tagged key block, the IBM optional block "10" with the Compliance Tag attribute enabled must be included with the skeleton key block.

On output:
generated_key_identifier1 will contain the key token as specified in the Derived Data structure. For the supported CCA key types for AES-DUKPT derived working keys, see Table 685 on page 1722.

generated_key_identifier2_length

Direction	Type
Input/Output	Integer

The length of the *generated_key_identifier2* parameter. The maximum length is 9992.

When the K2MAC keyword is specified:

- This value must be 64 when a fixed-length DES token or skeleton is provided.
- This value may be up to 9992 when a TR-31 key block or skeleton is provided.

Otherwise, this value must be 0.

generated_key_identifier2

Direction	Type
Input/Output	String

The buffer to receive the variable-length generated key.

On input:

- For CCA keys, this must be a DES MAC 64-byte key token or a skeleton token of a DES MAC key, with one of the MAC control vectors as shown in Table 194 on page 464. To derive a compliant-tagged key token, a compliant-tagged skeleton token must be supplied.
- For TR-31 keys, this must be a skeleton key block with key usage M0, M1, or M3, algorithm T, and mode of use C, G, or V as indicated by the MAC direction keyword. To generate a compliant-tagged key block, the IBM optional block "10" with the Compliance Tag attribute enabled must be included with the skeleton key block.

On output, *generated_key_identifier2* will contain the MAC token or block with the derived MAC key.

generated_key_identifier3_length

Direction	Type
Input/Output	Integer

The length of the *generated_key_identifier3* parameter. The maximum length is 9992.

When the rule array keywords K3IPEK or K3PIN are specified, the length must be at least 64 bytes.

When PINDATA is specified, the length must be 64.

Otherwise, the length must be 0.

generated_key_identifier3

Direction	Type
Input/Output	String

The input and output values for this parameter depends on the keyword specified in the *rule_array* parameter. The *rule_array* keyword for the *generated_key_identifier3* parameter can be PIN-DATA, K3PIN, or K3IPEK.

- When Rule Array Keyword is PIN-DATA, input must be a Data key token or skeleton token of a Data key with one of the 'PIN key with rule keyword PIN-DATA' control vectors as shown in Table 194 on page 464. On output, this parameter will contain the Data token with the derived PIN key.
- When Rule Array Keyword is K3PIN, input must be either:
  - A DES PIN key token or a skeleton token of a DES PIN key, with one of the PIN control vectors as shown in Table 194 on page 464. On output, this parameter will contain the PIN token with the derived PIN key.
  - A DES TR-31 key block or skeleton with key usage P0, algorithm T for DES, and mode of use D or E.
- When Rule Array Keyword is K3IPEK, input must be a null key token. Depending on the token output type keyword specified, the IPEK is either returned in an external CCA fixed-length DES key-token (TDES-TOK) or in an external non-CCA TR-31 key block (TR31-TOK). When TDES-TOK is specified, on output, the IPEK is returned in an external DES key-token wrapped by the TDES transport key. The control vector in the returned double-length DATA key-token is valued to binary zeros. When TR31-TOK is specified with the DES keyword, on output, the IPEK is returned in an external TR-31 key block wrapped by the TDES transport key. When TR-31-TOK is specified with the A-DUKPT keyword, on output, the IPEK is returned in an external TR-31 key block wrapped by the AES transport key. The key usage indicator in the AES-DUKPT derivation data must be set to X'8001' (Key Derivation Initial Key).
- To derive a compliant-tagged key token, a compliant-tagged skeleton token must be supplied.

Table 193. Contents of the TR-31 block header of the generated TR-31 key block and their meaning

TR-31 key block header field	Value in ASCII	Meaning
Key block version ID	B	Key block version and wrapping method. 'D' must be used for an AES IPEK.
	D	
Key usage	B1	IPEK key
Algorithm	T	TDES
	A	AES
Key mode of use	x	Key derivation key
Key version number	00	Unused
Key exportability (value depends on key-wrapping method, either specified in rule array or, if no keyword, by default configuration setting)	S if key-wrapping method is WRAP-ECB.	Sensitive: key exportable under any KEK
	E if key-wrapping method is any enhanced wrapping method.	Extra sensitive: key exportable under KEK meeting ANSI X9.24 Part 1 or Part 2 (no ECB mode encrypted KEKs allowed)
Number of optional blocks	00	No optional blocks

transport_key_identifier_length

Direction	Type
Input	Integer

The length of the *transport_key_identifier* parameter. If the transport key identifier is not used, the length must be 0.

When DES is specified with TDES-TOK or TR31-TOK, the length must be 64.

When A-DUKPT and TR31-TOK are specified, the length must the length of the AES transport key specified in *transport_key_identifier*. The maximum value is 725.

transport_key_identifier

Direction	Type
Input/Output	String

If the K3IPEK keyword is specified, the *transport_key_identifier* contains the label or key token for the key encrypting key to be used to wrap the IPEK.

For DES, the transport key must be a DES EXPORTER KEK.

For A-DUKPT, the transport key must be an AES EXPORTER KEK with key usage EXPTT31D and WR-AES. Otherwise, this field is ignored.

reserved2_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved2

Direction	Type
Ignored	String

This parameter is ignored.

reserved3_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved3

Direction	Type
Ignored	String

This parameter is ignored.

reserved4_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved4

Direction	Type
Ignored	String

This parameter is ignored.

reserved5_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved5

Direction	Type
Ignored	String

This parameter is ignored.

reserved6_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved6

Direction	Type
Ignored	String

This parameter is ignored.

Restrictions

The following table shows the valid skeleton tokens depending on the key type to be derived.

Table 194. Valid Control Vectors for Derived Keys

Key to be derived	Supported key types in the skeleton token		
Data encryption key	CIPHER	00 03 71 00 03 41 00 00	00 03 71 00 03 21 00 00
	ENCIPHER	00 03 60 00 03 41 00 00	00 03 60 00 03 21 00 00
	DECIPHER	00 03 50 00 03 41 00 00	00 03 50 00 03 21 00 00
Message authentication code (MAC) key	MAC	00 05 4D 00 03 41 00 00	00 05 4D 00 03 21 00 00
	MACVER	00 05 44 00 03 41 00 00	00 05 44 00 03 21 00 00
PIN key	IPINENC	00 21 5F 00 03 41 00 00	00 21 5F 00 03 21 00 00
	OPINENC	00 24 77 00 03 41 00 00	00 24 77 00 03 21 00 00
PIN key with rule keyword PIN-DATA	DATA PIN	00 00 7D 00 03 41 00 00	00 00 7D 00 03 21 00 00

Note that the following bits of the control vector are not checked and may have a value of either 0 or 1.

- Bit 17 - Export control.
- Bit 56 - Enhanced wrapping control.
- Bit 57 - TR-31 export control.
- Bits 4 and 5 - UDX.

Additional control vector bit that is not checked for PIN key with rule keyword PIN-DATA.

- Bit 61 - Not-CCA.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal key tokens that are stored in the CKDS.

The DUKPT key derivation process that is defined in the ANSI X9.24 standard describes the use of the derived keys in terms of a terminal, which sends requests, and a host, which processes those requests and sends responses.

Beginning with the July 2019 Licensed Internal Code (LIC), two direction or initiation rule-array keyword groups are added, one group for deriving MAC keys, and the other group for deriving data encryption keys. The use of these keywords is to specify the purpose of the key (MAC or data encryption) and whether the key is to be used to send or receive a request or to send or receive a response.

When a key is derived, it must be understood whether that key is used as a terminal-side key (term) or a host-side key (host). The key usage in the skeleton key token provided (for example, a MACVER key usage of MAC verify) determines the key usage for the derived key. In cases where DUKPT produces different key usages for the terminal and host keys, the correct usage must be chosen as shown in Table 195 on

page 465. The table also shows the key variant that is used in the derivation process for each DUKPT key usage.

Table 195. DES-DUKPT key variants for derived keys

DUKPT key usage description	DUKPT derivation variant	Direction or initiation keyword	CCA key type
PIN Encryption	00000000000000FF 00000000000000FF	N/A	IPINENC OPINENC
Message authentication, request or both ways	000000000000FF00 000000000000FF00	No direction keyword	MAC MACGEN
		REQ-MAC	MAC
Message authentication, response	00000000FF000000 00000000FF000000	No direction keyword	MACVER
		RSP-MAC (term)	MACVER
		RSP-MAC (host)	MACGEN
Data encryption, request or both ways	0000000000FF0000 0000000000FF0000	No direction keyword	CIPHER ENCIPHER
		REQ-ENC	CIPHER
Data encryption, response	000000FF00000000 000000FF00000000	No direction keyword	DECIPHER
		RSP-ENC (term)	DECIPHER
		RSP-ENC (host)	ENCIPHER

Note: A default DES MAC key has usage of generate and verify. The Key Token Build service can be used to build a skeleton DES MAC key that has usage of generate only (MACGEN). Call the service by specifying keywords INTERNAL, DES, DOUBLE or DOUBLE-O, and CV, and use this 16-byte value for the *control_vector* variable: X'0005480003410000000 5480003210000'.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the generated key.

Access control points

The Unique Key Derive access control point controls the function of this service. Specifying a 'Key wrapping method' in the rule array requires the **Unique Key Derive - Override default wrapping** access control point to be enabled in the active role.

Specifying the PIN-DATA rule array keyword requires the **Unique Key Derive - Allow PIN-DATA** access control point to be enabled in the active role.

Specifying the K3IPEK rule array keyword requires the **Unique Key Derive - K3IPEK** access control point to be enabled in the active role.

To disallow the import of a key wrapped with a weaker transport key, the **Symmetric Key Import2 - disallow weak import** access control must be enabled.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 196. Unique Key Derive required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys are not supported. Compliant-tagged key tokens are not supported. A-DUKPT, REQ-ENC, RSP-ENC, REQ-MAC, and RSP-MAC keywords are not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys are not supported. Compliant-tagged key tokens are not supported. REQ-ENC, RSP-ENC, REQ-MAC, and RSP-MAC keywords require a CEX5C or later with the July 2019 or later licensed internal code (LIC). A-DUKPT keyword is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys are not supported. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The A-DUKPT keyword requires a CEX6C or later with the October 2020 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 196. Unique Key Derive required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. A-DUKPT keyword is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	A-DUKPT keyword requires a CEX6C or later with the September 2020 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	A-DUKPT keyword requires a CEX6C or later with the September 2020 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Chapter 6. Protecting data

Use ICSF to protect sensitive data stored on your system, sent between systems, or stored off your system on magnetic tape. To protect data, encipher it under a key. When you want to read the data, decipher it from ciphertext to plaintext form.

ICSF provides *encipher* and *decipher callable services* to perform these functions. If you use a key to encipher data, you must use the same key to decipher the data. To use clear keys directly, ICSF provides *symmetric key decipher*, *symmetric key encipher*, *encode* and *decode callable services*. These services encipher and decipher with clear keys. You can use clear keys indirectly by first using the clear key import callable service, and then using the encipher and decipher callable services.

This topic describes these services:

- "Cipher Text Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3)" on page 471
- "Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)" on page 484
- "Decode (CSNBDCO and CSNEDCO)" on page 490
- "Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)" on page 492
- "Encode (CSNBECO and CSNEECO)" on page 499
- "Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)" on page 501
- "Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)" on page 509
- "Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)" on page 521
- "Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)" on page 531

Modes of operation

To encipher or decipher data or keys, ICSF uses either the U.S. National Institute of Standards and Technology (NIST) Data Encryption Standard (DES) algorithm or the Advanced Encryption Standard (AES) algorithm. The DES algorithm is documented in *Federal Information Processing Standard #46*. The AES algorithm is documented in Federal Information Processing Standard 197.

ICSF enciphers and deciphers using several modes of operation. Some of the modes have variations related to padding or blocking of the data. The text in parentheses is the processing rule associated with that mode.

The supported modes are:

- Electronic code book (ECB)
- Cipher block chaining (CBC)
  - Cipher block chaining with ciphertext stealing (CBC-CS)
  - Cipher block chaining compatible with CUSP/PCF (CUSP)
  - Cipher block chaining compatible with IPS (IPS)
  - Cipher block chaining using PKCS#7 padding (PKCS-PAD)
  - Cipher block chaining using ANSI X9.23 padding (X9.23)
  - Cipher block chaining using IBM 4700 padding (4700-PAD)
- Cipher Feedback (CFB)
  - Cipher Feedback with a non-blocksize segment (CFB-LCFB)
- Output Feedback (OFB)
- Galois/Counter Mode (GCM)

Electronic Code Book (ECB) Mode

In the ECB mode, each block of plaintext is separately enciphered and each block of the ciphertext is separately deciphered. In other words, the encipherment or decipherment of a block is totally independent of other blocks. ICSF uses the ECB encipherment mode for enciphering and deciphering data with clear keys using the encode and decode callable services.

Cipher Block Chaining (CBC) Mode

The CBC mode uses an initial chaining vector (ICV) in its processing. The CBC mode only processes blocks of data in exact multiples of the blocksize. The ICV is exclusive ORed with the first block of plaintext prior to the encryption step; the block of ciphertext just produced is exclusive-ORed with the next block of plaintext, and so on. You must use the same ICV to decipher the data. This disguises any pattern that may exist in the plaintext. CBC mode is the default for encrypting and decrypting data using the Encipher and Decipher callable services. "Cipher processing rules" on page 1643 describes the CBC-specific processing rules in detail.

Cipher Feedback (CFB) Mode

The CFB mode uses an initial chaining vector (ICV) in its processing. CFB mode performs cipher feedback encryption. CFB mode operates on segments instead of blocks. The segment length (called *s*) is between one bit and the block size (called *b*) for the underlying algorithm (DES or AES), inclusive. ICSF only allows segment sizes which are a multiple of eight bits (complete bytes). Each encryption step takes an input block, enciphers it with the key provided to generate an output block, takes the most significant *s* bits of the output block, and then exclusive ORs that with the plaintext segment. The first input block is the ICV and each subsequent input block is formed by concatenating the (*b*-*s*) least significant bits of the previous input block and the ciphertext (*s* bits) from the previous step to form a full block. The input text can be of any length. The output text will have the same length as the input text.

Output Feedback (OFB) Mode

The OFB mode uses an initial chaining vector (ICV) in its processing. OFB mode requires that the ICV is a nonce (the ICV must be unique for each execution of the mode under the given key). Each encryption step takes an input block, enciphers it with the key provided to generate an output block, and then exclusive ORs the output block with the plaintext block. The first input block is the ICV and each subsequent input block is the previous output block. The input text can be of any length. The output text will have the same length as the input text.

Galois/Counter Mode (GCM)

The GCM mode uses an initialization vector (IV) in its processing. This mode is used for authenticated encryption with associated data. GCM provides confidentiality and authenticity for the encrypted data and authenticity for the additional authenticated data (AAD). The AAD is not encrypted. GCM mode requires that the IV is a nonce, i.e., the IV must be unique for each execution of the mode under the given key. The steps for GCM encryption are:

1. The hash subkey for the GHASH function is generated by applying the block cipher to the "zero" block.
2. The pre-counter block (J_0) is generated from the IV. In particular, when the length of the IV is 96 bits, then the padding string $0^{31}||1$ is appended to the IV to form the pre-counter block. Otherwise, the IV is padded with the minimum number of '0' bits, possibly none, so that the length of the resulting string is a multiple of 128 bits (the block size); this string in turn is appended with 64 additional '0' bits, followed by the 64-bit representation of the length of the IV, and the GHASH function is applied to the resulting string to form the pre-counter block.
3. The 32-bit incrementing function is applied to the pre-counter block to produce the initial counter block for an invocation of the GCTR function on the plaintext. **The output of this invocation of the GCTR function is the ciphertext.**
4. The AAD and the ciphertext are each appended with the minimum number of '0' bits, possibly none, so that the bit lengths of the resulting strings are multiples of the block size. The concatenation of these

strings is appended with the 64-bit representations of the lengths of the AAD and the ciphertext to produce block *u*.

5. The GHASH function is applied to block *u* to produce a single output block.
6. This output block is encrypted using the GCTR function with the pre-counter block that was generated in Step "2" on page 470, and **the result is truncated to the specified tag length to form the authentication tag**.
7. The ciphertext and the tag are returned as the output.

   The plaintext can be of any length. The ciphertext will have the same length as the plaintext.

For GCM decryption, the tag is an input parameter. ICSF calculates a tag using the same process as encryption and compares that to the parameter passed by the caller. If they match, the decryption will proceed.

Triple DES Encryption

Triple-DES encryption uses a triple-length key comprised of three 8-byte DES keys to encipher 8 bytes of data using this method:

- Encipher the data using the first key
- Decipher the result using the second key
- Encipher the second result using the third key

The procedure is reversed to decipher data that has been triple-DES enciphered:

- Decipher the data using the third key
- Encipher the result using the second key
- Decipher the second result using the first key

ICSF uses the triple-DES encryption in the CBC encipherment mode.

A variation of the triple DES algorithm supports the use of a double-length data-encryption key comprised of two 8-byte DATA keys. In this method, the first 8-byte key is reused in the last encipherment step.

Due to export regulations, triple-DES encryption may not be available on your processor.

Cipher Text Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3)

This callable service deciphers encrypted data (ciphertext) under one cipher text translation key and reenciphers it under another cipher text translation key without having the data appear in the clear outside the cryptographic coprocessor. ICSF uses the ciphertext translation key as either the input or the output data transport key. Such a function is useful in a multiple node network, where sensitive data is passed through multiple nodes prior to it reaching its final destination.

"Using the Cipher Text Translate2 callable service" on page 76 provides some tips on using the callable service.

Use the Cipher Text Translate2 callable service to decipher text under an "input" key and then to encipher the text under an "output" key. Both AES and DES algorithms are supported. Translation between AES and DES is allowed with restrictions controlled by access control points.

The encryption modes supported are:

- DES – CBC, CUSP. and IPS
- AES – CBC and ECB

The padding methods supported are:

- DES – X9.23
- AES – PKCSPAD

Choosing between CSNBCTT2 and CSNBCTT3

CSNBCTT2 and CSNBCTT3 provide identical functions. When choosing the service to use, consider this:

- **CSNBCTT2** requires the input text and output text to reside in the caller's primary address space. Also, a program using CSNBCTT2 adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface. The callable service name for AMODE(64) invocation is CSNECTT2.
- **CSNBCTT3** allows the input text and output text to reside either in the caller's primary address space or in a data space. This allows you to translate more data with one call. However, a program using CSNBCTT3 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface. The callable service name for AMODE(64) invocation is CSNECTT3. For CSNBCTT3 and CSNECTT3, *text_id_in* and *text_id_out* are access list entry token (ALET) parameters of the data spaces containing the input text and output text.

Format

```
CALL CSNBCTT2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_in_length,
              key_identifier_in,
              initialization_vector_in_length,
              initialization_vector_in,
              cipher_text_in_length,
              cipher_text_in,
              chaining_vector_length,
              chaining_vector,
              key_identifier_out_length,
              key_identifier_out,
              initialization_vector_out_length,
              initialization_vector_out,
              cipher_text_out_length,
              cipher_text_out,
              reserved1_length,
              reserved1,
              reserved2_length,
              reserved2 )
```

```
CALL CSNBCTT3(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_in_length,
              key_identifier_in,
              initialization_vector_in_length,
              initialization_vector_in,
              cipher_text_in_length,
              cipher_text_in,
              chaining_vector_length,
              chaining_vector,
              key_identifier_out_length,
              key_identifier_out,
              initialization_vector_out_length,
              initialization_vector_out,
              cipher_text_out_length,
              cipher_text_out,
              reserved1_length,
              reserved1,
              reserved2_length,
              reserved2,
              text_id_in,
              text_id_out )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 4 or 5.

rule_array

Direction	Type
Input	String

The keywords that provide control information to the callable service. The following table provides a list. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 197. Keywords for Cipher Text Translate2

Keyword	Meaning
Inbound Processing Rule (One required)	
I-CBC	Specifies encryption using CBC mode for the inbound ciphertext. The text length must be a multiple of the block size. The DES block size is 8 bytes. The AES block size is 16 bytes.

Table 197. Keywords for Cipher Text Translate2 (continued)	
Keyword	**Meaning**
I-CUSP	Specifies that CBC with CUSP processing for the inbound ciphertext. The ciphertext may be any length. The ciphertext is the same length as the plaintext. This keyword is only valid with DES.
I-ECB	Specifies encryption using ECB mode for the inbound ciphertext. The text must be a multiple of the block size. This keyword is only valid for AES encryption.
I-IPS	Specifies that CBC with IPS processing has been used for the inbound ciphertext. The ciphertext may be any length. The ciphertext is the same length as the plaintext. This keyword is only valid with DES.
IPKCSPAD	Specifies that CBC with PKCS padding was used for the inbound ciphertext. The text was padded on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the AES block size, before the data was enciphered. Each pad character is valued to the number of pad characters added. This keyword is only valid for AES encryption.
I-X923	Specifies that CBC with X9.24 padding was used for the inbound ciphertext. This is compatible with the requirements in ANSI Standard X9.23. This keyword is only valid for DES encryption.
Outbound Processing Rule (One required)	
O-CBC	Specifies encryption in CBC mode will be used for the outbound ciphertext. The text length must be a multiple of the block size. The DES block size is 8 bytes. The AES block size is 16 bytes.
O-CUSP	Specifies that CBC with CUSP processing will be used for the outbound text. The outbound ciphertext will be the same length as the plaintext. This keyword is only valid with DES.
O-ECB	Specifies encryption using ECB mode will be used for the outbound ciphertext. The text must be a multiple of the block size. This keyword is only valid for AES encryption.
O-IPS	Specifies that CBC with IPS processing will be used for the outbound text. The outbound ciphertext will be the same length as the plaintext. This keyword is only valid with DES.
OPKCSPAD	Specifies that CBC with PKCS padding will be used for the outbound text. The outbound text will be padded on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the AES block size, before the data was enciphered. Each pad character is valued to the number of pad characters added. This keyword is only valid for AES encryption.
O-X923	Specifies that CBC with X9.24 padding will be used for the outbound text. This is compatible with the requirements in ANSI Standard X9.23. This keyword option is only valid for DES encryption.
Segmenting Control (One optional)	

Table 197. Keywords for Cipher Text Translate2 (continued)	
Keyword	**Meaning**
CONTINUE	Specifies the initialization vectors are taken from the chaining vector. The chaining vector will be updated and must not be modified between calls. This keyword is ignored for I-ECB and O-ECB processing rules. The CONTINUE keyword is not valid with the I-X923 or O-X923 keywords.
INITIAL	Specifies the initialization vectors will be taken from the *initialization_vector_in* and *initialization_vector_out* parameters. This is the default. This keyword is ignored for I-ECB and O-ECB processing rules.
Inbound Key Identifier (One Required)	
IKEY-DES	Specifies that the inbound key identifier is a DES key.
IKEY-AES	Specifies that the inbound key identifier is an AES key.
Outbound Key Identifier (One Required)	
OKEY-DES	Specifies that the outbound key identifier is a DES key.
OKEY-AES	Specifies that the outbound key identifer is an AES key.

key_identifier_in_length

Direction	Type
Input	Integer

Length of the *key_identifier_in* field in bytes.

The value is 64 when a label is supplied.

Otherwise, the value must be between the actual length of the key token and 9992.

key_identifier_in

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to decipher the inbound ciphertext. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys:

- Acceptable DES key types are DATA, CIPHER, CIPHERXI, CIPHERXL, and DECIPHER. The keys must have bit 19 for 'DECIPHER' set on in the control vector. The key may be a single-length, double-length, or triple-length key. If the **Cipher Text Translate2 - Allow only cipher text translate types** access control point is enabled, only CIPHERXI and CIPHERXL are allowed.
- Acceptable AES key types include the 64-byte AES DATA key and the variable length token CIPHER key with the DECRYPT bit on in the key usage field. The C-XLATE bit can optionally be on. If the **Cipher Text Translate2 - Allow only cipher text translate types** access control point is enabled, the C-XLATE bit must be turned on in the key usage field.

For TR-31 keys, key blocks must contain a data-encrypting key with usage D0 or D3, algorithm A, D, or T, and mode of use B or D. When the **Cipher Text Translate2 - Allow only cipher text translate types** access control is enabled, the key must have key usage D3.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

initialization_vector_in_length

Direction	Type
Input	Integer

Length of the *initialization_vector_in* field in bytes. For AES keys, the length is 16. For DES keys, the length is 8. When the initialization vector is not required (segmenting rule CONTINUE, processing rule I-ECB), the value must be 0.

initialization_vector_in

Direction	Type
Input	String

The initialization vector that is used to decipher the input data. This parameter is the initialization vector used at the previous cryptographic node. This parameter is required for segmenting rule INITIAL.

ciphertext_in_length

Direction	Type
Input	Integer

The length of the ciphertext to be processed. See the table of ciphertext length restrictions in the Usage Notes.

ciphertext_in

Direction	Type
Input	String

The text that is to be translated. The text is enciphered under the cipher key specified in the *key_identifier_in* parameter.

chaining_vector_length

Direction	Type
Input	Integer

The length of the *chaining_vector* parameter in bytes. The *chaining_vector* field must be 128 bytes long.

chaining_vector

Direction	Type
Input/Output	String

The *chaining_vector* parameter is a work area used by the service to carry segmented data between procedure calls. This area must not be modified between calls to the service.

key_identifier_out_length

Direction	Type
Input	Integer

Length of the *key_identifier_out* field in bytes.

This value is 64 when a label is supplied.

Otherwise, the value must be between the actual length of the key token and 9992.

key_identifier_out

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to encipher the outbound ciphertext. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys:

- Acceptable DES key types are DATA, CIPHER, CIPHERXL, CIPHERXO, and ENCIPHER. The keys must have bit 18 for 'ENCIPHER' set on in the control vector. The key may be a double- or triple-length key. If the **Cipher Text Translate2 - Allow only cipher text translate types** access control point is enabled, only CIPHERXO and CIPHERXL are allowed.
- Acceptable AES key types include the 64-byte AES DATA key and the variable length token CIPHER key with the ENCRYPT bit on in the key usage field. The C-XLATE bit can optionally be on. If the **Cipher Text Translate2 – Allow only cipher text translate types** access control point is enabled, the C-XLATE bit must be turned on in the key usage field.

For TR-31 keys, key blocks must contain a data-encrypting key with usage D0 or D3, algorithm A, D, or T, and mode of use B or E. When the **Cipher Text Translate2 - Allow only cipher text translate types** access control point is enabled, the key must have key usage D3.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

initialization_vector_out_length

Direction	Type
Input	Integer

Length of the *initialization_vector_out* field in bytes. For AES keys, the length is 16. For DES keys, the length is 8. When the initialization vector is not required (segmenting rule CONTINUE, processing rule O-ECB), the value must be 0.

initialization_vector_out

Direction	Type
Input	String

The initialization vector that is used to encipher the input data. This is the new initialization vector used when the callable service enciphers the plaintext. This parameter is required for segmenting rule INITIAL.

ciphertext_out_length

Direction	Type
Input/Output	Integer

Length of the *ciphertext_out* in bytes. This parameter will updated with the actual length of the data in the *ciphertext_out* parameter. Note that padding may require this value to be larger than the *ciphertext_in_length* parameter. See the table of ciphertext length restrictions in the Usage Notes.

ciphertext_out

Direction	Type
Output	String

The field where the callable service returns the translated text.

reserved1_length

Direction	Type
Input	Integer

The length of the *reserved1* parameter in bytes. The value must be zero.

reserved1

Direction	Type
Input	String

This parameter is ignored.

reserved2_length

Direction	Type
Input	Integer

The length of the *reserved2* parameter in bytes. The value must be zero.

reserved2

Direction	Type
Input	String

This parameter is ignored.

text_id_in

Direction	Type
Input	Integer

For CSNBCTT3 only, the ALET of the *ciphertext_in* parameter.

text_id_out

Direction	Type
Input	Integer

For CSNBCTT3 only, the ALET of the ciphertext_out parameter.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The initialization vectors must have already been established between the communicating applications or must be passed with the data.

The following table outlines the restrictions for the *ciphertext_in_length* and *ciphertext_out_length* parameters. The DES blocks referred to in this table are 8 bytes. The AES blocks referred to in this table are 16 bytes.

Table 198. Restrictions for ciphertext_in_length and ciphertext_out_length

Input cipher method	Output cipher method	Input ciphertext length restriction[s]	Output ciphertxt length restriction[s]
DES CBC	DES CBC X9.23	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the sum of the length of the input ciphertext and a DES block.
DES CBC	AES CBC PKCSPAD	Input cipher text must be a multiple of a DES block.	If the input ciphertext is NOT a multiple of an AES block, then the output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES block. If the input ciphertext is a multiple of an AES block, then the output ciphertext length must be greater than or equal to the sum of the input ciphertext length and an AES block.
DES CBC	DES CUSP or IPS	Input cipher text must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC	DES CBC	Input cipher text must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC	AES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC	AES ECB	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC CUSP or IPS	DES CBC CUSP or IPS	No restrictions	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC CUSP or IPS	DES CBC	Input cipher text must be a multiple of a DES block.	Output ciphertext length must be be greater than or equal to the input ciphertext length.
DES CBC CUSP or IPS	AES CBC or ECB	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be be greater than or equal to the input ciphertext length.
DES CBC CUSP or IPS	DES CBC X9.23	No restrictions	Output ciphertext length must be be greater than or equal to the sum of the input ciphertext length and a DES block.
DES CBC CUSP or IPS	AES CBC PKCSPAD	No restrictions	Output ciphertext length must be be greater than or equal to the sum of the input ciphertext length and a AES block.

Table 198. Restrictions for ciphertext_in_length and ciphertext_out_length (continued)

Input cipher method	Output cipher method	Input ciphertext length restriction[s]	Output ciphertxt length restriction[s]
DES CBC X9.23	DES CBC X9.23	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC X9.23	AES CBC PKCSPAD	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES bock.
DES CBC X9.23	DES CBC CUSP or IPS	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC X9.23	DES CBC	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length. **Note:** This operation will not be possible if the padding is determined by the adapter to be from 1-7 bytes.
DES CBC X9.23	AES CBC	Input ciphertext must be a multiple of a DES block but must not be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length. **Note:** This operation will not be possible if the padding is determined by the adapter to be from 1-7 bytes.
DES CBC X9.23	AES ECB	Input ciphertext must be a multiple of a DES block but must not be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length. **Note:** This operation will not be possible if the padding is determined by the adapter to be from 1-7 bytes.
AES CBC or ECB	DES CBC X9.23	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES bock.
AES CBC or ECB	AES CBC PKCSPAD	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and an AES bock.
AES CBC or ECB	DES CBC CUSP or IPS	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC or ECB	DES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.

Table 198. Restrictions for ciphertext_in_length and ciphertext_out_length (continued)

Input cipher method	Output cipher method	Input ciphertext length restriction[s]	Output ciphertxt length restriction[s]
AES CBC or ECB	AES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC or ECB	AES ECB	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC PKCSPAD	DES CBC X9.23	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC PKCSPAD	AES CBC PKCSPAD	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC PKCSPAD	DES CBC CUSP or IPS	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length minus 1.
AES CBC PKCSPAD	DES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length minus the length of a DES block. **Note:** This operation will not be possible if the padding is determined by the adapter to be from 1-7 bytes or 9-15 bytes.
AES CBC PKCSPAD	AES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length minus the length of a AES block. **Note:** This operation will not be possible if the padding is determined by the adapter to be from 1-15 bytes.
AES CBC PKCSPAD	AES ECB	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length minus the length of a AES block. **Note:** This operation will not be possible if the padding is determined by the adapter to be from 1-15 bytes.

There are requirements for the keys for the *key_identifier_in* and *key_identifier_out* parameters. The *key_identifier_in* key must be able to decipher text. The *key_identifier_out* key must be able to encipher text.

The following table shows the valid key types which are allowed for the *key_identifier_in* and *key_identifier_out* parameters. In the table, a variable length key token cipher key is denoted by vCIPHER. vCIPHER is the default which has the ENCRYPT and DECRYPT bits on in the usage field. vCIPHERe has

only the ENCRYPT bit on in the usage field. vCIPHERd has only the DECRYPT bit on in the usage field. Adding x to either of the preceding names means the TRANSLAT bit is on in the usage field for that key. (For example, vCIPHERex means a variable length token with the ENCRYPT and TRANSLAT bits turned on.)

AESDATA is the 64-byte AES DATA key type.

Table 199. Cipher Text Translate2 key usage

key_identifier_in (DEC bit except DATA and AESDATA)	**key_identifier_out (ENC bit except DATA and AESDATA)**
CIPHER **CIPHERXI** **CIPHERXL** **DATA** **DECIPHER**	AESDATA CIPHER CIPHERe CIPHERedx CIPHERex CIPHERXL CIPHERXO DATA ENCIPHER
AESDATA **CIPHER** **CIPHERd** **CIPHERdex** **CIPHERdx**	AESDATA CIPHER (requires ACP to be enabled) CIPHERe CIPHERedx CIPHERex CIPHERXL (requires ACP to be enabled) CIPHERXO (requires ACP to be enabled) DATA (must be at least double-length key with ACP) ENCIPHER (requires ACP to be enabled)

Note:

1. Translation from stronger encryption to single-key DES is not allowed.
2. Translation from a triple-length DES key to a double-length DES key requires the **Cipher Text Translate2 – Allow translate to weaker DES** access control point to be enabled.
3. When the **Cipher Text Translate2 – Allow only cipher text translate key types** access control point is enabled, only CIPHERXI, CIPHERXL, and CIPHERXO DES key types are allowed and the C-XLATE key usage bit must be on for AES CIPHER keys.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 200. Cipher Text Translate2 access control points

Access control point	**Description**
Cipher Text Translate2	Enable Cipher Text Translate2 service
Cipher Text Translate2 – Allow translate from AES to TDES	Allow translation from an AES key to 2 or 3 key triple DES key.
Cipher Text Translate2 – Allow translate to weaker AES	Allow translation from a stronger to weaker AES key. (For example, IN key AES256 and OUT key AES128.)

Table 200. Cipher Text Translate2 access control points (continued)	
Access control point	**Description**
Cipher Text Translate2 – Allow translate to weaker DES	Allow translation from a stronger to weaker DES key. The only supported translation is from 3-key TDES to 2-key TDES.
Cipher Text Translate2 – Allow only cipher text translate types	When enabled, the *key_identifiers* parameters must be a key with key type CIPHERXI, CIPHERXL, or CIPHERXO for DES and key type CIPHER with the C-XLATE key usage bit on for AES.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 201. Cipher Text Translate2 required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.

Table 201. Cipher Text Translate2 required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)

Use the decipher callable service to decipher data in an address space or a data space using the cipher block chaining mode. ICSF supports these processing rules to decipher data. You choose the type of processing rule that the decipher callable service should use for block chaining.

Processing Rule
Purpose

ANSI X9.23
For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, but the plaintext will be 1 to 8 bytes shorter than the ciphertext. The *text_length* will also be reduced to show the original length of the plaintext.

CBC
For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, and the plaintext will have the same length.

CUSP
For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.

IBM 4700
For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, but the plaintext will be 1 to 8 bytes shorter than the ciphertext. The *text_length* will also be reduced to show the original length of the plaintext.

IPS
For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.

The cipher block chaining (CBC) mode uses an initial chaining value (ICV) in its processing. The first 8 bytes of ciphertext is deciphered and then the ICV is exclusive ORed with the resulting 8 bytes of data to form the first 8-byte block of plaintext. Thereafter, the 8-byte block of ciphertext is deciphered and exclusive ORed with the previous 8-byte block of ciphertext until all the ciphertext is deciphered.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the *key_identifier* parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

A different ICV may be passed on each call to the decipher callable service. However, the same ICV that was used in the corresponding encipher callable service must be passed.

Short blocks are text lengths of 1 to 7 bytes. A short block can be the only block. Trailing short blocks are blocks of 1 to 7 bytes that follow an exact multiple of 8 bytes. For example, if the text length is 21, there are two 8-byte blocks and a trailing short block of 5 bytes. Because DES processes text only in exact multiples of 8 bytes, some special processing is required to decipher such short blocks. Short blocks and trailing short blocks of 1 to 7 bytes of data are processed according to the Cryptographic Unit

Support Program (CUSP) rules, or by the record chaining scheme devised by and used in the Information Protection System (IPS) in the IPS/CMS product.

These methods of treating short blocks and trailing short blocks do not increase the length of the ciphertext over the plaintext. If the plaintext was *padded* during encipherment, the length of the ciphertext will always be an exact multiple of 8 bytes.

ICSF supports these padding schemes:

- ANSI X9.23
- 4700-PAD

Choosing between CSNBDEC and CSNBDEC1

CSNBDEC and CSNBDEC1 provide identical functions. When choosing which service to use, consider this:

- **CSNBDEC** requires the ciphertext and plaintext to reside in the caller's primary address space. Also, a program using CSNBDEC adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

  The callable service name for AMODE(64) invocation is CSNEDEC.
- **CSNBDEC1** allows the ciphertext and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBDEC1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

  The callable service name for AMODE(64) invocation is CSNEDEC1.

  For CSNBDEC1 and CSNEDEC1, *cipher_text_id* and *clear_text_id* are access list entry token (ALET) parameters of the data spaces containing the ciphertext and plaintext.

Format

```
CALL CSNBDEC(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_identifier,
             text_length,
             cipher_text,
             initialization_vector,
             rule_array_count,
             rule_array,
             chaining_vector,
             clear_text )
```

```
CALL CSNBDEC1(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_identifier,
             text_length,
             cipher_text,
             initialization_vector,
             rule_array_count,
             rule_array,
             chaining_vector,
             clear_text,
             cipher_text_id,
             clear_text_id )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to decipher the data. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte DES key token of key type DATA, CIPHER, or DECIPHER.

For X9.143 keys, the identifier is a variable-length key block of a DES or TDES data-encrypting key: key usage D0, algorithm D or T, and mode of use B or D.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

text_length

Direction	Type
Input/Output	Integer

On entry, you supply the length of the ciphertext. The maximum length of text is 2147483647 bytes. A zero value for the *text_length* parameter is not valid. If the returned deciphered text (*clear_text*

parameter) is a different length because of the removal of padding bytes, the value is updated to the length of the plaintext.

Note: The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced.

The application program passes the length of the ciphertext to the callable service. The callable service returns the length of the plaintext to your application program.

cipher_text

Direction	Type
Input	String

The text to be deciphered.

initialization_vector

Direction	Type
Input	String

The 8-byte supplied string for the cipher block chaining. The first block of the ciphertext is deciphered and exclusive ORed with the initial chaining vector (ICV) to get the first block of cleartext. The input block is the next ICV. To decipher the data, you must use the same ICV used when you enciphered the data.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supply in the *rule_array* parameter. The value must be 1, 2, or 3.

rule_array

Direction	Type
Input	Character String

An array of 8-byte keywords providing the processing control information. The array is positional. See the keywords in Table 202 on page 487. The first keyword in the array is the processing rule. You choose the processing rule you want the callable service to use for deciphering the data. The second keyword is the ICV selection keyword. The third keyword (or the second if the ICV selection keyword is allowed to default) is the encryption algorithm to use.

Table 202. Keywords for the Decipher Rule Array Control Information

Keyword	Meaning
Processing Rule (required)	
Rules CUSP, IPS, X9.23, and 4700-PAD should be specified only when there is one request or on the last request of a sequence of chained requests	

Table 202. Keywords for the Decipher Rule Array Control Information (continued)	
Keyword	**Meaning**
CBC	Performs cipher block chaining as described in NIST SP 800-38A. The data must be a multiple of 8 bytes. An OCV is produced and placed in the *chaining_vector* parameter. If the ICV selection keyword CONTINUE is specified, the CBC OCV from the previous call is used as the ICV for this call.
CUSP	Performs deciphering that is compatible with IBM's CUSP and PCF products. The data can be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The CUSP/PCF OCV is placed in the *chaining_vector* parameter. If the ICV selection keyword CONTINUE is specified, the CUSP/PCF OCV from the previous call is used as the ICV for this call.
IPS	Performs deciphering that is compatible with IBM's IPS product. The data can be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The IPS OCV is placed in the *chaining_vector* parameter. If the ICV selection keyword CONTINUE is specified, the IPS OCV from the previous call is used as the ICV for this call.
X9.23	Deciphers with cipher block chaining and text length reduced to the original value. This is compatible with the requirements in ANSI standard X9.23. The ciphertext length must be an exact multiple of 8 bytes. Padding is removed from the plaintext.
4700-PAD	Deciphers with cipher block chaining and text length reduced to the original value. The ciphertext length must be an exact multiple of 8 bytes. Padding is removed from the plaintext.
ICV Selection (optional)	
CONTINUE	This specifies taking the initialization vector from the output chaining vector (OCV) contained in the work area to which the *chaining_vector* parameter points. CONTINUE is valid only for processing rules CBC, IPS, and CUSP.
INITIAL	This specifies taking the initialization vector from the *initialization_vector* parameter. INITIAL is the default value.
Encryption Algorithm (optional)	
DES	This specifies using the data encryption standard and ignoring the token marking.
TOKEN	This specifies using the data encryption algorithm in the DATA key token. This is the default.

"Cipher processing rules" on page 1643 describes the cipher processing rules in detail.

chaining_vector

Direction	Type
Input/Output	String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector holds the output chaining vector (OCV) from the caller. The OCV is the first 8 bytes in the 18-byte string.

The direction is output if the ICV selection keyword of the *rule_array* parameter is INITIAL. The direction is input/output if the ICV selection keyword of the *rule_array* parameter is CONTINUE.

clear_text

Direction	Type
Input/Output	String

The field where the callable service returns the deciphered text.

cipher_text_id

Direction	Type
Input	Integer

For CSNBDEC1/CSNEDEC1 only, the ALET of the ciphertext to be deciphered.

clear_text_id

Direction	Type
Input	Integer

For CSNBDEC1/CSNEDEC1 only, the ALET of the clear text supplied by the application.

Restrictions

The service will fail under these conditions:

- If the key token contains double or triple-length keys and triple-DES is not enabled.
- If a token is marked CDMF.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You **cannot** destructively overlap the plaintext and ciphertext fields. For example:

```
pppppp
     cccccc  is supported.

cccccc
     pppppp  is not supported.

ppppppcccccc is supported.

P represents the plaintext and c represents the ciphertext.
```

“Cipher processing rules” on page 1643 discusses the cipher processing rules.

The Encipher callable services are described under “Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)” on page 492.

Access control point

The **Decipher - DES** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 203. Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Decode (CSNBDCO and CSNEDCO)

Note: This service has been deprecated. New applications should use the Symmetric Key Decipher service that is described in "Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)" on page 521 instead of this service.

Use this callable service to decipher an 8-byte string using a clear key. The callable service uses the electronic code book (ECB) mode of the DES.

The callable service name for AMODE(64) invocation is CSNEDCO.

Considerations

If you have only a clear key, you are *not* limited to using only the encode and decode callable services. You can pass your clear key to the Multiple Clear Key Import service and get back a secure key token that will allow you to use the Encipher and Decipher services.

Format

```
CALL CSNBDCO(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              clear_key,
              cipher_text,
              clear_text)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

clear_key

Direction	Type
Input	String

The 8-byte clear key value that is used to decode the data.

cipher_text

Direction	Type
Input	String

The ciphertext that is to be decoded. Specify 8 bytes of text.

clear_text

Direction	Type
Output	String

The 8-byte field where the plaintext is returned by the callable service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 204. Decode required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions	
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions	
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions	
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	

Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)

Use the encipher callable service to encipher data in an address space or a data space using the cipher block chaining mode. ICSF supports these processing rules to encipher data. You choose the type of processing rule that the encipher callable service should use for the block chaining.

Processing Rule
Purpose

ANSI X9.23
For block chaining not necessarily in exact multiples of 8 bytes. This process rule pads the plaintext so that ciphertext produced is an exact multiple of 8 bytes.

CBC
For block chaining in exact multiples of 8 bytes.

CUSP
For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.

IBM 4700
For block chaining not necessarily in exact multiples of 8 bytes. This process rule pads the plaintext so that the ciphertext produced is an exact multiple of 8 bytes.

IPS
For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.

For more information about the processing rules, see Table 205 on page 496 and "Cipher processing rules" on page 1643.

The cipher block chaining (CBC) mode of operation uses an initial chaining vector (ICV) in its processing. The ICV is exclusive ORed with the first 8 bytes of plaintext prior to the encryption step, and thereafter, the 8-byte block of ciphertext just produced is exclusive ORed with the next 8-byte block of plaintext, and so on. This disguises any pattern that may exist in the plaintext.

The selection between single-DES encryption mode and triple-DES encryption mode is controlled by the length of the key supplied in the *key_identifier* parameter. If a single-length key is supplied, single-DES encryption is performed. If a double-length or triple-length key is supplied, triple-DES encryption is performed.

To nullify the CBC effect on the first 8-byte block, supply 8 bytes of zero. However, the ICV may require zeros.

Cipher block chaining also produces a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in *record chaining*.

Note that the OCV that results is the same, whether an encipher or a decipher callable service was invoked, assuming the same text, ICV, and key were used.

Short blocks are text lengths of 1 to 7 bytes. A short block can be the only block. Trailing short blocks are blocks of 1 to 7 bytes that follow an exact multiple of 8 bytes. For example, if the text length is 21, there are two 8-byte blocks, and a trailing short block of 5 bytes. Short blocks and trailing short blocks of 1 to 7 bytes of data are processed according to the Cryptographic Unit Support Program (CUSP) rules, or by the record chaining scheme devised by and used by the Information Protection System (IPS) in the IPS/CMS program product. These methods of treating short blocks and trailing short blocks do not increase the length of the ciphertext over the plaintext.

An alternative method is to pad the plaintext and produce a ciphertext that is longer than the plaintext. The plaintext can be padded with up to 8 bytes using one of several padding schemes. This padding produces a ciphertext that is an exact multiple of 8 bytes long.

If the cleartext is already a multiple of 8, the ciphertext can be created using any processing rule.

Because of padding, the returned ciphertext length is longer than the provided plaintext; the *text_length* parameter *will have been modified*. The returned ciphertext field should be 8 bytes longer than the length of the plaintext to accommodate the maximum amount of padding. You should provide this extension in your installation's storage because ICSF cannot detect whether the extension was done.

The minimum length of data that can be enciphered is one byte.

Attention: If you lose the data-encrypting key under which the data (plaintext) is enciphered, the data enciphered under that key (ciphertext) **cannot** be recovered.

Choosing between CSNBENC and CSNBENC1

CSNBENC and CSNBENC1 provide identical functions. When choosing which service to use, consider this:

- **CSNBENC** requires the cleartext and ciphertext to reside in the caller's primary address space. Also, a program using CSNBENC adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

  The callable service name for AMODE(64) invocation is CSNEENC.
- **CSNBENC1** allows the cleartext and ciphertext to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using

CSNBENC1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

The callable service name for AMODE(64) invocation is CSNEENC1.

For CSNBENC1 and CSNEENC1, *clear_text_id* and *cipher_text_id* are access list entry token (ALET) parameters of the data spaces containing the cleartext and ciphertext.

Format

```
CALL CSNBENC(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_identifier,
             text_length,
             clear_text,
             initialization_vector,
             rule_array_count,
             rule_array,
             pad_character,
             chaining_vector,
             cipher_text )
```

```
CALL CSNBENC1(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              key_identifier,
              text_length,
              clear_text,
              initialization_vector,
              rule_array_count,
              rule_array,
              pad_character,
              chaining_vector,
              cipher_text,
              clear_text_id,
              cipher_text_id )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to encipher the data. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte DES key token of key type DATA, CIPHER, or ENCIPHER.

For X9.143 keys, the identifier is a variable-length key block of a DES or TDES data-encrypting key: key usage D0, algorithm D or T, and mode of use B or E.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

text_length

Direction	Type
Input/Output	Integer

On entry, the length of the plaintext (*clear_text* parameter) you supply. The maximum length of text is 2147483647 bytes. A zero value for the *text_length* parameter is not valid. If the returned enciphered text (*cipher_text* parameter) is a different length because of the addition of padding bytes, the value is updated to the length of the ciphertext.

Note: The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647).

The application program passes the length of the plaintext to the callable service. The callable service returns the length of the ciphertext to the application program.

clear_text

Direction	Type
Input	String

The text that is to be enciphered.

initialization_vector

Direction	Type
Input	String

The 8-byte supplied string for the cipher block chaining. The first 8 bytes (or less) block of the data is exclusive ORed with the ICV and then enciphered. The input block is enciphered and the next ICV is created. You must use the same ICV to decipher the data.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supply in the *rule_array* parameter. The value must be 1, 2, or 3.

rule_array

Direction	Type
Input	Character String

An array of 8-byte keywords providing the processing control information. The array is positional. See the keywords in Table 205 on page 496. The first keyword in the array is the processing rule. You choose the processing rule you want the callable service to use for enciphering the data. The second keyword is the ICV selection keyword. The third keyword (or the second if the ICV selection keyword is allowed to default to INITIAL) is the encryption algorithm to use.

Table 205. Keywords for the Encipher Rule Array Control Information

Keyword	Meaning
Processing Rule (required) Rules CUSP, IPS, X9.23, and 4700-PAD should be specified only when there is one request or on the last request of a sequence of chained requests.	
CBC	Performs cipher block chaining as described in NIST SP 800-38A. The data must be a multiple of 8 bytes. An OCV is produced and placed in the *chaining_vector* parameter. If the ICV selection keyword CONTINUE is specified, the CBC OCV from the previous call is used as the ICV for this call.
CUSP	Performs ciphering that is compatible with IBM's CUSP and PCF products. The data can be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The CUSP/PCF OCV is placed in the *chaining_vector* parameter. If the ICV selection keyword CONTINUE is specified, the CUSP/PCF OCV from the previous call is used as the ICV for this call.
IPS	Performs ciphering that is compatible with IBM's IPS product. The data may be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The IPS OCV is placed in the *chaining_vector* parameter. If the ICV selection keyword CONTINUE is specified, the IPS OCV from the previous call is used as the ICV for this call.
X9.23	Performs cipher block chaining with 1 to 8 bytes of padding. This is compatible with the requirements in ANSI standard X9.23. If the data is not in exact multiples of 8 bytes, X9.23 pads the plaintext so that the ciphertext produced is an exact multiple of 8 bytes. The plaintext is padded to the next multiple 8 bytes, even if this adds 8 bytes. An OCV is produced.

Table 205. Keywords for the Encipher Rule Array Control Information (continued)	
Keyword	**Meaning**
4700-PAD	Performs padding by extending the user's plaintext with the caller's specified pad character, followed by a one-byte binary count field that contains the total number of bytes added to the message. 4700-PAD pads the plaintext so that the ciphertext produced is an exact multiple of 8 bytes. An OCV is produced.
ICV Selection (optional)	
CONTINUE	This specifies taking the initialization vector from the output chaining vector (OCV) contained in the work area to which the *chaining_vector* parameter points. CONTINUE is valid only for processing rules CBC, IPS, and CUSP.
INITIAL	This specifies taking the initialization vector from the *initialization_vector* parameter. INITIAL is the default value.
Encryption Algorithm (optional)	
DES	This specifies using the data encryption standard and ignoring the token marking.
TOKEN	This specifies using the data encryption algorithm in the DATA key token. TOKEN is the default.

These recommendations help the caller determine which encipher processing rule to use:

- If you are exchanging enciphered data with a specific implementation, for example, CUSP or ANSI X9.23, use that processing rule.
- If the ciphertext length must be equal to the plaintext length and the plaintext length cannot be a multiple of 8 bytes, use either the IPS or CUSP processing rule.

"Cipher processing rules" on page 1643 describes the cipher processing rules in detail.

pad_character

Direction	**Type**
Input	Integer

An integer, 0 to 255, that is used as a padding character for the 4700-PAD process rule (*rule_array* parameter).

chaining_vector

Direction	**Type**
Input/Output	String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector holds the output chaining vector (OCV) from the caller. The OCV is the first 8 bytes in the 18-byte string.

The direction is output if the ICV selection keyword of the *rule_array* parameter is INITIAL.

The direction is input/output if the ICV selection keyword of the *rule_array* parameter is CONTINUE.

cipher_text

Direction	Type
Output	String

The enciphered text the callable service returns. The length of the ciphertext is returned in the *text_length* parameter. The *cipher_text* may be 8 bytes longer than the length of the *clear_text* field because of the padding that is required for some processing rules.

clear_text_id

Direction	Type
Input	Integer

For CSNBENC1/CSNEENC1 only, the ALET of the clear text to be enciphered.

cipher_text_id

Direction	Type
Input	Integer

For CSNBENC1/CSNEENC1 only, the ALET of the ciphertext that the application supplied.

Restrictions

The service will fail under these conditions:

- If the key token contains double- or triple-length keys and triple-DES is not enabled.
- If a token is marked CDMF.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You **cannot** destructively overlap the plaintext and ciphertext fields. For example:

```
cccccc
     pppppp  is supported.
cccccc
     pppppp  is not supported.

ppppppcccccc is supported.

P represents the plaintext and c represents the ciphertext.
```

The method used to produce the OCV is the same with the CBC, 4700-PAD, and X9.23 processing rules. However, that method is different from the method used by the CUSP and IPS processing rules.

“Cipher processing rules” on page 1643 discusses the cipher processing rules.

The Decipher callable services are described under “Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)” on page 484.

Access control point

The **Encipher - DES** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 206. Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Encode (CSNBECO and CSNEECO)

Note: This service has been deprecated. New applications should use the Symmetric Key Encipher service that is described in "Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)" on page 531 instead of this service.

Use the encode callable service to encipher an 8-byte string using a clear key. The callable service uses the electronic code book (ECB) mode of the DES.

The callable service name for AMODE(64) invocation is CSNEECO.

Considerations

If you have only a clear key, you are *not* limited to using just the encode and decode callable services. You can pass your clear key to the Multiple Clear Key Import service and get back a secure key token that will allow you to use the Encipher and Decipher services.

Format

```
CALL CSNBECO(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             clear_key,
             clear_text,
             cipher_text)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

clear_key

Direction	Type
Input	String

The 8-byte clear key value that is used to encode the data.

clear_text

Direction	Type
Input	String

The plaintext that is to be encoded. Specify 8 bytes of text.

cipher_text

Direction	Type
Output	String

The 8-byte field where the ciphertext is returned by the callable service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 207. Encode required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions	
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions	
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions	
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	

Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)

The symmetric algorithm decipher callable service deciphers data with the AES algorithm. Encryption modes supported are Cipher Block Chaining (CBC) mode, Electronic Code Book (ECB) mode, and Galois/Counter Mode (GCM).

You can specify that the clear text data was padded before encryption using the method described in the PKCS standards. In this case, the callable service will remove the padding bytes and return the unpadded clear text data. PKCS padding is described in "PKCS padding method" on page 1646.

The callable service names for AMODE(64) invocation are CSNESAD and CSNESAD1.

Choosing between CSNBSAD and CSNBSAD1 or CSNESAD and CSNESAD1

CSNBSAD, CSNBSAD1, CSNESAD, and CSNESAD1 provide identical functions. When choosing which service to use, consider this:

- CSNBSAD and CSNESAD require the cipher text and plaintext to reside in the caller's primary address space. Also, a program using CSNBSAD adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSAD1 and CSNESAD1 allow the cipher text and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSAD1 and CSNESAD1 does not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSAD1 and CSNESAD1, *cipher_text_id* and *clear_text_id* are access list entry token (ALET) parameters of the data spaces containing the cipher text and plaintext.

Format

```
CALL CSNBSAD(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             key_parms_length,
             key_parms,
             block_size,
             initialization_vector_length,
             initialization_vector,
             chain_data_length,
             chain_data,
             cipher_text_length,
             cipher_text,
             clear_text_length,
             clear_text,
             optional_data_length,
             optional_data)
```

```
CALL CSNBSAD1(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_length,
             key_identifier,
             key_parms_length,
             key_parms,
             block_size,
             initialization_vector_length,
             initialization_vector,
             chain_data_length,
             chain_data,
             cipher_text_length,
             cipher_text,
             clear_text_length,
             clear_text,
             optional_data_length,
             optional_data,
             cipher_text_id,
             clear_text_id)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 2, 3 or 4.

rule_array

Direction	Type
Input	String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

Table 208. Symmetric Algorithm Decipher Rule Array Keywords

Keyword	Meaning
Algorithm (required, one keyword)	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes.
Processing Rule (optional, one keyword)	
CBC	Performs encryption in cipher block chaining (CBC) mode. The text length must be a multiple of the AES block size (16-bytes). This is the default value.
ECB	Performs encryption in electronic code book (ECB) mode. The text length must be a multiple of the AES block size (16-bytes).

Table 208. Symmetric Algorithm Decipher Rule Array Keywords (continued)	
Keyword	**Meaning**
GCM	Performs Galois/Counter mode decryption. The plaintext will have the same length as the ciphertext. Additionally, the authentication tag will be verified before the data is returned.
PKCS-PAD	Performs encryption in cipher block chaining (CBC) mode. The ciphertext length must be an exact multiple of 16 bytes. Padding is removed from the plaintext and the text length is reduced to the original value. This rule should be specified only when there is one request or on the last request of a sequence of chained requests.
X9.23PAD	Specifies that the cleartext was padded according to the PKCS padding scheme. Performs encryption in cipher block chaining (CBC) mode. The ciphertext length must be an exact multiple of 16 bytes. Padding is removed from the plaintext and the text length is reduced to the original value. This rule should be specified only when there is one request or on the last request of a sequence of chained requests.
Key Rule (required, one keyword)	
KEYIDENT	This indicates that the value in the *key_identifier* parameter is either an internal key token or the label of a key token in the CKDS. The key must be a secure AES key, that is, enciphered under the current master key.
ICV Selection (optional for CBC and PKCS-PAD, required for GCM, one keyword)	
INITIAL	This specifies that this is the first request of a sequence of chained requests and indicates that the initialization vector should be taken from the *initialization_vector* parameter. This is the default value for CBC and PKDS-PAD. This keyword is not valid with processing rule GCM.
CONTINUE	This specifies that this request is part of a sequence of chained requests, and is not the first request in that sequence. The initialization vector will be taken from the work area identified in the *chain_data* parameter. This keyword is only valid for processing rules CBC or PKCS-PAD.
ONLY	Specifies that this is the only request and indicates that the initialization vector should be taken from the *initialization_vector* parameter. Only valid with the processing rule GCM.

key_identifier_length

Direction	**Type**
Input	Integer

The length of the *key_identifier* parameter in bytes.

When the *key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input	String

The identifier of the data-encrypting key to decrypt the text. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA keys, the identifier is a variable-length AES key token of a data-encrypting key.

- 64-byte DATA key token (version X'04') .
- Variable-length CIPHER key token (version X'05'). The key usage must indicate DECRYPT and the appropriate mode of encryption (CBC, ECB, GCM, or ANY-MODE).

For TR-31 key blocks, the identifier is a variable-length key block of an AES data-encrypting key: key usage D0, algorithm A, and mode of use B or D.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

key_parms_length

Direction	Type
Input	Integer

The length of the *key_parms* parameter in bytes.

For the GCM processing rule, this is the length of the authentication tag to be verified. Valid lengths are 4, 8, 12, 13, 14, 15, and 16, but using a length of 4 or 8 is strongly discouraged.

For all other processing rules, the value must be zero.

key_parms

Direction	Type
Ignored	String

The *key_parms* parameter contains key related parameters.

For the GCM processing rule, *key_parms* will contain an authentication tag to be verified for the provided ciphertext (*cipher_text* parameter) and additional authenticated data (*optional_data* parameter). You must specify the same *key_parms* generated when the text was enciphered.

Otherwise, this parameter is ignored.

block_size

Direction	Type
Input	Integer

The block size for the cryptographic algorithm. AES requires the block size to be 16.

initialization_vector_length

Direction	Type
Input	Integer

The length of the *initialization_vector* parameter in bytes. For CBC, PKCS-PAD, and X9.23PAD, the length must be equal to the block length for the algorithm specified, 16. For the GCM processing rule, NIST recommends a length of 12, but tolerates any non-zero length up to a maximum of 2^{32}-1.

This parameter is ignored when the process rule is ECB.

initialization_vector

Direction	Type
Input	String

This parameter contains the initialization vector (IV) for CBC mode decryption. This includes CBC, GCM, PKCS-PAD, and X9.23PAD processing rule keywords. The IV must be the same value used when the data was encrypted.

This parameter is ignored when the process rule is ECB.

chain_data_length

Direction	Type
Input/Output	Integer

The length of the *chain_data* parameter in bytes. On input, it contains the length of the buffer provided with parameter *chain_data*. On output, it is updated with the length of the data returned in the *chain_data* parameter.

For CBC, the value must be at least 32. For ECB and GCM, the parameter is ignored.

chain_data

Direction	Type
Input/Output	String

A buffer that is used as a work area for sequences of chained symmetric algorithm decipher requests. The exact content and layout of *chain_data* is not described. Your application program must not change the data in this string.

When the keyword INITIAL is used, this is an output parameter and receives data that is needed when deciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output parameter; the value received as output from the previous call in the sequence is provided as input to this call, and in turn, this call will return new *chain_data* that will be used as input on the next call. When CONTINUE is used, both the data (*chain_data* parameter) and the length (*chain_data_length* parameter) must be the same values that were received in these parameters as output on the preceding call to the service in the chained sequence.

For ECB and GCM, this parameter is ignored.

cipher_text_length

Direction	Type
Input	Integer

The length of the cipher text. For processing rules CBC, ECB, PKCS-PAD, and X9.23PAD, the length must be a multiple of the algorithm block size. The maximum length is 2^{32}-1.

For GCM, the value may be zero.

When the Crypto Express adapter is a CEX5 or CEX6, the maximum value is 2^{29}-1. When the Crypto Express adapter is a CEX7 or CEX8, the maximum value is 2^{32}-1.

cipher_text

Direction	Type
Input	String

The text to be deciphered.

clear_text_length

Direction	Type
Input/Output	Integer

On input, this parameter specifies the size of the storage pointed to by the *clear_text* parameter, which must be at least the same size as the *cipher_text_length*. On output, this parameter has the actual length of the text stored in the *clear_text* parameter.

If process rules PKCS-PAD or X9.23PAD are used, the *clear_text_length* returned will be less than the *cipher_text_length* since padding bytes are removed.

clear_text

Direction	Type
Output	String

The deciphered text the service returns.

optional_data_length

Direction	Type
Input	Integer

The length of the *optional_data* parameter in bytes. For the GCM processing rule, this parameter contains the length of the Additional Authenticated Data (AAD). The value may be 0 to 2^{32}-1.

For all other processing rules, the value must be 0.

optional_data

Direction	Type
Ignored	String

Optional data required by a specified algorithm or processing mode. For the GCM processing rule, this parameter contains the Additional Authenticated Data (AAD). For all other processing rules, this field is ignored.

You must specify the same *optional_data* used when the text was enciphered.

cipher_text_id

Direction	Type
Input	Integer

For CSNBSAD1 and CSNESAD1 only, the ALET of the dataspace in which the *cipher_text* parameter resides.

clear_text_id

Direction	Type
Input	Integer

For CSNBSAD1 and CSNESAD1 only, the ALET of the dataspace in which the *clear_text* parameter resides.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The *clear_text* and *cipher_text* parameters may be in any dataspace. The *initialization_vector* and *optional_data* parameters must be in the caller's address space (primary).

Access control point

The **Symmetric Algorithm Decipher - secure AES keys** access control point controls the function of this service. Use of the GCM processing rule requires that the **Symmetric Algorithm Decipher – Galois/ Counter mode AES** access control is enabled.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 209. Symmetric Algorithm Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords GCM and ONLY require the March 2016 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keyword X9.23PAD is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Keywords GCM and ONLY require the March 2016 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keyword X9.23PAD is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Rule array keyword X9.23PAD requires the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 209. Symmetric Algorithm Decipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keyword X9.23PAD is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keyword X9.23PAD requires the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keyword X9.23PAD requires the CCA release 7.4 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)

The Symmetric Algorithm Encipher callable service enciphers data with the AES algorithm. Encryption modes supported are Cipher Block Chaining (CBC) mode, Electronic Code Book (ECB) mode, and Galois/Counter Mode (GCM).

The Symmetric Algorithm Encipher service supports the Australian Payment Network (APN) standards to generate and verify MACs and related processing as defined in AS2805.5.4.

To generate a MAC – Processing rule A28MACGN

Parameters:

- *key_identifier*: Double-length DES MAC key.
- *key_parms*: Double-length DES CIPHE R key.
- *clear_text*: Clear message text.
- *chain_data*:
  - Input: Starting MAC residue, encrypted by the key in the *key_parms* parameter.
  - Output: Final MAC residue, encrypted by the key in the *key_parms* parameter.
- *cipher_text*:
  - Output: 8-byte MAC.

To verify a MAC – Processing rule A28MACVR

Parameters:

- *key_identifier*: Double-length DES MAC key.
- *key_parms*: Double-length DES CIPHE R key.
- *clear_text*: Clear message text.

- *chain_data*:
  - Input: Starting MAC residue as indicated by the Residue value keyword, encrypted by the key in the *key_parms* parameter.
  - Output: Final MAC residue, encrypted by the key in the *key_parms* parameter.
- *cipher_text*:
  - Input: 8-byte MAC to verify.

One Way Function processing

Processing rule A28OWFEC
Parameters:

- *key_identifier*: Double-length DES EXPORTER key.
- *key_parms*: Double-length DES CIPHE R key.
- *chain_data*: Input value ECB wrapped by the key specified in the *key_parms* field.
- *cipher_text*: Output of the OWF.

Processing rule A28OWFCL
Parameters:

- *key_identifier*: Double-length DES CIPHER key.
- *chain_data*: Clear input value.
- *cipher_text*: Output of the OWF.

The callable service names for AMODE(64) invocation are CSNESAE and CSNESAE1

Choosing between CSNBSAE and CSNBSAE1 or CSNESAE and CSNESAE1

CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1 provide identical functions. When choosing which service to use, consider this:

- CSNBSAE and CSNESAE require the cipher text and plaintext to reside in the caller's primary address space. Also, a program using CSNBSAE adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSAE1 and CSNESAE1 allow the cipher text and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using CSNBSAE1 and CSNESAE1 does not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSAE1 and CSNESAE1, *cipher_text_id* and *clear_text_id* are access list entry token (ALET) parameters of the data spaces containing the cipher text and plaintext.

Format

```
CALL CSNBSAE(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              key_parms_length,
              key_parms,
              block_size,
              initialization_vector_length,
              initialization_vector,
              chain_data_length,
              chain_data,
              clear_text_length,
              clear_text,
```

```
              cipher_text_length,
              cipher_text,
              optional_data_length,
              optional_data)
```

```
CALL CSNBSAE1(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              key_parms_length,
              key_parms,
              block_size,
              initialization_vector_length,
              initialization_vector,
              chain_data_length,
              chain_data,
              clear_text_length,
              clear_text,
              cipher_text_length,
              cipher_text,
              optional_data_length,
              optional_data,
              clear_text_id,
              cipher_text_id)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 2, 3, 4, or 5.

rule_array

Direction	Type
Input	String

This keyword provides control information to the callable service. The keywords must be eight bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 210. Symmetric Algorithm Encipher Rule Array Keywords

Keyword	Meaning
Algorithm (required, one keyword)	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm will be used. The block size is 16-bytes, and the key length may be 16-, 24-, or 32-bytes (128-, 192-, 256-bits).
DES	Specifies the use of the Data Encryption Standard (DES) as the encryption algorithm. Only valid with processing rule A28MACGN, A28MACVR, A28OWFEC, and A28OWFCL keywords.
Processing Rule (one, required when the DES algorithm keyword is specified; optional when the AES keyword is specified).	
Rules for the DES algorithm (one required).	
A28MACGN	Specifies to generate a MAC as defined by the AusPayNet standard AS2805.5.4.
A28MACVR	Specifies to verify a MAC as defined by the AusPayNet standard AS2805.5.4. A Residue Value keyword must be specified.
A28OWFCL	Specifies to process the value in the *chain_data* parameter as clear data and return the results of the One Way Function as specified in AS2805.5.4 in the *ciphertext* parameter.
A28OWFEC	Specifies to process the value in the *chain_data* parameter as ECB encrypted using the key supplied in the *key_parms* parameter. The value will be decrypted and the results of the One Way Function as specified in AS2805.5.4 will be returned in the *ciphertext* parameter.
Rules for the AES algorithm (one optional).	
CBC	Performs encryption in cipher block chaining (CBC) mode. The text length must be a multiple of the AES block size (16-bytes). This is the default value.
ECB	Performs encryption in electronic code book (ECB) mode. The text length must be a multiple of the AES block size (16-bytes).

Table 210. Symmetric Algorithm Encipher Rule Array Keywords (continued)	
Keyword	**Meaning**
GCM	Perform Galois/Counter mode encryption. The plaintext may be any length. The ciphertext will have the same length as the plaintext. The *key_parms_length* and *key_parms* parameters are used to indicate the length of the tag (the value *t*) on input and contains the tag on output. Additional Authenticated Data (AAD) is contained in the *optional_data_length* and *optional_data* parameters.
PKCS-PAD	Performs encryption in cipher block chaining (CBC) mode, but the data is padded using PKCS padding rules. The length of the clear text data does not have to be a multiple of the cipher block length. The cipher text will be longer than the clear text by at least one byte, and up to 16-bytes. The PKCS padding method is described in "PKCS padding method" on page 1646. This rule should be specified only when there is one request or on the last request of a sequence of chained requests.
X9.23PAD	Performs encryption in cipher block chaining (CBC) mode, but the data is padded using the X9.23 padding scheme. The length of the clear text data does not have to be a multiple of the cipher block length. The cipher text will be longer than the clear text by at least one byte, and up to 16-bytes. This rule should be specified only when there is one request or on the last request of a sequence of chained requests.
Residue Value (one, required with A28MACVR). Only valid with processing rules A28MACVR.	
A28RES	Specifies that there is a residue value in the first 8 bytes of the *chain_data* parameter. These 8 bytes will be overwritten in the return. See the *chain_data* parameter for details.
A28NORES	Specifies that there is no residue value in the first 8 bytes of the *chain_data* parameter. These 8 bytes should be left null, as they will be overwritten in the return. See the *chain_data* parameter for details. Only valid with A28MACVR.
Key Rule (required, one keyword)	
KEYIDENT	This indicates that the value in the *key_identifier* parameter is either an internal key token or the label of a key token in the CKDS. The key must be a secure AES key, that is, enciphered under the current master key.
ICV Selection (one, required for GCM; otherwise, optional).	
INITIAL	This specifies that this is the first request of a sequence of chained requests and indicates that the initialization vector should be taken from the *initialization_vector* parameter. This is the default value for CBC and PKDS-PAD. This keyword is not valid with processing rule GCM.
CONTINUE	This specifies that this request is part of a sequence of chained requests, and is not the first request in that sequence. The initialization vector will be taken from the work area identified in the *chain_data* parameter. This keyword is only valid for processing rules CBC or PKCS-PAD. This keyword is not valid with the ECB or GCM processing rule keyword.

Table 210. Symmetric Algorithm Encipher Rule Array Keywords (continued)	
Keyword	**Meaning**
ONLY	Specifies that this is the only request and indicates that the initialization vector should be taken from the *initialization_vector* parameter. Only valid with the processing rule GCM, A28MACGN, A28MACVR, A28OWFEC, and A28OWFCL. This is the default for A28MACGN, A28MACVR, A28OWFEC, and A28OWFCL.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter in bytes.

When the *key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The identifier of the key to encrypt the text. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For processing rule keywords CBC, ECB, GCM, PKCS-PAD, and X9.23PAD:

- For CCA keys:
  - The identifier is a 64-byte AES key token of key type DATA (version X'04').
  - The identifier is a variable-length AES key token of key type CIPHER (version X'05'). The key usage must indicate ENCRYPT and the appropriate mode of encryption (CBC, ECB, GCM, or ANY-MODE).
- For X9.143 keys: the identifier is a variable-length key block of an AES data-encrypting key: key usage is D0, algorithm A, and mode of use B or E.

For processing rule keyword A28MACGN, the key is a MAC generation key:

- For CCA keys, the identifier is a 64-byte DES key token of key type MAC, and the ANY-MAC key usage attribute must be enabled in the control vector. The key must be a double-length key.
- For X9.143 keys, the identifier is a variable-length key block of a TDES or double-length DES message authentication key. For TDES the key block must have key usage is M3, algorithm T, mode of use G or C, and must be a triple-length key. For double-length DES, the key block must have key usage D0, algorithm T, mode of use B, and must be a double-length key.

For processing rule keyword A28MACVR, the key is a MAC verification key:

- For CCA keys, the identifier is a 64-byte DES key token of key type MAC, and the ANY-MAC key usage attribute must be enabled in the control vector. The key must be a double-length key.
- For X9.143 keys, the identifier is a variable-length key block of a TDES or double-length DES message authentication key. For TDES the key block must have key usage is M3, algorithm T, mode of use V or C, and must be a triple-length key. For double-length DES, the key block must have key usage D0, algorithm T, mode of use B, and must be a double-length key.

For processing rule keywords A28OWFEC, the key is a key-encrypting key:

- For CCA keys, the identifier is a 64-byte DES key token of key type EXPORTER. The key must be a double-length key.

- For X9.143 keys, the identifier is a variable-length key block of an TDES or double-length DES key-encrypting key. For TDES the key block must have key usage is K0, algorithm T, mode of use E, and must be a triple-length key. For double-length DES, the key block must have key usage D0, algorithm T, mode of use D or B, and must be a double-length key.

For processing rule keywords A28OWFCL, the key is a double-length data-encryption key:

- For CCA keys, the identifier is a 64-byte DES key token of key type CIPHER and the key usages must permit decryption. The key must be a double-length key.
- For X9.143 keys: the identifier is a variable-length key block of a DES data-encrypting key: key usage is D0, algorithm T, mode of use B or D, and must be a double-length key.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

key_parms_length

Direction	Type
Input	Integer

The length of the *key_parms* parameter in bytes.

For the GCM processing rule, this is the length of the authentication tag to be verified. Valid lengths are 4, 8, 12, 13, 14, 15, and 16, but using a length of 4 or 8 is strongly discouraged. If there is an error in processing, this value will be set to zero on output. Otherwise, it will be unchanged.

For processing rule keywords A28MACGN, A28MACVR, and A28OWFEC:

- When the *key_identifier* contains a label, the value must be 64.
- Otherwise, the value must be between the actual length of the token and 9992.

For all other processing rules, the value must be zero.

key_parms

Direction	Type
Input/Output	String

The *key_parms* parameter contains key related parameters.

For the GCM processing rule, *key_parms* will contain the generated authentication tag for the provided plaintext (*plain_text* parameter) and additional authenticated data (*optional_data* parameter). You must specify this generated *key_parms* when deciphering the text.

For processing rule keywords A28MACGN, A28MACVR, and A28OWFEC, this is the key used to encrypt the *chain_data* parameter. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

- For CCA keys, the identifier is a 64-byte DES key token of key type CIPHER. The key must be a double-length key.
- For TR-31 keys, the identifier is a variable-length key block of a TDES data-encrypting key: key usage is D0, algorithm T, and mode of use B or D. The key must be a double-length key.

When the value of the *key_parms_length* parameter is 0, this parameter is ignored.

block_size

Direction	Type
Input	Integer

The block size for the cryptographic algorithm. The block size for AES is 16. The block size for DES is 8.

initialization_vector_length

Direction	Type
Input	Integer

The length of the *initialization_vector* parameter in bytes. For CBC, PKCS-PAD, or X9.23PAD, the length must be equal to the block length for the algorithm specified, 16. For the GCM processing rule, NIST recommends a length of 12, but tolerates any non-zero length up to a maximum of 2^{32}-1.

This parameter is ignored when the process rule is ECB, A28MACGN, A28MACVR, A28OWFEC, and A28OWFCL.

initialization_vector

Direction	Type
Input	String

This parameter contains the initialization vector (IV) for CBC mode decryption. This includes CBC, GCM, PKCS-PAD, and X9.23PAD processing rule keywords. The same IV value must be used when the data is decrypted.

This parameter is ignored when the process rule is ECB, A28MACGN, A28MACVR, A28OWFEC, and A28OWFCL.

chain_data_length

Direction	Type
Input/Output	Integer

The length of the *chain_data* parameter in bytes. On input, it contains the length of the buffer provided with parameter *chain_data*. On output, it is updated with the length of the data returned in the *chain_data* parameter.

For CBC, the value must be at least 32.

For ECB and GCM, this parameter is ignored.

For processing rule keywords A28MACGN and A28MACVR, the value must be 8.

For processing rule keywords A28OWFEC, the value must be 8 or 16.

For processing rule keywords A28OWFCL, the value must be 1 to 16 inclusive.

chain_data

Direction	Type
Input/Output	String

A buffer that is used as a work area for sequences of chained symmetric algorithm encipher requests. The exact content and layout of *chain_data* is not described. Your application program must not change the data in this string.

When the keyword INITIAL is used, this is an output parameter and receives data that is needed when enciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output parameter; the value received as output from the previous call in the sequence is provided as input to this call, and in turn, this call will return new *chain_data* that will be used as input on the next call. When CONTINUE is used, both the data (*chain_data* parameter) and the length (*chain_data_length* parameter) must be the same values that were received in these parameters as output on the preceding call to the service in the chained sequence.

For processing rule keywords A28MACGN, A28MACVR, A28OWFEC, and A28OWFCL, this parameter contains the data that will be processed. When AS28RES is specified, the first 8 bytes will be the

residue value from a previous MAC operation. When AS28NORES is specified, the first 8 bytes will be zeros.

A28MACGN, A28MACVR:

Input: (8 bytes)
Input residue value enciphered by the key specified in the *key_parms* parameter or zero.

Output: (8 bytes)
The output residue value enciphered by the key in the *key_parms* parameter.

A28OWFEC:

Input:
Text enciphered by the key specified in the *key_parms* parameter to be used as input to the OWF.

Output:
No output in this parameter.

A28OWFCL:

Input:
Clear text that will be used as input to the OWF.

Output:
No output in this parameter.

For ECB and GCM, this parameter is ignored.

clear_text_length

Direction	Type
Input	Integer

The length of the clear text data in the *clear_text* parameter in bytes. For CBC and ECB processing rules, the length must be a multiple of the algorithm block size. For PKDS-PAD, X9.23PAD, and GCM processing rules, the length may be any value. The maximum length is 2^{32}-1.

For GCM, the value may be zero.

When the Crypto Express adapter is a CEX5 or CEX6, the maximum value is 2^{29}-1. When the Crypto Express adapter is a CEX7 or CEX8, the maximum value is 2^{32}-1.

For processing rules A28OWFEC and A28OWFCL, the value must be zero.

For processing rules A28MACGN and A28MACVR, the maximum length is 1024.

clear_text

Direction	Type
Input	String

The text to be enciphered.

When the *clear_text_length* is zero, this parameter is ignored.

cipher_text_length

Direction	Type
Input/Output	Integer

On input, this parameter specifies the size of the storage pointed to by the *cipher_text* parameter. On output, this parameter has the actual length of the text stored in the buffer addressed by the *cipher_text* parameter.

If process rule PKCS-PAD or X9.23PAD is specified, the cipher text length will exceed the clear text length by at least one byte, and up to 16-bytes. For other process rules, the cipher text length will be equal to the clear text length.

For processing rule keywords A28MACGN and A28MACVR, the value will be 8.

For processing rule keyword A28OWFEC, the value will be 4.

For processing rule keywords A28OWFCL, the value will be the length of the *chain_data* parameter.

cipher_text

Direction	Type
Input/Output	String

The enciphered text the service returns. If PKCS-PAD or X9.23PAD is specified, on output, the ciphertext buffer contains 1 - 16 bytes of data more than the cleartext input buffer contains.

For processing rule keyword A28MACGN, the 8-byte generated MAC will be returned.

For processing rule keyword A28MACVR, on input, this parameter contains the 8-byte MAC to be verified.

For processing rule keywords A28OWFEC and A28OWFCL, the output of the APN OWF will be returned.

optional_data_length

Direction	Type
Input	Integer

The length of the *optional_data* parameter in bytes. For the GCM processing rule, this parameter contains the length of the Additional Authenticated Data (AAD). The value may be 0 to 2^{32}-1.

For all other processing rules, the value must be 0.

optional_data

Direction	Type
Input/Output	String

Optional data required by a specified algorithm or processing mode. For the GCM processing rule, this parameter contains the Additional Authenticated Data (AAD). For all other processing rules, this field is ignored.

You must specify the same *optional_data* used when deciphering the text.

cipher_text_id

Direction	Type
Input	Integer

For CSNBSAE1 and CSNESAE1 only, the ALET of the dataspace in which the *cipher_text* parameter resides.

clear_text_id

Direction	Type
Input	Integer

For CSNBSAE1 and CSNESAE1 only, the ALET of the dataspace in which the *clear_text* parameter resides.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The *clear_text* and *cipher_text* parameters may be in any dataspace. The *initialization_vector* and *optional_data* parameters must be in the caller's address space (primary).

Access control point

Table 211 on page 519 lists the access controls for the Symmetric Algorithm Encipher service.

Table 211. Access controls for Symmetric Algorithm Encipher

Rule array keyword	Access control
AES	Symmetric Algorithm Encipher - Secure AES keys
A28MACGN, A28MACVR	Symmetric Algorithm Encipher – Allow A28MACGN and A28MACVR
A28OWFCL	Symmetric Algorithm Encipher - Allow APN A28OWFCL
A28OWFEC	Symmetric Algorithm Encipher - Allow APN A28OWFEC
GCM	Symmetric Algorithm Encipher – Galois/Counter mode AES

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 212. Symmetric Algorithm Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords GCM and ONLY require the March 2016 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keywords A28MACGN, A28MACVR, A28OWFEC, A28OWFCL, A28RES, A28NORES, and X9.23PAD are not supported. X9.143 key blocks are not supported.

Table 212. Symmetric Algorithm Encipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Keywords GCM and ONLY require the March 2016 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keywords A28MACGN, A28MACVR, A28OWFEC, A28OWFCL, A28RES, A28NORES, and X9.23PAD are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). • Rule array keywords A28MACGN, A28MACVR, A28OWFEC, A28OWFCL, A28RES, and A28NORES are not supported. • Rule array keyword X9.23PAD requires the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keywords A28MACGN, A28MACVR, A28OWFEC, A28OWFCL, A28RES, A28NORES, and X9.23PAD are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	• Rule array keywords A28MACGN, A28MACVR, A28OWFEC, A28OWFCL, A28RES, and A28NORES are not supported. • Rule array keyword X9.23PAD requires the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	• Rule array keywords A28MACGN, A28MACVR, A28OWFEC, A28OWFCL, A28RES, A28NORES, and X9.23PAD require the CCA release 7.4 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	Rule array keywords A28MACGN, A28MACVR, A28OWFEC, A28OWFCL, A28RES, and A28NORES are not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)

Use the symmetric key decipher callable service to decipher data using one of the supported modes. ICSF supports several processing rules to decipher data. You choose the type of processing rule that the Symmetric Key Decipher callable service should use for block chaining. See "Modes of operation" on page 469 for more information.

For any processing rule that indicates that the plaintext can be any length, this length applies to the overall length of the input data. ICV selection rules FIRST and MIDDLE always require the input text to be a multiple of the block size (segment size for CFB-LCFB).

Processing Rule
Purpose

ANSI X9.23
For cipher block chaining. The ciphertext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES). The plaintext will be between 1 and 8 bytes shorter than the ciphertext. This process rule always pads the plaintext during encryption so that ciphertext produced is an exact multiple of the block size, even if the plaintext was already a multiple of the blocksize.

CBC
For cipher block chaining. The ciphertext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The plaintext will have the same length as the ciphertext.

CBC-CS
For cipher block chaining. The ciphertext must be at least the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The plaintext will have the same length as the ciphertext. ICSF implements CBC-CS1 which preserves block ordering.

CFB
Performs cipher feedback encryption with the segment size equal to the block size. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.

CFB-LCFB
Performs cipher feedback encryption with the segment size set by the caller. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.

CTR
Performs counter mode decryption. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.

CUSP
For cipher block chaining. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.

ECB
Performs electronic code book encryption. The ciphertext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The plaintext will have the same length as the ciphertext.

GCM
Perform Galois/Counter mode decryption, which provides both confidentiality and authentication for the plaintext and authentication for the additional authenticated data (AAD). The ciphertext can be any length. The plaintext will have the same length as the ciphertext. Additionally, the authentication tag will be verified before any data is returned.

IPS
For cipher block chaining. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.

OFB
Perform output feedback mode encryption. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.

PKCS-PAD
For cipher block chaining. The ciphertext must be an exact multiple of the block size (8 bytes for DES and 16 bytes for AES). The plaintext will be between 1 and the blocksize (8 bytes for DES, 16 bytes for AES) bytes shorter than the ciphertext. This process rule always pads the ciphertext so that ciphertext produced is an exact multiple of the blocksize, even if the plaintext was already a multiple of the blocksize.

The Advanced Encryption Standard (AES) and Data Encryption Standard (DES) are supported. AES encryption uses a 128-, 192-, or 256-bit key. DES encryption uses a 56-, 112-, or 168-bit key. See the processing rule descriptions for limitations. For each algorithm, certain processing rules are not allowed. See the rule_array parameter description for more information.

All modes except ECB use an initial chaining vector (ICV) in their processing.

All modes that utilize chaining produce a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next decipher call. This results in record chaining.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the key_identifier parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

The key may be specified as a clear key value, an internal clear key token, or the label name of a clear key or an encrypted key in the CKDS.

Choosing between CSNBSYD and CSNBSYD1

CSNBSYD and CSNBSYD1 provide identical functions. When choosing which service to use, consider this:

- **CSNBSYD** requires the ciphertext and plaintext to reside in the caller's primary address space. Also, a program using CSNBSYD adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

  The callable service name for AMODE(64) invocation is CSNESYD.
- **CSNBSYD1** allows the ciphertext and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSYD1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

  For CSNBSYD1, *cipher_text_id* and *clear_text_id* are access list entry token (ALET) parameters of the data spaces containing the ciphertext and plaintext.

  The callable service name for AMODE(64) invocation is CSNESYD1.

Format

```
CALL CSNBSYD(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              key_parms_length,
              key_parms,
              block_size,
              initialization_vector_length,
              initialization_vector,
              chain_data_length,
              chain_data,
              cipher_text_length,
              cipher_text,
              clear_text_length,
              clear_text,
```

```
              optional_data_length,
              optional_data)
```

```
CALL CSNBSYD1(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              key_parms_length,
              key_parms,
              block_size,
              initialization_vector_length,
              initialization_vector,
              chain_data_length,
              chain_data,
              cipher_text_length,
              cipher_text,
              clear_text_length,
              clear_text,
              optional_data_length,
              optional_data,
              cipher_text_id,
              clear_text_id)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 1, 2, 3 or 4.

rule_array

Direction	Type
Input	String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

Table 213. Symmetric Key Decipher Rule Array Keywords

Keyword	Meaning
Algorithm (required)	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes. The *chain_data* field must be at least 32 bytes in length. The OCV is the first 16 bytes in the *chain_data*. AES does not support the CUSP, IPS, or X9.23 processing rules.
DES	Specifies that the Data Encryption Standard (DES) algorithm is to be used. The algorithm, DES or TDES, will be determined from the length of the key supplied. The key length may be 8, 16, or 24. The block size is 8 bytes. The *chain_data* field must be at least 16 bytes in length. The OCV is the first eight bytes in the *chain_data*. DES does not support the CTR or GCM processing rules.
Processing Rule (optional) Rules CBC-CS, CUSP, IPS, PKCS-PAD, and X9.23 should be specified only when there is one request or on the last request of a sequence of chained requests.	
CBC	Performs cipher block chaining. The text length must be a multiple of the block size for the specified algorithm. CBC is the default value.
CBC-CS	CBC mode (cipher block chaining) with ciphertext stealing. The text length must be at least the block size for the specified algorithm.
CFB	CFB mode (cipher feedback) that is compatible with IBM's Encryption Facility product. Input text may be any length.
CFB-LCFB	CFB mode (cipher feedback). This rule allows the value of *s* (the segment size) to be something other than the block size (*s* is set to the block size with the CFB processing rule). *key_parms_length* and *key_parms* are used to set the value of *s*. Input text may be any length.
CTR	CTR mode (counter mode). Input text may be any length.
CUSP	CBC mode (cipher block chaining) that is compatible with IBM's CUSP and PCF products. Input text may be any length.
ECB	Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.

Table 213. Symmetric Key Decipher Rule Array Keywords (continued)

Keyword	Meaning
GCM	GCM (Galois/Counter Mode). *key_parms_length* and *key_parms* are used to indicate the length of the tag (the value *t*) on input and contain the tag on output. Additional Authenticated Data (AAD) is contained in *optional_data_length* and *optional_data*. Input text may be any length. GCM does not support chaining, so CONTINUE and FINAL are not allowed for the ICV Selection rule.
GCM-LG	Processing is similar to the GCM rule. Use only when either *cipher_text_length* or *optional_data_length* are greater than or equal to 256 MiB (2^28 bits) and legacy authentication tags from ICSF FMID HCR77A1 and lower prior to APAR OA46558 are to be verified.
IPS	CBC mode (cipher block chaining) that is compatible with IBM's IPS product. Input text may be any length.
OFB	OFB mode (output feedback). Input text may be any length.
PKCS-PAD	CBC mode (cipher block chaining) but the ciphertext must be an exact multiple of the block length (8 bytes for DES and 16 bytes for AES). The plaintext will be 1 to 8 bytes shorter for DES and 1 to 16 bytes shorter for AES than the ciphertext.
X9.23	CBC mode (cipher block chaining) for 1 to 8 bytes of padding dropped from the output clear text.
Key Rule (optional)	
KEY-CLR	This specifies that the key parameter contains a clear key value. KEY-CLR is the default value.
KEYIDENT	This specifies that the *key_identifier* field will be an internal clear token, or the label name of a clear key or encrypted key in the CKDS. Normal CKDS labelname syntax is required.
ICV Selection (optional)	
INITIAL	This specifies taking the initialization vector from the *initialization_vector* parameter. INITIAL is the default value. INITIAL is not valid with processing rule GCM.
CONTINUE	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the *chain_data* parameter points. CONTINUE is not valid for processing rules ECB, GCM, or X9.23.
FINAL	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the *chain_data* parameter points. Using FINAL indicates that this call contains the last portion of data. FINAL is valid for processing rules CBC-CS, CFB, CFB-LCBF, CTR, and OFB.
ONLY	This specifies taking the initialization vector from the *initialization_vector* parameter and that the entirety of the data to be processed is in this single call. ONLY is valid for processing rules CBC-CS, CFB, CFB-LCFB, CTR, GCM, and OFB.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter.

For the KEY-CLR keyword, this parameter is the length in bytes of the clear key value only. The maximum size is 256 bytes.

For the KEYIDENT keyword, this parameter value must be 64.

key_identifier

Direction	Type
Input/Output	String

For the KEY-CLR keyword, this specifies the cipher key. The parameter must be left justified.

For the KEYIDENT keyword, this specifies an internal clear token, or the label name of a clear key or an encrypted key in the CKDS. Normal CKDS label name syntax is required.

The key algorithm may be DES or AES. The key type must be DATA for AES fixed-length key tokens, DATA or CIPHER for DES fixed-length key tokens, or CIPHER for variable-length AES key tokens.

For X9.143 (TR-31) key blocks, the identifier is the label of a variable-length key block of an AES, DES, or TDES data-encrypting key: key usage D0, algorithm A, D, or T, and mode use is B or D. IBM optional block "10" is required.

Notes:

- To use a fixed-length DES or AES encrypted key in the CKDS, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES). For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.
- To use an encrypted variable-length AES CIPHER key token, the key token must allow decryption, the cipher mode is being used, and export to the CPACF protected key format. For information on creating key tokens with the specified attributes, see "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297. See the CKDS KEYS utility Key Attributes panel for information on the current attributes of a CKDS key.
- To use an encrypted data-encryption key in a TR-31 key block, the CPACF export control attribute must be enabled in the IBM optional block '10'.

key_parms_length

Direction	Type
Input	Integer

The length of the *key_parms* parameter.

- For the CFB-LCFB and CTR processing rules, this length must be 1.
- For the GCM processing rule, this is the length in bytes of the authentication tag to be verified. Valid lengths are 4, 8, 12, 13, 14, 15, 16. Using a length of 4 or 8 is strongly discouraged.
- For all other processing rules, this field is ignored.

You must specify the same length used when enciphering the text.

key_parms

Direction	Type
Input	String

This parameter contains key-related parameters specific to the encryption algorithm and processing mode.

- For the CFB-LCFB processing rule, this 1-byte field specifies the segment size in bytes. Valid values are 1 to the block size, inclusive. The block size is eight for DES and sixteen for AES.
- For the CTR processing rule, this 1-byte field specifies the number of low order bytes of the counter to be incremented. The remaining upper order bytes are the nonce. Valid values are 1 to the block size, inclusive. The blocksize is sixteen for AES.
- For the GCM processing rule, this contains the authentication tag for the provided ciphertext (*cipher_text* parameter) and additional authenticated data (*optional_data* parameter).
- For all other processing rules, this field is ignored.

For the modes where *key_parms* is used, you must specify the same *key_parms* used when enciphering the text using the Symmetric Key Encipher.

block_size

Direction	Type
Input	Integer

This parameter contains the processing size of the text block in bytes. This value will be algorithm specific. Be sure to specify the same block size as used to encipher the text.

initialization_vector_length

Direction	Type
Input	Integer

The length of the *initialization_vector* parameter. This parameter is ignored for the ECB processing rule. For the GCM processing rule, NIST recommends a length of 12, but tolerates any non-zero length. For all other processing rules, the length should be equal to the block length for the algorithm specified.

initialization_vector

Direction	Type
Input	String

The initialization chaining value. You must use the same ICV that was used to encipher the data. This parameter is ignored for the ECB processing rule.

chain_data_length

Direction	Type
Input/Output	Integer

The length of the *chain_data* parameter. On output, the actual length of the chaining vector will be stored in the parameter. This parameter is ignored if the ICV selection keyword is ONLY.

For AES, this length must be at least 32 bytes. For DES, this length must be at least 16 bytes.

chain_data

Direction	Type
Input/Output	String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The direction is output if the ICV selection keyword is INITIAL. This parameter is ignored if the ICV selection keyword is ONLY.

The mapping of the *chain_data* depends on the algorithm specified. For AES, the *chain_data* field must be at least 32 bytes in length. The OCV is in the first 16 bytes in the *chain_data*. For DES, *chain_data* field must be at least 16 bytes in length. The OCV is the first 8 bytes in the *chain_data*.

cipher_text_length

Direction	Type
Input	Integer

The length of the ciphertext. A zero value in the *cipher_text_length* parameter is not valid except with the GCM processing rule when performing a GMAC operation. The length must be a multiple of the algorithm block size for the CBC, ECB, and PKCS-PAD processing rules, but may be any length with the other processing rules.

cipher_text

Direction	Type
Input	String

The text to be deciphered.

clear_text_length

Direction	Type
Input/Output	Integer

On input, this parameter specifies the size of the storage pointed to by the *clear_text* parameter, which must be at least the same size as the *cipher_text_length*. On output, this parameter has the actual length of the text stored in the *clear_text* parameter.

If process rules PKCS-PAD or X9.23PAD are used, the *clear_text_length* returned will be less than the *cipher_text_length* since padding bytes are removed.

clear_text

Direction	Type
Output	String

The deciphered text the service returns.

optional_data_length

Direction	Type
Input	Integer

The length of the *optional_data* parameter. For the GCM processing rule, this parameter contains the length of the Additional Authenticated Data (AAD). For all other processing rules, this field is ignored.

optional_data

Direction	Type
Input	String

Optional data required by a specified algorithm or processing mode. For the GCM processing rule, this parameter contains the Additional Authenticated Data (AAD). For all other processing rules, this field is ignored.

You must specify the same *optional_data* used when enciphering the text using Symmetric Key Encipher.

cipher_text_id

Direction	Type
Input	Integer

For CSNBSYD1 only, the ALET of the ciphertext to be deciphered.

clear_text_id

Direction	Type
Input	Integer

For CSNBSYD1 only, the ALET of the clear text supplied by the application.

Usage notes

- SAF may be invoked to verify the caller is authorized to use the specified key label stored in the CKDS.
- To use a fixed-length DES or AES encrypted key in the CKDS, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES). For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.
- To use an encrypted variable-length AES CIPHER key token, the key token must allow decryption, the cipher mode is being used, and export to the CPACF protected key format.
- No pre- or post-processing exits are enabled for this service.
- The master keys need to be loaded only when using this service with encrypted key labels.
- The AES algorithm will use hardware if it is available. Otherwise, clear key operations will be performed in software.
- AES has the same availability restrictions as triple-DES.
- This service will fail if execution would cause destructive overlay of the *cipher_text* field.

Access control points

When the label of an encrypted key is specified for the *key_identifier* parameter, the appropriate access control point must be enabled.

Table 214. Required access control points for Symmetric Key Decipher

Key algorithm	Access control point
Crypto Express5 and earlier CCA coprocessors	
AES	High-performance secure AES keys
DES	High-performance secure DES keys

Table 214. Required access control points for Symmetric Key Decipher (continued)	
Key algorithm	**Access control point**
Crypto Express6 and later CCA coprocessors	
All	Authenticated Key Export - EXPTSK

For AES CIPHER keys (CEX6C and later), the following access controls must be enabled:

- **Authenticated Key Export - DRVTXKEY**
- **Authenticated Key Export - EXPTSK**
- **Authenticated Key Export - SETSNKEY**

These access controls should be enabled by default, but **Authenticated Key Export - EXPTSK** may not be enabled and will cause return codes that are misleading.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 215. Symmetric Key Decipher required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. DES and AES CIPHER key tokens and triple-length DES DATA keys with a non-zero control vector are not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. Encrypted variable-length AES CIPHER key tokens require a CEX6C. Triple-length DES DATA keys with a non-zero control vector require a CEX6C with the November 2018 or later licensed internal code (LIC). DES CIPHER key tokens require a CEX6C with the P41458.002 or later MCL. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 215. Symmetric Key Decipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Encrypted keys require a Crypto express coprocessor. DES and AES CIPHER key tokens and triple-length DES DATA keys with a non-zero control vector are not supported. X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Related information

You **cannot** destructively overlap the plaintext and ciphertext fields. For example:

```
pppppp
     cccccc  is supported.

cccccc
     pppppp  is not supported.

ppppppccccccc is supported.

p represents the plaintext and c represents the ciphertext.
```

"Cipher processing rules" on page 1643 discusses the cipher processing rules.

Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)

Use the symmetric key encipher callable service to encipher data using one of the supported modes. ICSF supports several processing rules to encipher data. You choose the type of processing rule that the Symmetric Key Encipher callable service should use for the block chaining. See "Modes of operation" on page 469 for more information.

For any processing rule that indicates that the plaintext can be any length, this length applies to the overall length of the input data. ICV selection rules FIRST and MIDDLE always require the input text to be a multiple of the block size (segment size for CFB-LCFB).

Processing Rule
Purpose

ANSI X9.23
For cipher block chaining. The plaintext may be any length. The ciphertext will be between 1 and 8 bytes longer than the plaintext. This process rule always pads the plaintext during encryption so that ciphertext produced is an exact multiple of the block size, even if the plaintext was already a multiple of the blocksize.

CBC
For cipher block chaining. The plaintext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The ciphertext will have the same length as the plaintext.

CBC-CS
For cipher block chaining. The plaintext must be at least the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The plaintext will have the same length as the ciphertext. ICSF implements CBC-CS1 which preserves block ordering.

CFB
Performs cipher feedback encryption with the segment size equal to the block size. The plaintext can be of any length. The ciphertext will have the same length as the plaintext.

CFB-LCFB
Performs cipher feedback encryption with the segment size set by the caller. The plaintext can be of any length. The ciphertext will have the same length as the plaintext.

CTR
Performs counter mode encryption. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.

CUSP
For cipher block chaining. The plaintext can be of any length. The ciphertext will have the same length as the plaintext.

ECB
Performs electronic code book encryption. The plaintext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The ciphertext will have the same length as the plaintext.

GCM
Perform Galois/Counter mode encryption, which provides both confidentiality and authentication for the plaintext and authentication for the additional authenticated data (AAD). The plaintext can be of any length. The ciphertext will have the same length as the plaintext. Additionally, the authentication tag will be verified before any data is returned.

IPS
For cipher block chaining. The plaintext can be of any length. The ciphertext will have the same length as the plaintext.

OFB
Perform output feedback mode encryption. The plaintext can be of any length. The ciphertext will have the same length as the plaintext.

PKCS-PAD
For cipher block chaining. The plaintext may be any length. The ciphertext will be between 1 and 8 bytes longer than the plaintext. This process rule always pads the ciphertext so that ciphertext produced is an exact multiple of the blocksize, even if the plaintext was already a multiple of the blocksize.

The Advanced Encryption Standard (AES) and Data Encryption Standard (DES) are supported. AES encryption uses a 128-, 192-, or 256-bit key. DES encryption uses a 56-, 112-, or 168-bit key. See the processing rule descriptions for limitations. For each algorithm, certain processing rules are not allowed. See the rule_array parameter description for more information.

All modes except ECB use an initial chaining vector (ICV) in their processing.

All modes that tolerate chaining produce a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in record chaining.

The selection between single-DES encryption mode and triple-DES encryption mode is controlled by the length of the key supplied in the *key_identifier* parameter. If a single-length key is supplied, single-DES encryption is performed. If a double-length or triple-length key is supplied, triple-DES encryption is performed.

The key may be specified as a clear key value, an internal clear key token, or the label name of a clear key or an encrypted key in the CKDS.

Choosing between CSNBSYE and CSNBSYE1

CSNBSYE and CSNBSYE1 provide identical functions. When choosing which service to use, consider this:

- **CSNBSYE** requires the cleartext and ciphertext to reside in the caller's primary address space. Also, a program using CSNBSYE adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

  The callable service name for AMODE(64) invocation is CSNESYE.
- **CSNBSYE1** allows the cleartext and ciphertext to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using CSNBSYE1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

  For CSNBSYE1, *clear_text_id* and *cipher_text_id* are access list entry token (ALET) parameters of the data spaces containing the cleartext and ciphertext.

  The callable service name for AMODE(64) invocation is CSNESYE1.

Format

```
CALL CSNBSYE(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             key_parms_length,
             key_parms,
             block_size,
             initialization_vector_length,
             initialization_vector,
             chain_data_length,
             chain_data,
             clear_text_length,
             clear_text,
             cipher_text_length,
             cipher_text,
             optional_data_length,
             optional_data)
```

```
CALL CSNBSYE1(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             length,
             key_identifier,
             key_parms_length,
             key_parms,
             block_size,
             initialization_vector_length,
```

```
                initialization_vector,
                chain_data_length,
                chain_data,
                clear_text_length,
                clear_text,
                cipher_text_length,
                cipher_text,
                optional_data_length,
                optional_data,
                clear_text_id,
                cipher_text_id)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 1, 2, 3 or 4.

rule_array

Direction	Type
Input	String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

Table 216. Symmetric Key Encipher Rule Array Keywords	
Keyword	**Meaning**
Algorithm (required)	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes. The *chain_data* field must be at least 32 bytes in length. The OCV is the first 16 bytes in the *chain_data*. AES does not support the CUSP, IPS, or X9.23 processing rules.
DES	Specifies that the Data Encryption Standard (DES) algorithm is to be used. The algorithm, DES or TDES, will be determined from the length of the key supplied. The key length may be 8, 16, or 24. The block size is 8 bytes. The *chain_data* field must be at least 16 bytes in length. The OCV is the first eight bytes in the *chain_data*. DES does not support the CTR or GCM processing rules.
Processing Rule (optional) Rules CBC-CS, CUSP, IPS, PKCS-PAD, and X9.23 should be specified only when there is one request or on the last request of a sequence of chained requests.	
CBC	CBC mode (cipher block chaining). The text length must be a multiple of the block size for the specified algorithm. CBC is the default value.
CBC-CS	CBC mode (cipher block chaining) with ciphertext stealing. The text length must be at least the block size for the specified algorithm.
CFB	CFB mode (cipher feedback) that is compatible with IBM's Encryption Facility product. Input text may be any length.
CFB-LCFB	CFB mode (cipher feedback). This rule allows the value of s (the segment size) to be something other than the block size (s is set to the block size with the CFB processing rule). The *key_parms_length* and *key_parms* parameters are used to set the value of *s*. Input text may be any length.
CTR	CTR mode (counter mode). Input text may be any length.
CUSP	CBC mode (cipher block chaining) that is compatible with IBM's CUSP and PCF products. Input text may be any length.
ECB	ECB mode (electronic codebook). The text length must be a multiple of the block size for the specified algorithm.
GCM	GCM mode (Galois/Counter Mode). The *key_parms_length* and *key_parms* parameters are used to indicate the length of the tag (the value *t*) on input and contain the tag on output. Additional Authenticated Data (AAD) is contained in the *optional_data_length* and *optional_data* parameters. Input text may be any length.
GCM-LG	Processing is similar to the GCM rule. Use only when either *clear_text_length* or *optional_data_length* are greater than or equal to 256 MiB (2^28 bits) and legacy authentication tags from ICSF FMID HCR77A1 and lower prior to APAR OA46558 are to be generated.
IPS	CBC mode (cipher block chaining) that is compatible with IBM's IPS product. Input text may be any length.
OFB	OFB mode (output feedback). Input text may be any length.

Table 216. Symmetric Key Encipher Rule Array Keywords (continued)	
Keyword	**Meaning**
PKCS-PAD	CBC mode (cipher block chaining) not necessarily in exact multiples of the block length (8 bytes for DES and 16 bytes for AES). PKCS-PAD always pads the plaintext so that the ciphertext produced is an exact multiple of the block length and longer than the plaintext.
X9.23	CBC mode (cipher block chaining) for 1 to 8 bytes of padding added according to ANSI X9.23. Input text may be any length.
Key Rule (optional)	
KEY-CLR	This specifies that the key parameter contains a clear key value. KEY-CLR is the default.
KEYIDENT	This specifies that the *key_identifier* field will be an internal clear token, or the label name of a clear key or encrypted key in the CKDS. Normal CKDS labelname syntax is required.
ICV Selection (optional)	
INITIAL	This specifies taking the initialization vector from the *initialization_vector* parameter. INITIAL is the default value. INITIAL is not valid with processing rule GCM.
CONTINUE	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the *chain_data* parameter points. CONTINUE is not valid for processing rules ECB, GCM, or X9.23.
FINAL	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the *chain_data* parameter points. Using FINAL indicates that this call contains the last portion of data. FINAL is valid for processing rules CBC-CS, CFB, CFB-LCBF, CTR, and OFB.
ONLY	This specifies taking the initialization vector from the *initialization_vector* parameter and that the entirety of the data to be processed is in this single call. ONLY is valid for processing rules CBC-CS, CFB, CFB-LCFB, CTR, GCM, and OFB.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter.

For the KEY-CLR keyword, this parameter is the length in bytes of the clear key value only.

For the KEYIDENT keyword, this parameter value must be 64.

key_identifier

Direction	Type
Input/Output	String

For the KEY-CLR keyword, this specifies the cipher key. The parameter must be left justified.

For the KEYIDENT keyword, this specifies an internal clear token, or the label name of a clear key or an encrypted key in the CKDS. Normal CKDS label name syntax is required.

The key algorithm may be DES or AES. The key type must be DATA for AES fixed-length key tokens, DATA or CIPHER for DES fixed-length key tokens, or CIPHER for variable-length AES key tokens.

For X9.143 (TR-31) key blocks, the identifier is the label of a variable-length key block of an AES, DES, or TDES data-encrypting key: key usage D0, algorithm A, D, or T, and mode use is B or E. IBM optional block "10" is required.

Notes:

- To use a fixed-length DES or AES encrypted key in the CKDS, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES). For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.
- To use an encrypted variable-length AES CIPHER key token, the key token must allow encryption, the cipher mode is being used, and export to the CPACF protected key format. For information on creating key tokens with the specified attributes, see "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297. See the CKDS KEYS utility Key Attributes panel for information on the current attributes of a CKDS key.
- To use an encrypted data-encryption key in a TR-31 key block, the CPACF export control attribute must be enabled in the IBM optional block '10'.

key_parms_length

Direction	Type
Input	Integer

The length of the *key_parms* parameter.

- For the CFB-LCFB and CTR processing rules, this length must be 1.
- For the GCM processing rule, this is the length in bytes of the authentication tag to be generated. Valid lengths are 4, 8, 12, 13, 14, 15, 16. Using a length of 4 or 8 is strongly discouraged.
- For all other processing rules, this field is ignored.

When deciphering the text, you must specify this same length.

key_parms

Direction	Type
Input/Output	String

This parameter contains key-related parameters specific to the encryption algorithm and processing mode.

- For the CFB-LCFB processing rule, this 1-byte field specifies the segment size in bytes. Valid values are 1 to the blocksize, inclusive. The block size is eight for DES and sixteen for AES.
- For the CTR processing rule, this 1-byte field specifies the number of low order bytes of the counter to be incremented. The remaining upper order bytes are the nonce. Valid values are 1 to the block size, inclusive. The blocksize is sixteen for AES.
- For the GCM processing rule, this will contain the generated authentication tag for the provided plaintext (plain_text parameter) and additional authenticated data (optional_data parameter).
- For all other processing rules, this field is ignored.

For the modes where *key_parms* is used, you must specify the same *key_parms* when deciphering the text using the Symmetric Key Decipher callable service.

block_size

Direction	Type
Input	Integer

This parameter contains the processing size of the text block in bytes. This value will be algorithm specific.

initialization_vector_length

Direction	Type
Input	Integer

The length of the *initialization_vector* parameter. This parameter is ignored for the ECB processing rule. For the GCM processing rule, NIST recommends a length of 12, but tolerates any non-zero length. For all other processing rules, the length should be equal to the block length for the algorithm specified.

initialization_vector

Direction	Type
Input	String

The initialization chaining value. You must use the same ICV to decipher the data. This parameter is ignored for the ECB processing rule.

chain_data_length

Direction	Type
Input/Output	Integer

The length of the *chain_data* parameter. On output, the actual length of the chaining vector will be stored in the parameter. This parameter is ignored if the ICV selection keyword is ONLY.

For AES, this length must be at least 32 bytes. For DES, this length must be at least 16 bytes.

chain_data

Direction	Type
Input/Output	String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The direction is output if the ICV selection keyword is INITIAL. This parameter is ignored if the ICV selection keyword is ONLY.

The mapping of the *chain_data* depends on the algorithm specified. For AES, the *chain_data* field must be at least 32 bytes in length. The OCV is in the first 16 bytes in the *chain_data*. For DES, the *chain_data* field must be at least 16 bytes in length. The OCV is the first 8 bytes in the *chain_data*.

clear_text_length

Direction	Type
Input	Integer

The length of the cleartext. A zero value in the *clear_text_length* parameter is not valid except with the GCM processing rule when performing a GMAC operation. The length must be a multiple of the algorithm block size for the CBC, ECB, and PKCS-PAD processing rules, but may be any length with

the other processing rules. For the processing rules that support partial blocks (or segments for CFB-LCFB), it is recommended that the final block (or segment) be the only one that is partial. Having a partial block in the middle is not a supported operation as defined by the standards documents and may not be portable to other encryption systems.

clear_text

Direction	Type
Input	String

The text to be enciphered.

cipher_text_length

Direction	Type
Input/Output	Integer

On input, this parameter specifies the size of the storage pointed to by the *cipher_text* parameter. On output, this parameter has the actual length of the text stored in the buffer addressed by the *cipher_text* parameter.

cipher_text

Direction	Type
Output	String

The enciphered text the service returns.

optional_data_length

Direction	Type
Input	Integer

The length of the *optional_data* parameter. For the GCM processing rule, this parameter contains the length of the Additional Authenticated Data (AAD), and may be any length, including zero. For all other processing rules, this field is ignored.

optional_data

Direction	Type
Input	String

Optional data required by a specified algorithm. Optional data required by a specified algorithm or processing mode. For the GCM processing rule, this parameter contains the Additional Authenticated Data (AAD). For all other processing rules, this field is ignored.

You must specify the same optional_data when deciphering the text using Symmetric Key Decipher.

clear_text_id

Direction	Type
Input	Integer

For CSNBSYE1 only, the ALET of the clear text to be enciphered.

cipher_text_id

Direction	Type
Input	Integer

For CSNBSYE1 only, the ALET of the ciphertext that the application supplied.

Usage notes

- SAF may be invoked to verify the caller is authorized to use the specified key label stored in the CKDS.
- To use a fixed-length DES or AES encrypted key in the CKDS, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES). For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.
- To use an encrypted variable-length AES CIPHER key token, the key token must allow encryption, the cipher mode is being used, and export to the CPACF protected key format.
- No pre- or post-processing exits are enabled for this service.
- The master keys need to be loaded only when using this service with the encrypted key labels.
- The AES algorithm will use hardware if it is available. Otherwise, clear key operations will be performed in software.
- AES has the same availability restrictions as triple-DES.
- This service will fail if execution would cause destructive overlay of the *clear_text* field.

Access control points

When the label of an encrypted key is specified for the *key_identifier* parameter, the appropriate access control point must be enabled.

Table 217. Required access control points for Symmetric Key Encipher

Key algorithm	Access control point
Crypto Express5 and earlier CCA coprocessors	
AES	High-performance secure AES keys
DES	High-performance secure DES keys
Crypto Express6 and later CCA coprocessors	
All	Authenticated Key Export - EXPTSK

For AES CIPHER keys (CEX6C and later), the following access controls must be enabled:

- **Authenticated Key Export - DRVTXKEY**
- **Authenticated Key Export - EXPTSK**
- **Authenticated Key Export - SETSNKEY**

These access controls should be enabled by default, but **Authenticated Key Export - EXPTSK** may not be enabled and will cause return codes that are misleading.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 218. Symmetric Key Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. DES and AES CIPHER key tokens and triple-length DES DATA keys with a non-zero control vector are not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. Encrypted variable-length AES CIPHER key tokens require a CEX6C. Triple-length DES DATA keys with a non-zero control vector require a CEX6C with the November 2018 or later licensed internal code (LIC). DES CIPHER key tokens require a CEX6C with the P41458.002 or later MCL. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Encrypted keys require a Crypto express coprocessor. DES and AES CIPHER key tokens and triple-length DES DATA keys with a non-zero control vector are not supported. X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Related information

You **cannot** destructively overlap the plaintext and ciphertext fields. For example:

```
pppppp
     cccccc  is not supported.

cccccc
     pppppp  is supported.

ppppppcccccc is supported.

p represents the plaintext and c represents the ciphertext.
```

The method used to produce the OCV is the same with the CBC and X9.23 processing rules. However, that method is different from the method used by the CUSP and IPS processing rules.

“Cipher processing rules” on page 1643 discusses the cipher processing rules.

Chapter 7. Verifying data integrity and authenticating messages

ICSF provides several methods to verify the integrity of transmitted messages and stored data:

- Message authentication code (MAC)
- Hash functions, including modification detection code (MDC) processing and one-way hash generation

Note: You can also use digital signatures (see Chapter 12, "Using digital signatures," on page 1109) to authenticate messages.

The choice of callable service depends on the security requirements of the environment in which you are operating. If you need to ensure the authenticity of the sender as well as the integrity of the data, and both the sender and receiver can share a secret key, consider message authentication code processing. If you need to ensure the integrity of transmitted data in an environment where it is not possible for the sender and the receiver to share a secret cryptographic key, consider hashing functions, such as the modification detection code process.

The callable services are described in the following topics:

- "HMAC Generate (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)" on page 545
- "HMAC Verify (CSNBHMV or CSNBHMV1 and CSNEHMV or CSNEHMV1)" on page 550
- "MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)" on page 555
- "MAC Generate2 (CSNBMGN2, CSNBMGN3, CSNEMGN2, and CSNEMGN3)" on page 562
- "MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)" on page 567
- "MAC Verify2 (CSNBMVR2, CSNBMVR3, CSNEMVR2, and CSNEMVR3)" on page 574
- "MDC Generate (CSNBMDG or CSNBMDG1 and CSNEMDG or CSNEMDG1)" on page 579
- "One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1)" on page 584
- "Symmetric MAC Generate (CSNBSMG or CSNBSMG1 and CSNESMG or CSNESMG1)" on page 589
- "Symmetric MAC Verify (CSNBSMV or CSNBSMV1 and CSNESMV or CSNESMV1)" on page 594

How MACs are used

When a message is sent, an application program can generate an authentication code for it using the MAC Generate, MAC Generate2, or HMAC Generate callable service. ICSF supports:

- The ANSI X9.9-1 basic procedure.
- The ANSI X9.19 basic procedure and optional double key MAC procedure for DES.
- Block cipher-based MAC algorithm, called CMAC (NIST SP 800-38B) for AES.
- FIPS-198 Keyed-Hash Message Authentication Code method for HMAC.

The message text may be in clear or encrypted form. The originator of the message sends the MAC with the message text.

When the receiver gets the message, an application program calls the *MAC verification callable service*. The callable service generates a MAC using the same algorithm as the sender and either the single-length, double-length, or triple-length MAC verification key, the single-length, double-length, or triple-length MAC generation key, or DATA key, and the message text. The MACVER callable service compares the MAC it generates with the one sent with the message and issues a return code that indicates whether the MACs match. If the return code indicates that the MACs match, the receiver can accept the message as genuine and unaltered. If the return code indicates that the MACs do not match, the receiver can assume that the message is either bogus or has been altered. The newly computed MAC is not revealed outside the cryptographic feature.

In a similar manner, MACs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

Secure use of the MAC generation and MAC verification services requires the use of MAC and MACVER keys in these services, respectively. To accomplish this, the originator of the message generates a MAC/MACVER key pair, uses the MAC key in the MAC generation service, and exports the MACVER key to the receiver. The originator of the message enforces key separation on the link by encrypting the MACVER key under a transport key that is not an NOCV key before exporting the key to the receiver. With this type of key separation enforced, the receiver can only receive a MACVER key and can use only this key in the MAC verification service. This ensures that the receiver cannot alter the message and produce a valid MAC with the altered message. These security features are not present if DATA keys are used in the MAC generation service, or if DATA or MAC keys are used in the MAC verification service.

By using MACs, you get the following benefits:

- **For data transmitted over a network,** you can validate the authenticity of the message as well as ensure that the data has not been altered during transmission. For example, an active eavesdropper can tap into a transmission line, and interject bogus messages or alter sensitive data being transmitted. If the data is accompanied by a MAC, the recipient can use a callable service to detect whether the data has been altered. Since both the sender and receiver share a secret key, the receiver can use a callable service that calculates a MAC on the received message and compares it to the MAC transmitted with the message. If the comparison is equal, the message may be accepted as unaltered. Furthermore, since the shared key is secret, when a MAC is verified it can be assumed that the sender was, in fact, the other person who knew the secret key.
- **For data stored on tape or DASD,** you can ensure that the data read back onto the system was the same as the data written onto the tape or DASD. For example, someone might be able to bypass access controls. Such an access might escape the notice of auditors. However, if a MAC is stored with the data, and verified when the data is read, you can detect alterations to the data.

How hashing functions are used

Hashing functions include the MDC and one-way hash. You need to hash text before submitting it to digital signature services (see Chapter 12, “Using digital signatures,” on page 1109).

How MDCs are used

When a message is sent, an application program can generate a modification detection code for it using the *MDC generation callable service*. The service computes the modification detection code, a 128-bit value, using a one-way cryptographic function and the message text (which itself may be in clear or encrypted form). The originator of the message ensures that the MDC is transmitted with integrity to the intended receiver of the message. For example, the MDC could be published in a reliable source of public information.

When the receiver gets the message, an application program calls the *MDC callable service*. The callable service generates an MDC by using the same one-way cryptographic function and the message text. The application program can compare the new MDC with the one generated by the originator of the message. If the MDCs match, the receiver knows that the message was not altered.

In a similar manner, MDCs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

By using MDCs, you get the following benefits:

For data transmitted over a network between locations that do not share a secret key
: You can ensure that the data has not been altered during transmission. It is easy to compute an MDC for specific data, yet hard to find data that will result in a given MDC. In effect, the problem of ensuring the integrity of a large file is reduced to ensuring the integrity of a 128-bit value.

For data stored on tape or DASD
: You can ensure that the data read back onto the system was the same as the data written onto the tape or DASD. Once an MDC has been established for a file, the MDC generation callable service can

be run at any later time on the file. The resulting MDC can be compared with the stored MDC to detect deliberate or inadvertent modification.

SHA-1 is a FIPS standard required for DSS. MD5 is a hashing algorithm used to derive Message Digests in Digital Signature applications.

HMAC Generate (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)

Use the HMAC generate callable service to generate a keyed hash message authentication code (MAC) for the text string provided as input.

The HMAC key supplied in a variable-length symmetric key token may contain a secure or a clear key value. Clear HMAC keys can be built using the Key Token Build2 (CSNBKTB2) callable service.

The callable service names for AMODE(64) are CSNEHMG and CSFEHMG1.

Choosing between CSNBHMG and CSNBHMG1

CSNBHMG and CSNBHMG1 provide identical functions. When choosing which service to use, consider the following:

- CSNBHMG requires the application-supplied text to reside in the caller's primary address space.
- CSNBHMG1 allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to process more data with one call. For CSNBHMG1, *text_id_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

Format

```
CALL CSNBHMG(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              text_length,
              text,
              chaining_vector_length,
              chaining_vector,
              mac_length,
              mac )
```

```
CALL CSNBHMG1(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              text_length,
              text,
              chaining_vector_length,
              chaining_vector,
              mac_length,
              mac,
              text_id_in )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Output	Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 2 or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The following table lists the keywords. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 219. Keywords for HMAC Generate Control Information

Keyword	Meaning
Token algorithm (One required)	
HMAC	Specifies the HMAC algorithm to be used to generate the MAC.
Hash method (One required)	

Table 219. Keywords for HMAC Generate Control Information (continued)	
Keyword	**Meaning**
SHA-1	Specifies the FIPS-198 HMAC procedure using the SHA-1 hash method, a symmetric key and text to produce a 20-byte (160-bit) MAC.
SHA-224	Specifies the FIPS-198 HMAC procedure using the SHA-224 hash method, a symmetric key and text to produce a 28-byte (224-bit) MAC.
SHA-256	Specifies the FIPS-198 HMAC procedure using the SHA-256 hash method, a symmetric key and text to produce a 32-byte (256-bit) MAC.
SHA-384	Specifies the FIPS-198 HMAC procedure using the SHA-384 hash method, a symmetric key and text to produce a 48-byte (384-bit) MAC.
SHA-512	Specifies the FIPS-198 HMAC procedure using the SHA-512 hash method, a symmetric key and text to produce a 64-byte (512-bit) MAC.
Segmenting Control (One optional)	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

key_identifier_length

Direction	**Type**
Input	Integer

Specifies the length in bytes of the *key_identifier* parameter.

If the *key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	**Type**
Input/Output	String

The identifier of the key to generate the MAC. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA keys, the identifier is a variable-length HMAC key token of key type MAC with and the key usage fields indicating GENERATE and match the hash method specified in the rule array.

For X9.143 keys, the identifier is a variable-length key block of an HMAC key: algorithm H, key usage of M7, and mode of use C or G. It must have an "HM" optional block with a hash identifier matching the hash method specified in the rule array.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

The minimum required HMAC key size depends on the hash method specified in the rule array.

Table 220. Minimum HMAC key size in bits based on hash method

Hash method	Minimum HMAC key size (in bits)
SHA-1	80
SHA-224	112
SHA-256	128
SHA-384	192
SHA-512	256

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of *text* is 2147483647 bytes. For FIRST and MIDDLE calls, the *text_length* must be a multiple of 64 for SHA-1, SHA-224 and SHA-256 and a multiple of 128 for SHA-384 and SHA-512 hash methods.

text

Direction	Type
Input	String

The application-supplied text for which the MAC is generated.

chaining_vector_length

Direction	Type
Input/Output	Integer

The length of the *chaining_vector* in bytes. The value must be 128 bytes.

chaining_vector

Direction	Type
Input/Output	String

An 128-byte string that ICSF uses as a system work area. The chaining vector permits data to be chained from one invocation call to another.

On calls that use ONLY or FIRST rules, initialize this parameter as binary zeros. On calls that use MIDDLE or LAST rules, your application program must not change the data in this string.

mac_length

Direction	Type
Input/Output	Integer

The length of the *mac* parameter in bytes. This parameter is updated to the actual length of the *mac* parameter on output. The minimum value is 4, and the maximum value is 64.

mac

Direction	Type
Output	String

The field in which the callable service returns the MAC value if the segmenting rule is ONLY or LAST.

text_id_in

Direction	Type
Input	Integer

For CSNBHMG1 only, the ALET of the text for which the MAC is generated.

Usage notes

Secure key calls are routed to a CCA coprocessor. Clear key requests exploit the CPACF instructions.

Access control points

When a clear HMAC key token is specified, these access controls are not used.

This table lists the access control points in the domain role that control the function for this service.

Table 221. HMAC Generate Access Control Points

Hash method	Access control point
SHA-1	HMAC Generate - SHA-1
SHA-224	HMAC Generate - SHA-224
SHA-256	HMAC Generate - SHA-256
SHA-384	HMAC Generate - SHA-384
SHA-512	HMAC Generate - SHA-512

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 222. HMAC Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys only. X9.143 key blocks are not supported.

Table 222. HMAC Generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys only. X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys only. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	Clear keys only. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

HMAC Verify (CSNBHMV or CSNBHMV1 and CSNEHMV or CSNEHMV1)

Use the HMAC verify callable service to verify a keyed hash message authentication code (MAC) for the text string provided as input.

The HMAC key supplied in a variable-length symmetric key token may contain a secure or a clear key value. Clear HMAC keys can be built using the Key Token Build2 (CSNBKTB2) callable service.

The callable service names for AMODE(64) are CSNEHMV and CSFEHMV1.

Choosing between CSNBHMV and CSNBHMV1

CSNBHMV and CSNBHMV1 provide identical functions. When choosing which service to use, consider the following:

- CSNBHMV requires the application-supplied text to reside in the caller's primary address space.
- CSNBHMV1 allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to process more data with one call. For CSNBHMV1, *text_id_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

Format

```
CALL CSNBHMV(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              text_length,
              text,
              chaining_vector_length,
              chaining_vector,
              mac_length,
              mac )
```

```
CALL CSNBHMV1(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              text_length,
              text,
              chaining_vector_length,
              chaining_vector,
              mac_length,
              mac,
              text_id_in )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 2 or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The following table lists the keywords. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 223. Keywords for HMAC Verify Control Information

Keyword	Meaning
Token algorithm (One required)	
HMAC	Specifies the HMAC algorithm to be used to verify the MAC.
Hash method (One required)	
SHA-1	Specifies the FIPS-198 HMAC procedure using the SHA-1 hash method, a symmetric key and text to produce a 20-byte (160-bit) MAC.
SHA-224	Specifies the FIPS-198 HMAC procedure using the SHA-224 hash method, a symmetric key and text to produce a 28-byte (224-bit) MAC.
SHA-256	Specifies the FIPS-198 HMAC procedure using the SHA-256 hash method, a symmetric key and text to produce a 32-byte (256-bit) MAC.
SHA-384	Specifies the FIPS-198 HMAC procedure using the SHA-384 hash method, a symmetric key and text to produce a 48-byte (384-bit) MAC.
SHA-512	Specifies the FIPS-198 HMAC procedure using the SHA-512 hash method, a symmetric key and text to produce a 64-byte (512-bit) MAC.
Segmenting Control (optional)	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.

Table 223. Keywords for HMAC Verify Control Information (continued)	
Keyword	**Meaning**
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *key_identifier* parameter.

If the *key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify the MAC. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA keys, the identifier is a variable-length HMAC key token of key type MAC with and the key usage fields indicating GENERATE or VERIFY and match the hash method specified in the rule array.

For X9.143 keys, the identifier is a variable-length key block of an HMAC key: algorithm H, key usage of M7, and mode of use C or V. It must have an "HM" optional block with a hash identifier matching the hash method specified in the rule array.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

The minimum required HMAC key size depends on the hash method specified in the rule array.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of *text* is 2147483647 bytes. For FIRST and MIDDLE calls, the *text_length* must be a multiple of 64 for SHA-1, SHA-224 and SHA-256 and a multiple of 128 for SHA-384 and SHA-512 hash methods.

text

Direction	Type
Input	String

The application-supplied text for which the MAC is generated.

chaining_vector_length

Direction	Type
Input/Output	Integer

The length of the *chaining_vector* in bytes. The value must be 128 bytes.

chaining_vector

Direction	Type
Input/Output	String

An 128-byte string that ICSF uses as a system work area. The chaining vector permits data to be chained from one invocation call to another.

On calls that use ONLY or FIRST rules, initialize this parameter as binary zeros. On calls that use MIDDLE or LAST rules, your application program must not change the data in this string.

mac_length

Direction	Type
Input	Integer

The length of the *mac* parameter in bytes. The maximum value is 64.

mac

Direction	Type
Input	String

The field that contains the MAC value you want to verify.

text_id_in

Direction	Type
Input	Integer

For CSNBHMV1 only, the ALET of the text for which the MAC is generated.

Usage notes

Secure key calls are routed to a CCA coprocessor. Clear key requests exploit the CPACF instructions.

Access control points

When a clear HMAC key token is specified, these access controls are not used.

This table lists the access control points in the domain role that control the function for this service.

Table 224. HMAC Verify Access Control Points

Hash method	Access control point
SHA-1	HMAC Verify - SHA-1
SHA-224	HMAC Verify - SHA-224
SHA-256	HMAC Verify - SHA-256
SHA-384	HMAC Verify - SHA-384
SHA-512	HMAC Verify - SHA-512

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 225. HMAC Verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys only. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys only. X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys only. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	Clear keys only. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)

Use the MAC Generate callable service to generate a 4-, 6-, or 8-byte message authentication code (MAC) for an application-supplied text string. The following DES or Triple-DES MAC ciphering methods are supported:

- ANS X9.9 Option 1 Procedure and either Triple-DES encryption in cipher-block chaining (CBC) mode or single-DES encryption.
- ANS X9.19 Optional Procedure and Triple-DES encryption in CBC mode.
- EMV-related message-padding and calculation method using single DES.
- NIST SP800-38B (2005) procedure and Triple-DES encryption in CBC mode.

Specify the MAC ciphering method through the choice of a rule-array keyword.

EMVMAC and EMVMACD
EMV message authentication processes are defined in the EMV 4.0 Book 2, Annex A1.2. The MAC is computed based on ISO/IEC 9797-1, MAC Algorithm 1 or MAC Algorithm 3, depending on key length. When specifying a single-length DES key, use EMVMAC. When specifying a double-length Triple-DES key, use EMVMACD.

Note: The EMV specification permits the MAC to be 4 - 8 bytes in length. This service only uses MAC lengths of 4, 6, and 8 bytes.

TDESCMAC
NIST SP800-38B (2005) procedure. Uses double-length or triple-length Triple-DES key.

TDES-MAC
ANS X9.9 Option 1 (binary data) procedure using ISO 16609 CBC-mode Triple-DES (TDES) encryption of the data. Uses a double-length or triple-length Triple-DES key.

X9.9-1
ANS X9.9 Option 1 (binary data) procedure, by default when a single-length DES key is provided. This is the same as ANS X9.19 Basic Procedure and ISO/IEC 9797-1, MAC Algorithm 1.

X9.19OPT
ANS X9.19 Optional Procedure, by default when a double-length Triple-DES key is provided. This is the same as ISO/IEC 9797-1, MAC Algorithm 3.

Any of these DES key types can be used: DATA, DATAM, or MAC.

Choosing between CSNBMGN and CSNBMGN1

CSNBMGN and CSNBMGN1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBMGN** requires the application-supplied text to reside in the caller's primary address space. Also, a program using CSNBMGN adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

  The callable service name for AMODE(64) invocation is CSNEMGN.
- **CSNBMGN1** allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to process more data with one call. However, a program using CSNBMGN1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified before it can run with other cryptographic products that follow this programming interface.

  The callable service name for AMODE(64) invocation is CSNEMGN1.

  For CSNBMGN1, *text_id_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

Format

```
CALL CSNBMGN(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              key_identifier,
              text_length,
              text,
              rule_array_count,
              rule_array,
              chaining_vector,
              mac )
```

```
CALL CSNBMGN1(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               key_identifier,
```

```
                text_length,
                text,
                rule_array_count,
                rule_array,
                chaining_vector,
                mac,
                text_id_in )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_identifier

Direction	Type
Input/Output	String

The identifier of the MAC key to generate the MAC. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage.

The type of key depends on the MAC process rule in the *rule_array* parameter.

For CCA key tokens, the identifier is a 64-byte DES key token containing:

- A single, double length, or triple-length MAC generation key,
- A DATAM key, or
- A single-length or double-length DATA key.

For X9.143 key blocks, the identifier is a variable-length key block of a DES or TDES MAC key with mode of use C or G, and key usage M0, M1, M3, M6, dependent upon the MAC process rule specified (see Table 226 on page 558).

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 2147483647 bytes. If the *text_length* is not a multiple of 8 bytes and if the ONLY or LAST keyword of the *rule_array* parameter is called, the text is padded in accordance with the processing rule specified.

Note: The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced.

text

Direction	Type
Input	String

The application-supplied text for which the MAC is generated.

rule_array_count

Direction	Type
Input	Integer

The number of keywords specified in the *rule_array* parameter. The value can be 0, 1, 2, or 3.

rule_array

Direction	Type
Input	Character String

Zero to three keywords that provide control information to the callable service. The keywords are shown in Table 226 on page 558. The keywords must be in 24 bytes of contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. For example,

```
'X9.9-1  MIDDLE  MACLEN4 '
```

The order of the *rule_array* keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

Table 226. Keywords for MAC Generate control information

Keyword	Meaning
MAC Process Rules (optional)	

Table 226. Keywords for MAC Generate control information (continued)	
Keyword	**Meaning**
EMVMAC	EMV padding rule with a single-length MAC key. The *key_identifier* parameter must identify a single-length CCA MAC or DATA key token or an X9.143 (TR-31) key block with key usage M1, algorithm D, and mode of use C or G. The text is always padded so that the resulting text length is a multiple of 8 bytes. The first pad character is X'80'. The remaining 0 to 7 pad characters are X'00'.
EMVMACD	EMV padding rule with double key MAC. The *key_identifier* parameter must identify a double-length or triple-length CCA MAC key or an X9.143 (TR-31) key block with key usage M3, algorithm T, and mode of use C or G. The padding rules are the same as for EMVMAC.
X9.19OPT	ANSI X9.19 optional double key MAC procedure. The *key_identifier* parameter must identify a double-length MAC key. The *key_identifier* parameter must identify a double-length or triple-length CCA MAC key or an X9.143 (TR-31) key block with key usage M3, algorithm T, and mode of use C or G. The padding rules are the same as for X9.9-1.
X9.9-1	ANSI X9.9-1 and X9.19 basic procedure. The *key_identifier* parameter must identify a single-length CCA MAC or single-length CCA DATA key or an X9.143 (TR-31) key block with key usage M1, algorithm D, and mode of use C or G. X9.9-1 causes the MAC to be computed from all of the data. The text is padded only if the text length is not a multiple of 8 bytes. If padding is required, the pad character X'00' is used. This is the default value.
TDES-MAC	ISO 16609 procedure. The *key_identifier* parameter must identify a double-length or triple-length CCA MAC or a double-length CCA DATA key or an X9.143 (TR-31) key block with key usage M0, algorithm T, and mode of use C or G. The text is padded only if the text length is not a multiple of 8 bytes.
TDESCMAC	NIST SP800-38B (2005) procedure. The *key_identifier* must identify a double-length or triple-length CCA MAC key or a double-length DATA key or an X9.143 (TR-31) key block with key usage M6, algorithm T, and mode of use C or G.
Segmenting Control (optional)	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.

Table 226. Keywords for MAC Generate control information (continued)	
Keyword	**Meaning**
ONLY	Only call; segmenting is not employed by the application program. This is the default value.
MAC Length and Presentation (optional)	
HEX-8	Generates a 4-byte MAC value and presents it as 8 hexadecimal characters.
HEX-9	Generates a 4-byte MAC value and presents it as 2 groups of 4 hexadecimal characters with a space between the groups.
MACLEN4	Generates a 4-byte MAC value. This is the default value.
MACLEN6	Generates a 6-byte MAC value.
MACLEN8	Generates an 8-byte MAC value.

chaining_vector

Direction	Type
Input/Output	String

An 128-byte string that ICSF uses as a system work area. The chaining vector permits data to be chained from one invocation call to another.

On calls that use ONLY or FIRST rules, initialize this parameter as binary zeros. On calls that use MIDDLE or LAST rules, your application program must not change the data in this string.

mac

Direction	Type
Output	String

The 8-byte or 9-byte field in which the callable service returns the MAC value if the segmenting rule is ONLY or LAST. Allocate an 8-byte field for MAC values of 4 bytes, 6 bytes, 8 bytes, or HEX-8. Allocate a 9-byte MAC field if you specify HEX-9 in the *rule_array* parameter.

text_id_in

Direction	Type
Input	Integer

For CSNBMGN1/CSNEMGN1 only, the ALET of the text for which the MAC is generated.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control point

The **MAC Generate** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 227. MAC Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 227. MAC Generate required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Related information

For more information about MAC processing rules and segmenting control, refer to IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface reference.

The MAC Verify callable service is described in "MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)" on page 567.

MAC Generate2 (CSNBMGN2, CSNBMGN3, CSNEMGN2, and CSNEMGN3)

Use the MAC Generate2 callable service to generate a keyed hash message authentication code (HMAC) or a ciphered message authentication code (CMAC) for the message string provided as input.

The MAC generate key must be in a variable-length HMAC key token for HMAC and an AES MAC token for CMAC. The key must have key usage that allows the key to generate MACs.

For the HMAC algorithm, clear key tokens are supported. Clear HMAC keys can be built using the Key Token Build2 (CSNBKTB2) callable service.

The callable service names for AMODE(64) are CSNEMGN2 and CSNEMGN3.

Choosing between CSNBMGN2 and CSNBMGN3

CSNBMGN2 and CSNBMGN3 provide identical functions. When choosing which service to use, consider the following:

- CSNBMGN2 requires the application-supplied text to reside in the caller's primary address space.
- CSNBMGN3 allows the application-supplied text to reside either in the caller's primary address space or in a data space. This allows you to process more data with one call. For CSNBMGN3, *text_id_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

Format

```
CALL CSNBMGN2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              text_length,
              text,
              chaining_vector_length,
              chaining_vector,
              mac_length,
              mac )
```

```
CALL CSNBMGN3(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             text_length,
             text,
             chaining_vector_length,
             chaining_vector,
             mac_length,
             mac,
             text_id_in )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1, 2, or 3.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 228. Keywords for MAC Generate2 control information

Keyword	Meaning
Token algorithm (One required)	
AES	Specifies the use of the AES CMAC algorithm to generate a MAC.
HMAC	Specifies the use of the HMAC algorithm to generate a MAC.
Hash method (One required for HMAC only)	
SHA-1	Specifies the use of the SHA-1 hash method.
SHA-224	Specifies the use of the SHA-224 hash method.
SHA-256	Specifies the use of the SHA-256 hash method.
SHA-384	Specifies the use of the SHA-384 hash method.
SHA-512	Specifies the use of the SHA-512 hash method.
Segmenting Control (One optional)	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

key_identifier_length

Direction	Type
Input	Integer

key_identifier_length specifies the length in bytes of the *key_identifier* parameter.

If the *key_identifier* parameter contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA keys, the identifier is a variable-length key token of:

- AES message authentication key: the key type is MAC, and the key usage fields must indicate GENONLY or GENERATE and CMAC.
- HMAC message authentication key: the key type is MAC and the key usage fields must indicate GENONLY or GENERATE and the hash algorithm selected. The key identifier may be a clear or secure operational key token.

For X9.143 keys, the identifier is a variable-length key block of:

- AES message authentication key: key usage M6, algorithm A, and mode of use C or G.
- HMAC message authentication key: key usage M7, algorithm H or I, and mode of use C or G.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

text_length

Direction	Type
Input	Integer

The length of the text you supplied in the *text* parameter. The maximum length of *text* is 2147483647 bytes. For FIRST and MIDDLE calls, the *text_length* must be:

- A multiple of 64 for the SHA-1, SHA-224, and SHA-256 hash methods.
- A multiple of 128 for the SHA-384 and SHA-512 hash methods.
- A multiple of 16 for the AES CMAC method.

text

Direction	Type
Input	String

The application-supplied text for which the MAC is generated.

chaining_vector_length

Direction	Type
Input/Output	Integer

chaining_vector_length specifies the length in bytes of the *chaining_vector* parameter. The value must be 128.

chaining_vector

Direction	Type
Input/Output	String

An 128-byte string that ICSF uses as a system work area. The chaining vector permits data to be chained from one invocation call to another.

On calls that use ONLY or FIRST rules, initialize this parameter as binary zeros. On calls that use MIDDLE or LAST rules, your application program must not change the data in this string.

mac_length

Direction	Type
Input/Output	Integer

The length of the *mac* parameter in bytes. This parameter is updated to the actual length of the *mac* parameter on output. For HMAC, the minimum value is 4 and the maximum value is 64. For AES, the value must be 16.

mac

Direction	Type
Output	String

The field in which the callable service returns the MAC value if the segmenting rule is ONLY or LAST.

text_id_in

Direction	Type
Input	Integer

For CSNBMGN3 only, the ALET of the text for which the MAC is generated.

Usage notes

Secure key calls are routed to a CCA coprocessor. Clear key requests exploit the CPACF instructions.

Access control points

When a clear HMAC key token is specified, these access controls are not used.

This table lists the access control points in the domain role that control the function for this service.

Table 229. MAC Generate2 Access Control Points

Hash method	Access control point
CMAC	MAC Generate2 - AES CMAC
SHA-1	HMAC Generate - SHA-1
SHA-224	HMAC Generate - SHA-224
SHA-256	HMAC Generate - SHA-256
SHA-384	HMAC Generate - SHA-384
SHA-512	HMAC Generate - SHA-512

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 230. MAC Generate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	For AES, a MAC length of 8 requires the July 2015 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys for HMAC algorithm only. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	For AES, a MAC length of 8 requires the July 2015 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys for HMAC algorithm only. X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys for HMAC algorithm only. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	Clear keys for HMAC algorithm only. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)

Use the MAC Verify callable service to verify a 4-, 6-, or 8-byte message authentication code (MAC) for an application-supplied text string. The following DES or Triple-DES MAC ciphering methods are supported:

- ANS X9.9 Option 1 Procedure and either Triple-DES encryption in cipher-block chaining (CBC) mode or single-DES encryption.
- ANS X9.19 Optional Procedure and Triple-DES encryption in CBC mode.
- EMV-related message-padding and calculation method using single DES.
- NIST SP800-38B (2005) procedure and Triple-DES encryption in CBC mode.

Specify the MAC ciphering method through the choice of a rule-array keyword.

EMVMAC and EMVMACD
EMV message authentication processes are defined in the EMV 4.0 Book 2, Annex A1.2. The MAC is computed based on ISO/IEC 9797-1, MAC Algorithm 1 or MAC Algorithm 3, depending on key length. When specifying a single-length DES key, use EMVMAC. When specifying a double-length Triple-DES key, use EMVMACD.

Note: The EMV specification permits the MAC to be 4 - 8 bytes in length. This service only uses MAC lengths of 4, 6, and 8 bytes.

TDESCMAC
NIST SP800-38B (2005) procedure. Uses double-length or triple-length Triple-DES key.

TDES-MAC
ANS X9.9 Option 1 (binary data) procedure using ISO 16609 CBC-mode Triple-DES (TDES) encryption of the data. Uses a double-length or triple-length Triple-DES key.

X9.9-1
ANS X9.9 Option 1 (binary data) procedure, by default when a single-length DES key is provided. This is the same as ANS X9.19 Basic Procedure and ISO/IEC 9797-1, MAC Algorithm 1.

X9.19OPT
ANS X9.19 Optional Procedure, by default when a double-length Triple-DES key is provided. This is the same as ISO/IEC 9797-1, MAC Algorithm 3.

Any of these DES key types can be used: DATA, DATAM, or MAC.

Choosing between CSNBMVR and CSNBMVR1

CSNBMVR and CSNBMVR1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBMVR** requires the application-supplied text to reside in the caller's primary address space. Also, a program using CSNBMVR adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

  The callable service name for AMODE(64) invocation is CSNEMVR.
- **CSNBMVR1** allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to verify more data with one call. However, a program using CSNBMVR1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface and may need to be modified before it can run with other cryptographic products that follow this programming interface.

  The callable service name for AMODE(64) invocation is CSNEMVR1.

  For CSNBMVR1, *text_id_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

Format

```
CALL CSNBMVR(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_identifier,
             text_length,
             text,
```

```
              rule_array_count,
              rule_array,
              chaining_vector,
              mac )
```

```
CALL CSNBMVR1(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               key_identifier,
               text_length,
               text,
               rule_array_count,
               rule_array,
               chaining_vector,
               mac,
               text_id_in )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_identifier

Direction	Type
Input/Output	String

The identifier of the MAC key to verify the MAC. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage.

The type of key depends on the MAC process rule in the rule_array parameter.

For CCA key tokens, the identifier is a 64-byte DES key token containing:

- A single, double-length, or triple-length MAC verify key,
- A single, double length, or triple-length MAC generation key,
- A DATAM or DATAMV key, or
- A single-length or double-length DATA key.

For X9.143 key blocks, the identifier is a variable-length key block of a DES or TDES MAC key with mode of use C or V, and key usage M0, M1, M3, M6, dependent upon the MAC process rule specified (see Table 231 on page 571).

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 2147483647 bytes. If the *text_length* parameter is not a multiple of 8 bytes and if the ONLY or LAST keyword of the *rule_array* parameter is called, the text is padded in accordance with the processing rule specified.

Note: The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647).

text

Direction	Type
Input	String

The application-supplied text for which the MAC is generated.

rule_array_count

Direction	Type
Input	Integer

The number of keywords specified in the *rule_array* parameter. The value can be 0, 1, 2, or 3.

rule_array

Direction	Type
Input	String

Zero to three keywords that provide control information to the callable service. The keywords are shown in Table 231 on page 571. The keywords must be in 24 bytes of contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. For example,

```
'X9.9-1  MIDDLE  MACLEN4 '
```

The order of the *rule_array* keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

Table 231. Keywords for MAC Verify control information	
Keyword	**Meaning**
MAC Process Rules (optional)	
EMVMAC	EMV padding rule with a single-length MAC key. The *key_identifier* parameter must identify a single-length CCA MAC, MACVER, or DATA key token or an X9.143 (TR-31) key block with key usage M1, algorithm D, and mode of use C or V. The text is always padded so that the resulting text length is a multiple of 8 bytes. The first pad character is X'80'. The remaining 0 to 7 pad characters are X'00'.
EMVMACD	EMV padding rule with double key MAC. The *key_identifier* parameter must identify a double-length or triple-length CCA MAC or MACVER key or an X9.143 (TR-31) key block with key usage M3, algorithm T, and mode of use C or V. The padding rules are the same as for EMVMAC.
X9.19OPT	ANSI X9.19 optional double key MAC procedure. The *key_identifier* parameter must identify a double-length or triple-length CCA MAC or MACVER key or an X9.143 (TR-31) key block with key usage M3, algorithm T, and mode of use C or V. The padding rules are the same as for X9.9-1. ANSI X9.9-1 and X9.19 basic procedure.
X9.9-1	ANSI X9.9-1 and X9.19 basic procedure. The *key_identifier* parameter must identify a single-length CCA MAC, MACVER, or DATA key or an X9.143 (TR-31) key block with key usage M1, algorithm D, and mode of use C or V. X9.9-1 causes the MAC to be computed from all of the data. The text is padded only if the text length is not a multiple of 8 bytes. If padding is required, the pad character X'00' is used. This is the default value.
TDES-MAC	ISO 16609 procedure. The *key_identifier* parameter must identify a double-length or triple-length CCA MAC or MACVER or a double-length CCA DATA key or an X9.143 (TR-31) key block with key usage M0, algorithm T, and mode of use C or V. The text is padded only if the text length is not a multiple of 8 bytes.
TDESCMAC	NIST SP800-38B (2005) procedure. The *key_identifier* parameter must identify a double-length or triple-length CCA MAC or MACVER key or a double-length DATA key or an X9.143 (TR-31) key block with key usage M6, algorithm T, and mode of use C or V.
Segmenting Control (optional)	
FIRST	First call; this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.

Table 231. Keywords for MAC Verify control information (continued)	
Keyword	**Meaning**
ONLY	Only call; the application program does not employ segmenting. This is the default value.
MAC Length and Presentation (optional)	
HEX-8	Verifies a 4-byte MAC value that is represented as 8 hexadecimal characters.
HEX-9	Verifies a 4-byte MAC value that is represented as 2 groups of 4 hexadecimal characters with a space character between the groups.
MACLEN4	Verifies a 4-byte MAC value. This is the default value.
MACLEN6	Verifies a 6-byte MAC value.
MACLEN8	Verifies an 8-byte MAC value.

chaining_vector

Direction	Type
Input/Output	String

An 128-byte string that ICSF uses as a system work area. The chaining vector permits data to be chained from one invocation call to another.

On calls that use ONLY or FIRST rules, initialize this parameter as binary zeros. On calls that use MIDDLE or LAST rules, your application program must not change the data in this string.

mac

Direction	Type
Input	String

The 8- or 9-byte field that contains the MAC value you want to verify. The value in the field must be left-justified and padded with zeros. If you specified the X'09' keyword in the rule_array parameter, the input MAC is 9 bytes.

text_id_in

Direction	Type
Input	Integer

For CSNBMVR1/CSNEMVR1 only, the ALET of the text for which the MAC is to be verified.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

To verify a MAC in one call, specify the ONLY keyword on the segmenting rule keyword for the *rule_array* parameter. For two or more calls, specify the FIRST keyword for the first input block, MIDDLE for intermediate blocks (if any), and LAST for the last block.

For a given text string, the MAC resulting from the verification process is the same regardless of how the text is segmented, or how it was segmented when the original MAC was generated.

Access control point

The **MAC Verify** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 232. MAC Verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keyword TDESCMAC requires the May 2021 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 232. MAC Verify required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Related information

For more information about MAC processing rules and segmenting control, refer to IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface reference.

The MAC Generate callable service is described in "MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)" on page 555.

MAC Verify2 (CSNBMVR2, CSNBMVR3, CSNEMVR2, and CSNEMVR3)

Use the MAC Verify2 callable service to verify a keyed hash message authentication code (HMAC) or a ciphered message authentication code (CMAC) for the message text provided as input.

The MAC verify key must be in a variable-length HMAC key token for HMAC and an AES MAC token for CMAC. The key must have key usage that allows the key to verify MACs.

For the HMAC algorithm, clear key tokens are supported. Clear HMAC keys can be built using the Key Token Build2 (CSNBKTB2) callable service.

The callable service names for AMODE(64) are CSNEMVR2 and CSNEMVR3.

Choosing between CSNBMVR2 and CSNBMVR3

CSNBMVR2 and CSNBMVR3 provide identical functions. When choosing which service to use, consider the following:

- CSNBMVR2 requires the application-supplied text to reside in the caller's primary address space.
- CSNBMVR3 allows the application-supplied text to reside either in the caller's primary address space or in a data space. This allows you to process more data with one call. For CSNBMVR3, *text_id_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

Format

```
CALL CSNBMVR2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             text_length,
             text,
             chaining_vector_length,
             chaining_vector,
             mac_length,
             mac )
```

```
CALL CSNBMVR3(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             text_length,
             text,
             chaining_vector_length,
             chaining_vector,
             mac_length,
             mac,
             text_id_in )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1, 2, or 3.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 233. Keywords for MAC Verify2 control information

Keyword	Meaning
Token algorithm (One required)	
AES	Specifies the use of the AES CMAC algorithm to generate a MAC.
HMAC	Specifies the use of the HMAC algorithm to generate a MAC.
Hash method (One required for HMAC only)	
SHA-1	Specifies the use of the SHA-1 hash method.
SHA-224	Specifies the use of the SHA-224 hash method.
SHA-256	Specifies the use of the SHA-256 hash method.
SHA-384	Specifies the use of the SHA-384 hash method.
SHA-512	Specifies the use of the SHA-512 hash method.
Segmenting Control (One optional)	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

key_identifier_length

Direction	Type
Input	Integer

key_identifier_length specifies the length in bytes of the *key_identifier* parameter.

If the *key_identifier* parameter contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify the MAC. The *key identifier* is an operational token or the key label of an operational token in key storage.

For CCA keys, the identifier is a variable-length key token of:

- AES message authentication key: the key type is MAC, and the key usage fields must indicate GENERATE or VERIFY and CMAC.
- HMAC message authentication key: the key type is MAC and the key usage fields must indicate GENERATE or VERIFY and the hash algorithm selected. The key identifier may be a clear or secure operational key token.

For X9.143 keys, the identifier is a key block of:

- AES message authentication key: key usage M6, algorithm A, and mode of use C or V.
- HMAC message authentication key: key usage M7, algorithm H or I, and mode of use C or V.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

text_length

Direction	Type
Input	Integer

The length of the text you supplied in the *text* parameter. The maximum length of *text* is 2147483647 bytes. For FIRST and MIDDLE calls, the *text_length* must be:

- A multiple of 64 for the SHA-1, SHA-224, and SHA-256 hash methods.
- A multiple of 128 for the SHA-384 and SHA-512 hash methods.
- A multiple of 16 for the AES CMAC method.

text

Direction	Type
Input	String

The application-supplied text for which the MAC is generated.

chaining_vector_length

Direction	Type
Input/Output	Integer

chaining_vector_length specifies the length in bytes of the *chaining_vector* parameter. The value must be 128.

chaining_vector

Direction	Type
Input/Output	String

An 128-byte string that ICSF uses as a system work area. The chaining vector permits data to be chained from one invocation call to another.

On calls that use ONLY or FIRST rules, initialize this parameter as binary zeros. On calls that use MIDDLE or LAST rules, your application program must not change the data in this string.

mac_length

Direction	Type
Input	Integer

The length of the *mac* parameter in bytes. For HMAC, the maximum value is 64. For AES, the value must be 8 or 16.

mac

Direction	Type
Input	String

The field that contains the MAC value you want to verify.

text_id_in

Direction	Type
Input	Integer

For CSNBMVR3 only, the ALET of the text for which the MAC is to be verified.

Usage notes

Secure key calls are routed to a CCA coprocessor. Clear key requests exploit the CPACF instructions.

Access control points

When a clear HMAC key token is specified, these access controls are not used.

This table lists the access control points in the domain role that control the function for this service.

Table 234. MAC Verify2 Access Control Points

Hash method	Access control point
CMAC	MAC Verify2 - AES CMAC
SHA-1	HMAC Verify - SHA-1
SHA-224	HMAC Verify - SHA-224
SHA-256	HMAC Verify - SHA-256
SHA-384	HMAC Verify - SHA-384
SHA-512	HMAC Verify - SHA-512

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 235. MAC Verify2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	For AES, a MAC length of 8 requires the July 2015 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys for HMAC algorithm only. X9.143 key blocks are not supported.

Table 235. MAC Verify2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	For AES, a MAC length of 8 requires the July 2015 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys for HMAC algorithm only. X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions	Clear keys for HMAC algorithm only. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	Clear keys for HMAC algorithm only. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

MDC Generate (CSNBMDG or CSNBMDG1 and CSNEMDG or CSNEMDG1)

A modification detection code (MDC) can be used to provide a form of support for data integrity.

Use the MDC Generate callable service to generate a 128-bit modification detection code (MDC) for an application-supplied text string.

The returned MDC value should be securely stored and/or sent to another user. To validate the integrity of the text string at a later time, the MDC Generate callable service is again used to generate a 128-bit MDC. The new MDC value is compared with the original MDC value. If the values are equal, the text is accepted as unchanged.

Choosing between CSNBMDG and CSNBMDG1

CSNBMDG and CSNBMDG1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBMDG** requires the application-supplied text to reside in the caller's primary address space. Also, a program using CSNBMDG adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

  The callable service name for AMODE(64) invocation is CSNEMDG.
- **CSNBMDG1** allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to process more data with one call. However, a program using CSNBMDG1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface and may need to be modified before it can run with other cryptographic products that follow this programming interface.

  The callable service name for AMODE(64) invocation is CSNEMDG1.

  For CSNBMDG1, *text_id_in* parameter specifies the access list entry token (ALET) for the data space containing the application-supplied text.

Format

```
CALL CSNBMDG(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             text_length,
             text,
             rule_array_count,
             rule_array,
             chaining_vector,
             mdc )
```

```
CALL CSNBMDG1(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              text_length,
              text,
              rule_array_count,
              rule_array,
              chaining_vector,
              mdc,
              text_id_in )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 2147483647 bytes.

Note: The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647).

Additional restrictions on length of the text depend on whether padding of the text is requested, and on the segmenting control used.

- When padding is requested (by specifying a process rule of PADMDC-2 or PADMDC-4 in the *rule_array* parameter), a text length of 0 is valid for any segment control specified in the *rule_array* parameter (FIRST, MIDDLE, LAST, or ONLY). When LAST or ONLY is specified, the supplied text will be padded with X'FF's and a padding count is specified in the last byte to bring the total text length to the next multiple of 8 that is greater than or equal to 16,
- When no padding is requested (by specifying a process rule of MDC-2 or MDC-4), the total length of the text provided (over a single or segmented calls) must be at least 16 bytes, and a multiple of 8.

  For segmented calls with no padding, text length of 0 is valid on any of the calls provided the total length over the segmented calls is at least 16 and a multiple of 8.

  For a single call (that is, segment control is ONLY) with no padding, the length the text provided must be at least 16, and a multiple of 8.

text

Direction	Type
Input	String

The application-supplied text for which the MDC is generated.

rule_array_count

Direction	Type
Input	Integer

The number of keywords specified in the *rule_array* parameter. This value must be 2.

rule_array

Direction	Type
Input	Character String

The two keywords that provide control information to the callable service are shown in Table 236 on page 582. The two keywords must be in 16 bytes of contiguous storage with each of the two keywords left-justified in its own 8-byte location and padded on the right with blanks. For example,

```
'MDC-2   FIRST   '
```

Choose one of the MDC process rule control keywords and one of the segmenting control keywords from the following table.

Table 236. Keywords for MDC Generate control information

Keyword	Meaning
MDC Process Rules (required)	
MDC-2	MDC-2 specifies two encipherments per 8 bytes of input text and no padding of the input text.
MDC-4	MDC-4 specifies four encipherments per 8 bytes of input text and no padding of the input text.
PADMDC-2	PADMDC-2 specifies two encipherments per 8 bytes of input text and padding of the input text. When the segment rule specifies ONLY or LAST, the input text is padded with X'FF's and a padding count is specified in the last byte to bring the total text length to the next even multiple of 8 that is greater than, or equal to, 16.
PADMDC-4	PADMDC-4 specifies four encipherments per 8 bytes of input text and padding of the input text. When the segment rule specifies ONLY or LAST, the input text is padded with X'FF's and a padding count is specified in the last byte to bring the total text length to the next even multiple of 8 that is greater than, or equal to, 16.
Segmenting Control (required)	
FIRST	First call; this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program.

chaining_vector

Direction	Type
Input/Output	String

An 128-byte string that ICSF uses as a system work area. The chaining vector permits data to be chained from one invocation call to another.

On calls that use ONLY or FIRST rules, initialize this parameter as binary zeros. On calls that use MIDDLE or LAST rules, your application program must not change the data in this string.

mdc

Direction	Type
Input/Output	String

A 16-byte field in which the callable service returns the MDC value when the segmenting rule is ONLY or LAST. When the segmenting rule is FIRST or MIDDLE, the value returned in this field is an intermediate MDC value that will be used as input for a subsequent call and must not be changed by the application program.

text_id_in

Direction	Type
Input	Integer

For CSNBMDG1/CSNEMDG1 only, the ALET for the data space containing the text for which the MDC is to be generated.

Usage notes

To calculate an MDC in one call, specify the ONLY keyword for segmenting control in the *rule_array* parameter. For more than one call, specify the FIRST keyword for the first input block, the MIDDLE keyword for any intermediate blocks, and the LAST keyword for the last block. For a given text string, the resulting MDC is the same whether the text is segmented or not.

The two versions of MDC calculation (with two or four encipherments per 8 bytes of input text) allow the caller to trade a performance improvement for a decrease in security. Since 2 encipherments create results different from the results of 4 encipherments, ensure that you use the same number of encipherments to verify the MDC value.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 237. MDC Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions	
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions	

Table 237. MDC Generate required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions	
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	

One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1)

Use the One-Way Hash Generate callable service to generate a one-way hash or provide an extendable-output function on specified text. This service supports the following methods:

Table 238. Blocksize and hash length for hash methods		
Hash method	**Blocksize (in bytes)**	**Hash length (in bytes)**
MD5	64	16
RPMD-160	64	20
SHA-1	64	20
SHA-224	64	28^
SHA-256	64	32
SHA-384	128	48^
SHA-512	128	64
SHA3-224	144	28
SHA3-256	136	32
SHA3-384	104	48
SHA3-512	72	64
SHAKE128	168	*
SHAKE256	136	*

^
See description in the *hash_length* parameter for additional requirements.

*****
Extendable-output functions can generate any non-zero length.

The callable service names for AMODE(64) invocation are CSNEOWH and CSNEOWH1.

Format

```
CALL CSNBOWH(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             text_length,
```

```
              text,
              chaining_vector_length,
              chaining_vector,
              hash_length,
              hash)
```

```
CALL CSNBOWH1(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               rule_array_count,
               rule_array,
               text_length,
               text,
               chaining_vector_length,
               chaining_vector,
               hash_length,
               hash,
               text_id_in)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 1 or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service are listed in Table 239 on page 586. The optional chaining flag keyword indicates whether calls to this service are chained together logically to overcome buffer size limitations. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 239. Keywords for One-Way Hash Generate Rule Array Control Information

Keyword	Meaning
Hash Method (required)	
MD5	Hash algorithm is MD5 algorithm. Use this hash method for PKCS-1.0 and PKCS-1.1.
MD5-LG	Hash algorithm is similar to the MD5 algorithm. Legacy hash values from release HCR7751 and lower prior to APAR OA33657 will be generated for verification purposes with previously archived hash values.
RPMD-LG	Hash algorithm is similar to the RIPEMD-160. Legacy hash values from release HCR7751 and lower prior to APAR OA33657 will be generated for verification purposes with previously archived hash values.
RPMD-160	Hash algorithm is RIPEMD-160.
SHA-1	Hash algorithm is SHA-1 algorithm. Use this hash method for DSS.
SHA-224	Hash algorithm is SHA-224 algorithm.
SHA-256	Hash algorithm is SHA-256 algorithm.
SHA-384	Hash algorithm is SHA-384 algorithm.
SHA-512	Hash algorithm is SHA-512 algorithm.
SHA3-224	Hash algorithm is SHA3-224 algorithm.
SHA3-256	Hash algorithm is SHA3-256 algorithm.
SHA3-384	Hash algorithm is SHA3-384 algorithm.
SHA3-512	Hash algorithm is SHA3-512 algorithm.
SHAKE128	Extendable-output function is SHAKE-128.
SHAKE256	Extendable-output function is SHAKE-256.
SHA1LG	Hash algorithm is similar to the SHA-1 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.

Table 239. Keywords for One-Way Hash Generate Rule Array Control Information (continued)	
Keyword	**Meaning**
SHA224LG	Hash algorithm is similar to the SHA-224 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.
SHA256LG	Hash algorithm is similar to the SHA-256 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.
SHA384LG	Hash algorithm is similar to the SHA-384 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.
SHA512LG	Hash algorithm is similar to the SHA-512 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.
Chaining Flag (optional)	
FIRST	Specifies this is the first call in a series of chained calls. For hash method keywords other than the SHA3 family or SHAKE family, intermediate results are stored in the *hash* field.
LAST	Specifies this is the last call in a series of chained calls.
MIDDLE	Specifies this is a middle call in a series of chained calls. For hash method keywords other than the SHA3 family or SHAKE family, intermediate results are stored in the *hash* field.
ONLY	Specifies this is the only call and the call is not chained. This is the default.

text_length

Direction	**Type**
Input	Integer

The length of the *text* parameter in bytes. For more information, see Table 238 on page 584.

Note: If you specify the FIRST or MIDDLE keyword, then the text length must be a multiple of the blocksize of the hash method. For MD5, RPMD-160, SHA-1, SHA-224 and SHA-256, this is a multiple of 64 bytes. For SHA-384 and SHA-512, this is a multiple of 128 bytes.

For ONLY and LAST, this service performs the required padding according to the algorithm specified.

text

Direction	Type
Input	String

The application-supplied text on which this service performs the hash.

chaining_vector_length

Direction	Type
Input	Integer

The byte length of the *chaining_vector* parameter. For hash method keywords other than the SHA3 family or SHAKE family, this must be 128 bytes. For hash method keywords in the SHA3 family or SHAKE family, this must be 256 bytes.

chaining_vector

Direction	Type
Input/Output	String

This field is a 128-byte or 256-byte work area. Your application must not change the data in this string. The chaining vector permits chaining data from one call to another.

hash_length

Direction	Type
Input	Integer

The length of the *hash* field in bytes. See Table 238 on page 584 for the minimum lengths. For SHAKE128 and SHAKE256, the length can be any non-zero value. On ONLY or LAST, the length specified here is generated into the *hash* parameter.

Note: The length of the SHA-224 hash is 28 bytes and the length of the SHA-384 hash is 48 bytes. The extra bytes (4 bytes for SHA-224 and 16 bytes for SHA-384) are used during the generation of the hash value. The final hash value is left-justified and padded with zeroes.

hash

Direction	Type
Input/Output	String

This field contains the hash or extended-output-function (XOF) message digest, left-justified. For hash method keywords other than the SHA3 family or SHAKE family, if you specify the FIRST or MIDDLE keyword, this field contains the intermediate hash value. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message.

For SHAKE128 and SHAKE256, when ONLY or LAST is specified, the extended-output-function (XOF) message digest generates the number of bytes specified for *hash_length*.

text_id_in

Direction	Type
Input	Integer

For CSNBOWH1 only, the ALET for the data space containing the text for which to generate the hash.

Usage notes

Although some hashing methods allow it, bit length text is not supported for any hashing method.

If the CSF.CSFSERV.AUTH.CSFOWH.DISABLE SAF resource profile is defined in the XFACILIT SAF resource class, no SAF authorization checks will be performed against the CSFSERV class when using this service. If CSF.CSFSERV.AUTH.CSFOWH.DISABLE is not defined, the SAF authorization check will be performed. Disabling the SAF check may improve the performance of your application.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 240. One-Way Hash Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions	Keywords SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, and SHAKE256 are not supported.
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions	
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions	
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	

Symmetric MAC Generate (CSNBSMG or CSNBSMG1 and CSNESMG or CSNESMG1)

Use the symmetric MAC generate callable service to generate a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using an AES key.

The callable service names for AMODE(64) invocation are CSNESMG and CSNESMG1.

Choosing between CSNBSMG and CSNBSMG1 or CSNESMG and CSNESMG1

CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1 provide identical functions. When choosing which service to use, consider this:

- CSNBSMG and CSNESMG require the text to reside in the caller's primary address space. Also, a program using CSNBSMG adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSMG1 and CSNESMG1 allow the text to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using

CSNBSMG1 and CSNESMG1 do not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSMG1 and CSNESMG1, *text_id_in* is an access list entry token (ALET) parameter of the data spaces containing the text.

Format

```
CALL CSNBSMG(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_identifier_length,
             key_identifier,
             text_length,
             text,
             rule_array_count,
             rule_array,
             chaining_vector_length,
             chaining_vector,
             reserved_data_length,
             reserved_data,
             mac_length,
             mac )
```

```
CALL CSNBSMG1(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              key_identifier_length,
              key_identifier,
              text_length,
              text,
              rule_array_count,
              rule_array,
              chaining_vector_length,
              chaining_vector,
              reserved_data_length,
              reserved_data,
              mac_length,
              mac,
              text_id_in)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_identifier_length

Direction	Type
Input	String

The length of the *key_identifier* parameter. For the KEY-CLR keyword, the length is in bytes and includes only the value of the key length. The key length value can be 16, 24, or 32. For the KEYIDENT keyword, the length must be 64. For the KEY-DRV keyword, this is the length in bytes of the key material. The length can be from 16 to 32 bytes.

key_identifier

Direction	Type
Input	String

For the KEY-CLR keyword, this specifies the clear AES key. The parameter must be left justified. For the KEYIDENT keyword, this specifies an internal clear AES token or the label name of a clear AES key in the CKDS. Normal CKDS label name syntax is required. For the KEY-DRV keyword, this specifies the key material from which to derive the 128-bit AES key.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 2147483647 bytes. If the *text_length* is not a multiple of 8 bytes and if the ONLY or LAST keyword of the *rule_array* parameter is called, the text is padded in accordance with the processing rule specified.

text

Direction	Type
Input	String

The application-supplied text for which the MAC is generated.

rule_array_count

Direction	Type
Input	Integer

The number of keywords specified in the *rule_array* parameter. The value can be 1, 2, 3 or 4.

rule_array

Direction	Type
Input	Character String

This keyword provides control information to the callable service. The keywords must be eight bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

Table 241. Keywords for Symmetric MAC Generate control information

Keyword	Meaning
Algorithm (required)	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used.
MAC processing rule (optional)	
CBC-MAC	CBC MAC with padding for any key length. This is the default value.
XCBC-MAC	AES-XCBC-MAC-96 and AES-XCBC-PRF-128 MAC generation with padding for 128-bit keys.
Key rule (optional)	
KEY-CLR	This specifies that the key parameter contains a clear key value. This is the default value.
KEYIDENT	This specifies that the key_identifier field will be an internal clear token or the label name of a clear key in the CKDS. Normal CKDS label name syntax is required.
KEY-DRV	This specifies that the key parameter contains up to 256 bits of key material from which to derive a 128-bit AES key for the XCBC-MAC operation. Only valid with XCBC-MAC.
Segmenting Control (optional)	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

chaining_vector_length

Direction	Type
Input/Output	Integer

The length of the *chaining_vector* parameter. On output, the actual length of the chaining vector will be stored in the parameter.

chaining_vector

Direction	Type
Input/Output	String

This field is used as a system work area for the chaining vector. The chaining vector holds the output chaining vector from the caller.

The mapping of the *chaining_vector* depends on the algorithm specified. For AES, the *chaining_vector* field must be at least 36 bytes in length.

On calls that use ONLY or FIRST rules, initialize this parameter as binary zeros. On calls that use MIDDLE or LAST rules, your application program must not change the data in this string.

reserved_data_length

Direction	Type
Input	Integer

Reserved for future use. Value must be zero.

reserved_data

Direction	Type
Ignored	String

Reserved for future use.

mac_length

Direction	Type
Input	Integer

The length in bytes of the MAC to be returned in the mac field. The allowable values are 12 and 16 bytes.

mac

Direction	Type
Output	String

The 12-byte or 16-byte field in which the callable service returns the MAC value if the segmenting rule is ONLY or LAST.

text_id_in

Direction	Type
Input	Integer

For CSNBSMG1 and CSNESMG1 only, the ALET of the text for which the MAC is generated.

Usage notes

For the XCBC-MAC processing rule, *text_length* can be 0 when specifying the ONLY or LAST rule. However, LAST should be preceded by a call specifying FIRST or MIDDLE. A *text_length* of 0 is not allowed with any other MAC processing rules

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 242. Symmetric MAC Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions	
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions	
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions	
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	

Symmetric MAC Verify (CSNBSMV or CSNBSMV1 and CSNESMV or CSNESMV1)

Use the symmetric MAC verify callable service to verify a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using an AES key.

The callable service names for AMODE(64) invocation are CSNESMV and CSNESMV1.

Choosing between CSNBSMV and CSNBSMV1 or CSNESMV and CSNESMV1

CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1 provide identical functions. When choosing which service to use, consider this:

- CSNBSMV and CSNESMV require the text to reside in the caller's primary address space. Also, a program using CSNBSMV adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSMV1 and CSNESMV1 allow the text to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSMV1 and CSNESMV1 do not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSMV1 and CSNESMV1, *text_id_in* is an access list entry token (ALET) parameter of the data spaces containing the text.

Format

```
CALL CSNBSMV(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              key_identifier_length,
              key_identifier,
              text_length,
              text,
              rule_array_count,
              rule_array,
              chaining_vector_length,
              chaining_vector,
              reserved_data_length,
              reserved_data,
              mac_length,
              mac )
CALL CSNBSMV1(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              key_identifier_length,
              key_identifier,
              text_length,
              text,
              rule_array_count,
              rule_array,
              chaining_vector_length,
              chaining_vector,
              reserved_data_length,
              reserved_data,
              mac_length,
              mac,
              text_id_in )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter. For the KEY-CLR keyword, the length is in bytes and includes only the value of the key length. The key length value can be 16, 24, or 32. For the KEYIDENT keyword, the length must be 64. For the KEY-DRV keyword, this is the length in bytes of the key material. The length can be from 16 to 32 bytes.

key_identifier

Direction	Type
Input	String

For the KEY-CLR keyword, this specifies the clear AES key. The parameter must be left justified. For the KEYIDENT keyword, this specifies an internal clear AES token or the label name of a clear AES key in the CKDS. Normal CKDS label name syntax is required. For the KEY-DRV keyword, this specifies the key material from which to derive the 128-bit AES key.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 2147483647 bytes. If the *text_length* parameter is not a multiple of 8 bytes and if the ONLY or LAST keyword of the *rule_array* parameter is called, the text is padded in accordance with the processing rule specified.

text

Direction	Type
Input	String

The application-supplied text for which the MAC is verified.

rule_array_count

Direction	Type
Input	Integer

The number of keywords specified in the *rule_array* parameter. The value can be 1, 2, 3 or 4.

rule_array

Direction	Type
Input	String

This keyword provides control information to the callable service. The keywords must be eight bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

The order of the *rule_array* keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

Table 243. Keywords for Symmetric MAC Verify control information

Keyword	Meaning
Algorithm (required)	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used.
MAC processing rule (optional)	
CBC-MAC	CBC MAC with padding for any key length. This is the default value.
XCBC-MAC	AES-XCBC-MAC-96 and AES-XCBC-PRF-128 MAC generation with padding for 128-bit keys.
Key rule (optional)	
KEY-CLR	This specifies that the key parameter contains a clear key value. This is the default value.
KEYIDENT	This specifies that the key_identifier field will be an internal clear token or the label name of a clear key in the CKDS. Normal CKDS label name syntax is required.
KEY-DRV	This specifies that the key parameter contains up to 256 bits of key material from which to derive a 128-bit AES key for the XCBC-MAC operation. Only valid with XCBC-MAC.
Segmenting Control (optional)	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

chaining_vector_length

Direction	Type
Input/Output	String

The length of the *chaining_vector* parameter. On output, the actual length of the chaining vector will be stored in the parameter.

chaining_vector

Direction	Type
Input/Output	String

This field is used as a system work area for the chaining vector. The chaining vector holds the output chaining vector from the caller.

The mapping of the *chaining_vector* depends on the algorithm specified. For AES, the *chaining_vector* field must be at least 36 bytes in length.

On calls that use ONLY or FIRST rules, initialize this parameter as binary zeros. On calls that use MIDDLE or LAST rules, your application program must not change the data in this string.

reserved_data_length

Direction	Type
Input	Integer

Reserved for future use. Value must be zero.

reserved_data

Direction	Type
Ignored	String

Reserved for future use.

mac_length

Direction	Type
Input	Integer

The length in bytes of the MAC to be verified in the *mac* field. The allowable values are 12 and 16 bytes.

mac

Direction	Type
Input	String

The 12-byte or 16-byte field that contains the MAC value you want to verify. The value must be left-justified and padded with zeros.

text_id_in

Direction	Type
Input	Integer

For CSNBSMV1 and CSNESMV1 only, the ALET of the text for which the MAC is to be verified.

Usage notes

For the XCBC-MAC processing rule, *text_length* can be 0 when specifying the ONLY or LAST rule. However, LAST should be preceded by a call specifying FIRST or MIDDLE. A *text_length* of 0 is not allowed with any other MAC processing rules

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 244. Symmetric MAC Verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions	
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions	
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions	
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions	

Chapter 8. Financial services

The process of validating personal identities in a financial transaction system is called *personal authentication*. The personal identification number (PIN) is the basis for verifying the identity of a customer across financial industry networks. ICSF provides callable services to translate, verify, and generate PINs. You can use the callable services to prevent unauthorized disclosures when organizations handle PINs.

These callable services are described in these topics:

- "Authentication Parameter Generate (CSNBAPG and CSNEAPG)" on page 620
- "Clear PIN Encrypt (CSNBCPE and CSNECPE)" on page 624
- "Clear PIN Generate (CSNBPGN and CSNEPGN)" on page 629
- "Clear PIN Generate Alternate (CSNBCPA and CSNECPA)" on page 634
- "CVV Key Combine (CSNBCKC and CSNECKC)" on page 640
- "EMV Scripting Service (CSNBESC and CSNEESC)" on page 646
- "EMV Transaction (ARQC/ARPC) Service (CSNBEAC and CSNEEAC)" on page 657
- "EMV Verification Functions (CSNBEVF and CSNEEVF)" on page 665
- "Encrypted PIN Generate (CSNBEPG and CSNEEPG)" on page 671
- "Encrypted PIN Translate (CSNBPTR and CSNEPTR)" on page 678
- "Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)" on page 685
- "Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)" on page 701
- "Encrypted PIN Verify (CSNBPVR and CSNEPVR)" on page 713
- "Encrypted PIN Verify2 (CSNBPVR2 and CSNEPVR2)" on page 720
- "Field Level Decipher (CSNBFLD and CSNEFLD)" on page 730.
- "Field Level Encipher (CSNBFLE and CSNEFLE)" on page 739.
- "Format Preserving Algorithms Decipher (CSNBFFXD and CSNEFFXD)" on page 749.
- "Format Preserving Algorithms Encipher (CSNBFFXE and CSNEFFXE)" on page 754.
- "Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT)" on page 760.
- "FPE Decipher (CSNBFPED and CSNEFPED)" on page 767.
- "FPE Encipher (CSNBFPEE and CSNEFPEE)" on page 776.
- "FPE Translate (CSNBFPET and CSNEFPET)" on page 786.
- "PIN Change/Unblock (CSNBPCU and CSNEPCU)" on page 795
- "Recover PIN from Offset (CSNBPFO and CSNEPFO)" on page 806
- "Secure Messaging for Keys (CSNBSKY and CSNESKY)" on page 811
- "Secure Messaging for PINs (CSNBSPN and CSNESPN)" on page 816
- "SET Block Compose (CSNDSBC and CSNFSBC)" on page 823
- "SET Block Decompose (CSNDSBD and CSNFSBD)" on page 828
- "Transaction Validation (CSNBTRV and CSNETRV)" on page 834
- "VISA CVV Service Generate (CSNBCSG and CSNECSG)" on page 838
- "VISA CVV Service Verify (CSNBCSV and CSNECSV)" on page 843

How Personal Identification Numbers (PINs) are used

Many people are familiar with PINs, which allow them to use an automated teller machine (ATM). From the system point of view, PINs are used primarily in financial networks to authenticate users — typically, a user is assigned a PIN, and enters the PIN at automated teller machines (ATMs) to gain access to his or her accounts. It is extremely important that the PIN be kept private, so that no one other than the account owner can use it. ICSF allows your applications to generate PINs, to verify supplied PINs, and to translate PINs from one format to another.

How VISA card verification values are used

The Visa International Service Association (VISA) and MasterCard International, Incorporated, have specified a cryptographic method to calculate a value that relates to the personal account number (PAN), the card expiration date, and the service code. The VISA card-verification value (CVV) and the MasterCard card-verification code (CVC) can be encoded on either track 1 or track 2 of a magnetic striped card and are used to detect forged cards. Because most online transactions use track-2, the ICSF callable services generate and verify the CVV[3] by the track-2 method.

The VISA CVV service generate callable service calculates a 1- to 5-byte value through the DES-encryption of the PAN, the card expiration date, and the service code using two data-encrypting keys or two MAC keys. The VISA CVV service verify callable service calculates the CVV by the same method, compares it to the CVV supplied by the application (which reads the credit card's magnetic stripe) in the *CVV_value*, and issues a return code that indicates whether the card is authentic.

Translating data and PINs in networks

More and more data is being transmitted across networks where, for various reasons, the keys used on one network cannot be used on another network. Encrypted data and PINs that are transmitted across these boundaries must be "translated" securely from encryption under one key to encryption under another key. For example, a traveler visiting a foreign city might wish to use an ATM to access an account at home. The PIN entered at the ATM might need to be encrypted at the ATM and sent over one or more financial networks to the traveler's home bank. At the home bank, the PIN must be verified prior to access being allowed. On intermediate systems (between networks), applications can use the Encrypted PIN translate callable service to re-encrypt a PIN block from one key to another. Running on ICSF, such applications can ensure that PINs never appear in the clear and that the PIN-encrypting keys are isolated on their own networks.

Working with Europay–MasterCard–Visa smart cards

There are several services you can use in secure communications with EMV smart cards. The processing capabilities are consistent with the specifications provided in these documents:

- *EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2*
- *Design Visa Integrated Circuit Card Specification Manual*
- *Integrated Circuit Card Specification (VIS) 1.4.0 Corrections*

EMV smart cards include the following processing capabilities:

- The diversified key generate (CSNBDKG and CSNEDKG) callable service with rule-array options **TDES-XOR**, **TDESEMV2**, and **TDESEMV4** enables you to derive a key used to cipher and authenticate messages, and more particularly message parts, for exchange with an EMV smart card. You use the derived key with services such as encipher, decipher, MAC generate, MAC verify, secure messaging for keys, and secure messaging for PINs. These message parts can be combined with message parts created using the secure messaging for keys and secure messaging for PINs services.

[3] The VISA CVV and the MasterCard CVC refer to the same value. CVV is used here to mean both CVV and CVC.

- The secure messaging for keys (CSNBSKY and CSNESKY) service enables you to securely incorporate a key into a message part (generally the value portion of a TLV component of a secure message for a card). Similarly, the secure messaging for PINs (CSNBSPN and CSNESPN) service enables secure incorporation of a PIN block into a message part.
- The PIN change/unblock (CSNBPCU and CSNEPCU) service enables you to encrypt a new PIN to send to a new EMV card, or to update the PIN value on an initialized EMV card. This verb generates both the required session key (from the master encryption key) and the required authentication code (from the master authentication key).
- The **ZERO-PAD** option of the PKA encrypt (CSNDPKE) service enables you to validate a digital signature created according to ISO 9796-2 standard by encrypting information you format, including a hash value of the message to be validated. You compare the resulting enciphered data to the digital signature accompanying the message to be validated.
- The MAC generate and MAC verify services post-pad a X'80'...X'00' string to a message as required for authenticating messages exchanged with EMV smart cards.

PIN callable services

You use the PIN callable services to generate, verify, and translate PINs. This topic discusses the PIN callable services, as well as the various PIN algorithms and PIN block formats supported by ICSF. It also explains the use of PIN-encrypting keys.

Generating a PIN

To generate personal identification numbers, call the Clear PIN Generate or Encrypted PIN Generate callable service. Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the Clear PIN generate callable service generates a clear PIN and a PIN verification value, or offset. The Clear PIN Generate callable service can only execute in special secure mode. For a description of this mode, see “Special secure mode” on page 10. Using a PIN generation algorithm, data used in the algorithm, the PIN generation key, and an outbound PIN encrypting key, the encrypted PIN generate callable service generates and formats a PIN and encrypts the PIN block.

Encrypting a PIN

To format a PIN into a supported PIN block format and encrypt the PIN block, call the Clear PIN encrypt callable service.

Generating a PIN Validation Value (PVV) from an encrypted PIN block

To generate a clear VISA PIN validation value (PVV) from an encrypted PIN block, call the *clear PIN generate alternate* callable service. The PIN block can be encrypted under an input PIN-encrypting key (IPINENC) or an output PIN encrypting key (OPINENC).

Verifying a PIN

To verify a supplied PIN, call the *Encrypted PIN Verify* or the *Encrypted PIN Verify2* callable service.

For the Encrypted PIN Verify service
You supply the enciphered PIN, the PIN-encrypting key that enciphers the PIN, and other data. You must also specify the PIN verification key and PIN verification algorithm. The callable service generates a verification PIN. The service compares the two personal identification numbers and if they are the same, it verifies the supplied PIN.

For the Encrypted PIN Verify2 service
You supply the enciphered PIN block, the enciphered reference PIN block, the PIN-encrypting keys that enciphers the PINs, and other data. The service decrypts the PIN blocks and compares the two personal identification numbers and if they are the same, it verifies the supplied PIN.

Translating a PIN

To translate a PIN block format from one PIN-encrypting key to another or from one PIN block format to another, call the *Encrypted PIN translate*, *Encrypted PIN translate enhanced,* or *Encrypted PIN translate2* callable services. You must identify the input PIN-encrypting key that originally enciphered the PIN. You also need to specify the output PIN-encrypting key that you want the callable service to use to encipher the PIN. If you want to change the PIN block format, specify a different output PIN block format from the input PIN block format.

The *Encrypted PIN translate enhanced* service handles PIN blocks where the PAN field is enciphered using format preserving encryption.

The *Encrypted PIN translate2* service has the same functionality of the *Encrypted PIN translate* service with additional support for ISO-4 PIN blocks.

Table 245. Valid translation rules

From > To	3621	3624	4704	ECI2	ECI3	VIS2	VIS3	VIS4	ISO0	ISO1	ISO2	ISO3	ISO4
3621	V	V	V	V	V	V	V	V	V	V	N	V	V
3624	V	V	V	V	V	V	V	V	V	V	N	V	V
4704	V	V	V	V	V	V	V	V	V	V	N	V	V
ECI2	V	V	V	V	V	V	V	V	V	V	N	V	V
ECI3	V	V	V	V	V	V	V	V	V	V	N	V	V
VIS2	V	V	V	V	V	V	V	V	V	V	N	V	V
VIS3	V	V	V	V	V	V	V	V	V	V	N	V	V
VIS4	N	N	N	N	N	N	N	V	V	N	N	V	V
ISO0	N	N	N	N	N	N	N	N	V	N	N	V	V
ISO1	N	N	N	N	N	N	N	N	V	V	N	V	V
ISO2	N	N	N	N	N	N	N	N	N	N	N	N	N
ISO3	N	N	N	N	N	N	N	N	V	N	N	V	V
ISO4	N	N	N	N	N	N	N	N	V	N	N	V	V

V
Valid translation between these two PIN block formats.

N
Invalid translation between these two PIN block formats.

Algorithms for generating and verifying a PIN

ICSF supports these algorithms for generating and verifying personal identification numbers:

- IBM 3624 institution-assigned PIN
- IBM 3624 customer-selected PIN (through a PIN offset)
- IBM German Bank Pool PIN (verify through an institution key)
- VISA PIN through a VISA PIN validation value
- Interbank PIN

The algorithms are discussed in detail in "PIN formats and algorithms" on page 1631.

Using PINs on different systems

ICSF allows you to translate different PIN block formats, which lets you use personal identification numbers on different systems. ICSF supports these formats:

- IBM 3624
- IBM 3621 (same as IBM 5906)
- IBM 4704 encrypting PINPAD format
- ISO 0 (same as ANSI 9.8, VISA 1, and ECI 1)
- ISO 1 (same as ECI 4)
- ISO 2
- ISO 3
- ISO 4
- VISA 2
- VISA 3
- VISA 4
- ECI 2
- ECI 3

The formats are discussed in "PIN formats and algorithms" on page 1631.

PIN-encrypting keys

There are inbound and output PIN-encrypting keys. Both DES and AES keys are supported. For further key separation, an installation can choose to have each PIN block format enciphered under a different PIN-encrypting key.

For more information about PIN-encrypting keys, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Derived unique key per transaction algorithms

ICSF supports ANSI X9.24 derived unique key per transaction algorithms to generate AES and DES PIN-encrypting keys from user data. For DES, ICSF supports both single-length and double-length key generation. Keywords for single-length DES and double-length DES key generation cannot be mixed.

Encrypted PIN Translate and Encrypted PIN Translate2

The UKPTIPIN, UKPTOPIN and UKPTBOTH keywords will cause the services to generate single-length DES keys. DUKPT-IP, DUKPT-OP and DUKPT-BH are the respective keywords to generate double-length DES keys. The *input_PIN_profile* and *output_PIN_profile* must supply the current key serial number when these keywords are specified.

The ADUKPTBH, ADUKPTIP, and ADUKPTOP keywords will cause the services to generate AES keys. The *input_PIN_profile* and *output_PIN_profile* must supply the PIN-block format and single block of derivation data extension when these keywords are specified.

Encrypted PIN Translate Enhanced

The DUKPT-IP, DUKPT-OP and DUKPT-BH keywords will cause the service to generate double-length DES keys. The *input_PIN_profile* and *output_PIN_profile* must supply the current key serial number when these keywords are specified.

The ADUKPTBH, ADUKPTIP, and ADUKPTOP keywords will cause the services to generate AES keys. The *input_PIN_profile* and *output_PIN_profile* must supply the PIN-block format and single block of derivation data extension when these keywords are specified.

Encrypted PIN Verify

The UKPTIPIN keyword will cause the service to generate DES single-length keys. DUKPT-IP is the keyword for double-length DES key generation. The *input_PIN_profile* must supply the current key serial number when these keywords are specified.

The ADUKPTIP keyword will cause the services to generate AES keys. The *input_PIN_profile* must supply the PIN-block format and single block of derivation data extension when the keyword is specified.

Encrypted PIN Verify2

The UKPT keyword will cause the service to generate DES single-length keys. DUKPT is the keyword for double-length DES key generation. The *input_PIN_profile* and *reference_PIN_profile* must supply the current key serial number when these keywords are specified.

The ADUKPT keyword will cause the services to generate AES keys. The *input_PIN_profile* and *reference_PIN_profile* must supply the PIN-block format and single block of derivation data extension when the keyword is specified.

For more information

For more information about PIN-encrypting keys, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

ANSI X9.8 PIN restrictions

Access control points (ACP) in the domain role control PIN block processing restrictions from the X9.8 standard. These access control points are available on IBM z196, IBM z114, or later servers. These callable services are affected by these access control points. These access control points are disabled by default in the domain role. A TKE Workstation is required to enable these ACPs.

- Clear PIN Generate Alternate (CSNBCPA and CSNECPA)
- Encrypted PIN Generate (CSNBEPG and CSNEEPG)
- Encrypted PIN Translate (CSNBPTR and CSNEPTR)
- Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)
- Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)
- Encrypted PIN Verify (CSNBPVR and CSNEPVR)
- Secure Messaging for PINs (CSNBSPN and CSNESPN)

There are four access control points:

- ANSI X9.8 PIN - Enforce PIN block restrictions
- ANSI X9.8 PIN - Allow modification of PAN
- ANSI X9.8 PIN - Allow only ANSI PIN blocks
- ANSI X9.8 PIN - Use stored decimalization tables only

PIN decimalization tables can be stored in the CCA cryptographic coprocessor that is a CEX3C or later for use by callable services. Only tables that have been activated can be used. A TKE Workstation is required to manage the tables in the coprocessors.

Note: ICSF routes work to all active coprocessors based on work load. All coprocessors must have the same set of active decimalization tables for the **ANSI X9.8 PIN - Use stored decimalization tables only** access control point to be effective.

ANSI X9.8 PIN - Enforce PIN block restrictions

When **ANSI X9.8 PIN - Enforce PIN block restrictions** access control point is enabled, the following restrictions will be enforced.

- CSNBPTR, CSNBPTR2, CSNBPTRE, CSNBPFO, and CSNBSPN will not accept IBM 3624 PIN format in the output profile parameter when the input profile parameter is not IBM 3624.
- CSNBPTR, CSNBPTR2, and CSNBPTRE will not accept ISO-0 or ISO-3 formats in the input PIN profile unless ISO-0 or ISO-3 is in the output PIN profile.
- CSNBPTR, CSNBPTR2, CSNBPTRE, CSNBPFO, and CSNBSPN will not accept ISO-1 or ISO-2 formats in the output profile parameter when the input profile parameter contains ISO-0, ISO-3, or VISA4
- When the input profile parameter of CSNBPTR, CSNBPTR2, CSNBPTRE, CSNBPFO, or CSNBSPN contains either ISO-0 or ISO-3 formats, the decrypted PIN block will be examined to ensure that the PAN within the PIN block is the same as the PAN which was supplied as the input PAN parameter, and that this is the same as the PAN which was supplied as the output PAN parameter.
- The input PAN and output PAN parameters of CSNBPTR or CSNBSPN must be equivalent.
- When the rule array for CSNBCPA contains VISA-PVV, the input PIN profile must contain ISO-0 or ISO-3 formats.

ANSI X9.8 PIN - Allow modification of PAN

In order to enable the **ANSI X9.8 PIN - Allow modification of PAN** access control point, the **ANSI X9.8 PIN - Enforce PIN block restrictions** must also be enabled. The **ANSI X9.8 PIN - Allow modification of PAN** access control point cannot be enabled by itself.

When the **ANSI X9.8 PIN - Allow modification of PAN** access control point is enabled, the input PAN and output PAN parameters will be tested in CSNBPTR, CSNBPTR2, or CSNBSPN. The input PAN will be compared to the portions of the PAN which are recoverable from the decrypted PIN block. If the PANs compare, then the account number will be changed in the output PIN block.

ANSI X9.8 PIN - Allow only ANSI PIN blocks

In order to enable the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** access control point, the **ANSI X9.8 PIN - Enforce PIN block restrictions** must also be enabled. The **ANSI X9.8 PIN - Allow only ANSI PIN blocks** access control point cannot be enabled by itself.

When this access control point is enabled, CSNBPTR, CSNBPTR2, and CSNBPTRE will allow reformatting of the PIN block as shown in the following table.

Table 246. ANSI X9.8 PIN - Allow only ANSI PIN blocks

From	Reformat to ISO Format 0, 3, 4	Reformat to ISO Format 1	Reformat to ISO Format 2
ISO Format 0, 3, 4	Reformat permitted without change of PAN. Change of PAN only permitted in sensitive state for card issuance.	Not permitted.	Permitted for submission to an IC card.
ISO Format 1	Reformat permitted.	Reformat permitted.	Permitted for submission to an IC card.
ISO Format 2	Not permitted.	Not permitted.	Permitted for submission to an IC card.

ANSI X9.8 PIN – Use stored decimalization tables only

The ANSI X9.8 PIN – Use stored decimalization tables only access control point may be enabled by itself.

When this access control point is enabled, CSNBPGN, CSNBCPA, CSNBEPG, and CSNBPVR services must supply a decimalization table that matches the active decimalization tables stored in the coprocessors. The decimalization table in the *data_array* parameter will be compared against the active decimalization

tables in the coprocessor and if the supplied table matches a stored table, the request will be processed. If the supplied table does not match any of the stored tables or there are no stored tables, the request will fail.

PIN decimalization tables can be stored in the CEX3C or later coprocessors for use by callable services. Only tables that have been activated can be used. A TKE Workstation is required to manage the tables in the coprocessors.

Note: ICSF routes work to all active coprocessors based on work load. All coprocessors must have the same set of decimalization tables for the decimalization table access control point to be effective.

Enhanced PIN Security

Enhanced PIN security mode

An Enhanced PIN Security Mode is available. This optional mode is selected by enabling the **Enhanced PIN Security** access control point in coprocessor domain role. When active, this control point affects all PIN callable services that extract or format a PIN using a PIN-block format of 3621 or 3624 with a PIN-extraction method of PADDIGIT.

Table 247 on page 608 summarizes the callable services affected by the Enhanced PIN Security Mode and describes the effect that the mode has when the access control point is enabled.

Table 247. Callable services affected by enhanced PIN security mode

PIN-block format and PIN-extraction method	Callable services affected	PIN processing changes when Enhanced PIN Security Mode enabled
ECI-2, 3621, or 3624 formats AND PINLENxx	CSNBCPA CSNBPTR CSNBPTRE CSNBPVR CSNBPVR2	The PINLENxx keyword in rule_array parameter for PIN extraction method is not allowed if the Enhanced PIN Security Mode is enabled. **Note:** The services will fail with return code 8 reason code '7E0'x.
3621 or 3624 format and PADDIGIT	CSNBCPA CSNBPTR CSNBPTR2 CSNBPTRE CSNBPVR CSNBPVR2 CSNBPCU	PIN extraction determines the PIN length by scanning from right to left until a digit, not equal to the pad digit, is found. The minimum PIN length is set at four digits, so scanning ceases one digit past the position of the 4th PIN digit in the block.
3621 or 3624 format and PADDIGIT	CSNBCPE CSNBEPG CSNBPTR CSNBPTR2 CSNBPTRE	PIN formatting does not examine the PIN, in the output PIN block, to see if it contains the pad digit.

Table 247. Callable services affected by enhanced PIN security mode (continued)		
PIN-block format and PIN-extraction method	**Callable services affected**	**PIN processing changes when Enhanced PIN Security Mode enabled**
3621 or 3624 format and PADDIGIT	CSNBPTR CSNBPTR2 CSNBPTRE	Restricted to non-decimal digit for PAD digit.

Enhanced PIN checking for CSNBPTR and CSNBPTR2

For the PIN translate services, additional checking is available when the TRANSLAT rule array keyword is specified. When the **Encrypted PIN Translate - Translate PIN Check** access control is enabled, checking of the PIN block is performed. The checking is similar to the checking done when the REFORMAT keyword is specified.

PIN block error processing mode

To prevent the abuse of PIN processing error messages due to information leakage derived from the return code reason codes returned under various conditions, the PIN checking errors will be replaced with a general ISO format error. This optional mode is selected by enabling the **General ISO PIN Error Mode** access control in the coprocessor domain role. Services affected are:

- Encrypted PIN Translate (CSNBPTR and CSNEPTR).
- Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2).
- DK PIN Change (CSNBDPC and CSNEDPC).
- DK PIN Verify (CSNBDPV and CSNEDPV).

Return code 8 reason code 2514 (9D2) will be issued instead of the following return code 8 reason codes: 100, 106, 110, 108, 407, 3004, and 3016.

The PIN profile

The PIN profile consists of:

- PIN block format (see “PIN block format” on page 610)
- Format control (see “Format control” on page 612)
- Pad digit (see “Pad digit” on page 612)

The PIN profile can optionally contain one of the following:

- Current Key Serial Number (for DES-DUKPT, see “Current key serial number” on page 613).
- AES-DUKPT Derivation Data (see Table 684 on page 1719.

Table 248 on page 609 shows the format of a PIN profile.

Table 248. Format of a PIN profile	
Bytes	**Description**
0–7	PIN block format.
8-15	Format control.
16–23	Pad digit.
The layout of the next section depends on the DUKPT algorithm used.	
DES-DUKPT	

Table 248. Format of a PIN profile (continued)	
Bytes	**Description**
24–47	Current key serial number.
AES-DUKPT	
24–43	AES-DUKPT derivation data.

PIN block format

This keyword specifies the format of the PIN block. The 8-byte or 16-byte value must be left-justified and padded with blanks. Refer to Table 249 on page 610 for a list of valid values.

Table 249. Format values of PIN blocks	
Format value	**Description**
ECI-2	Eurocheque International format 2
ECI-3	Eurocheque International format 3
ISO-0	ISO format 0, ANSI X9.8, VISA 1, and ECI 1
ISO-1	ISO format 1 and ECI 4
ISO-2	ISO format 2
ISO-3	ISO format 3
ISO-4	ISO format 4
VISA-2	VISA format 2
VISA-3	VISA format 3
VISA-4	VISA format 4
3621	IBM 3621 and 5906
3624	IBM 3624
4704-EPP	IBM 4704 with encrypting PIN pad

PIN block format and PIN extraction method keywords

In the Clear PIN Generate Alternate, Encrypted PIN Translate and Encrypted PIN Verify callable services, you may specify a PIN extraction keyword for a given PIN block format. In this table, the allowable PIN extraction methods are listed for each PIN block format. The first PIN extraction method keyword listed for a PIN block format is the default.

Table 250. PIN block format and PIN extraction method keywords		
PIN Block Format	**PIN Extraction Method Keywords**	**Description**
ECI-2	PINLEN04	The PIN extraction method keywords specify a PIN extraction method for a PINLEN04 format.
ECI-3	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.

Table 250. PIN block format and PIN extraction method keywords (continued)

PIN Block Format	PIN Extraction Method Keywords	Description
ISO-0	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
ISO-1	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
ISO-2	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
ISO-3	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
ISO-4	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
VISA-2	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
VISA-3	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
VISA-4	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
3621	PADDIGIT, HEXDIGIT, PINLEN04 to PINLEN12, PADEXIST	The PIN extraction method keywords specify a PIN extraction method for an IBM 3621 PIN block format. The first keyword, PADDIGIT, is the default PIN extraction method for the PIN block format.
3624	PADDIGIT, HEXDIGIT, PINLEN04 to PINLEN16, PADEXIST	The PIN extraction method keywords specify a PIN extraction method for an IBM 3624 PIN block format. The first keyword, PADDIGIT, is the default PIN extraction method for the PIN block format.

Table 250. PIN block format and PIN extraction method keywords (continued)		
PIN Block Format	**PIN Extraction Method Keywords**	**Description**
4704-EPP	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.

The PIN extraction methods operate as follows:

PINBLOCK
Specifies that the service use one of these:

- the PIN length, if the PIN block contains a PIN length field
- the PIN delimiter character, if the PIN block contains a PIN delimiter character.

PADDIGIT
Specifies that the service use the pad value in the PIN profile to identify the end of the PIN.

HEXDIGIT
Specifies that the service use the first occurrence of a digit in the range from X'A' to X'F' as the pad value to determine the PIN length.

PINLEN*xx*
Specifies that the service use the length specified in the keyword, where xx can range from 4 to 16 digits, to identify the PIN.

PADEXIST
Specifies that the service use the character in the 16th position of the PIN block as the value of the pad value.

Format control

This keyword specifies whether there is any control on the user-supplied PIN format. The 8-byte value must be left-justified and padded with blanks. None is the only supported format control.

NONE
No format control.

Pad digit

Some PIN formats require this parameter. If the PIN format does not need a pad digit, the callable service ignores this parameter. Table 251 on page 612 shows the format of a pad digit. The PIN profile pad digit must be specified in uppercase.

Table 251. Format of a pad digit	
Bytes	**Description**
16–22	Seven space characters
23	Character representation of a hexadecimal pad digit or a space if a pad digit is not needed. Characters must be one of these: 0–9, A–F, or a blank.

Each PIN format supports only a pad digit in a certain range. This table lists the valid pad digits for each PIN block format.

Table 252. Pad digits for PIN block formats		
PIN Block Format	**Output PIN Profile**	**Input PIN Profile**
ECI-2	Pad digit is not used	Pad digit is not used
ECI-3	Pad digit is not used	Pad digit is not used
ISO-0	F	Pad digit is not used
ISO-1	Pad digit is not used	Pad digit is not used
ISO-2	Pad digit is not used	Pad digit is not used
ISO-3	Pad digit is not used	Pad digit is not used
ISO-4	Pad digit is not used	Pad digit is not used
VISA-2	0 through 9	Pad digit is not used
VISA-3	0 through F	Pad digit is not used
VISA-4	F	Pad digit is not used
3621	0 through F	0 through F
3624	0 through F	0 through F
4704-EPP	F	Pad digit is not used

The callable service returns an error that indicates that the pad digit is not valid if all of these conditions are met:

- The PTR Enhanced Security access control point is enabled in the active role.
- The output PIN profile specifies 3621 or 3624 as the PIN-block format.
- The output PIN profile specifies a decimal digit (0-9) as the pad digit.

Recommendations for the pad digit

IBM recommends that you use a non-decimal pad digit in the range of A through F when processing IBM 3624 and IBM 3621 PIN blocks. If you use a decimal pad digit, the creator of the PIN block must ensure that the calculated PIN does not contain the pad digit, or unpredictable results may occur.

For example, you can exclude a specific decimal digit from being in any calculated PIN by using the IBM 3624 calculation procedure and by specifying a decimalization table that does not contain the desired decimal pad digit.

Current key serial number

The current key serial number is the concatenation of the initial key serial number (a 59-bit value) and the encryption counter (a 21-bit value). The concatenation is an 80-bit (10-byte) value. Table 253 on page 613 shows the format of the current key serial number.

When DUKPT is specified, the PIN profile parameter is extended to a 48-byte field and must contain the current key serial number.

Table 253. Format of the current key serial number field	
Bytes	**Description**
24–47	Character representation of the current key serial number used to derive the initial PIN encrypting key. It is left justified and padded with 4 blanks.

Decimalization tables

Decimalization tables can be loaded in the coprocessors to restrict attacks using modified tables. The management of the tables requires a TKE Workstation.

Clear PIN Generate (CSNBPGN and CSNEPGN), Clear PIN Generate Alternate (CSNBCPA and CSNECPA), Encrypted PIN Generate (CSNBEPG and CSNEEPG), and Encrypted PIN Verify (CSNBPVR and CSNEPVR) callable services will make use of the stored decimalization tables.

The **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is used to restrict the use of tables. When the access control point is enabled, the table supplied by the callable service will be compared against the active tables stored in the coprocessor. If the supplied table does not match any of the active tables, the request will fail.

A TKE workstation (Version 7.1 or later) is required to manage the PIN decimalization tables. The tables must be loaded and then activated. Only active tables are checked when the access control point is enabled.

Note: ICSF routes work to all active coprocessors based on work load. All coprocessors must have the same set of decimalization tables for the decimalization table access control point to be effective.

Format preserving encryption

Format preserving encryption (FPE) is a method of encryption where the resulting cipher text has the same form as the input clear text. The form of the text can vary according to use and the application. One example is a 16 digit credit card number. After using FPE to encrypt a credit card number, the resulting cipher text is another 16 digit number. In this example of the credit card number, the output cipher text is limited to numeric digits only.

The CSNBFPEE, CSNBFPED, CSNBFPET, and CSNBPTRE callable services implement the VISA Format Preserving Encryption algorithm, which is a counter mode stream cipher.

The CSNBFFXD, CSNBFFXE, and CSNBFFXT callable services implement the NIST FFX algorithms. The FF1, FF2, and FF2.1 algorithms are all built in a similar way, using AES as the base cipher for the operations. The overall algorithm uses a Pseudorandom Function (PRF) as its main encryption function using a variable length Feistal network. Each of the three algorithms contain a different PRF to achieve the result. Each algorithm also takes in a tweak string to further vary the action of the PRF. FF1 uses either a 128-bit AES key or a 256-bit AES key. FF2 and FF2.1 only support AES 128-bit keys.

The FPE services require some knowledge of the input clear text character set in order to create the appropriate output ciphertext. The CSNBFPEE, CSNBFPED, CSNBFPET, and CSNBPTRE callable services use the following tables to determine valid character sets for the clear text input parameters:

Base-10 alphabet

This alphabet is used when the character set only consists of numbers 0 through 9. The original data type of the source field may be of any type. This alphabet requires the following values to be used in the VFPE algorithm:

```
Number of characters in alphabet('n'): 10
```

Table 254. Base-10 alphabet

FPE alphabet number	Character	ISO 7811 modified 5-bit ASCII	ISO 7811 modified 7-bit ASCII	Normal data type encoding		
				4-bit binary coded decimal	7-bit ASCII	8-bit EBCDIC
0	0	10000	0010000	0000	0110000	11110000
1	1	00001	1010001	0001	0110001	11110001

Table 254. Base-10 alphabet (continued)						
FPE alphabet number	**Character**	**ISO 7811 modified 5-bit ASCII**	**ISO 7811 modified 7-bit ASCII**	**Normal data type encoding**		
				4-bit binary coded decimal	**7-bit ASCII**	**8-bit EBCDIC**
2	2	00010	1010010	0010	0110010	11110010
3	3	10011	0010011	0011	0110011	11110011
4	4	00100	1010100	0100	0110100	11110100
5	5	10101	0010101	0101	0110101	11110101
6	6	10110	0010110	0110	0110110	11110110
7	7	00111	1010111	0111	0110111	11110111
8	8	01000	1011000	1000	0111000	11111000
9	9	11001	0011001	1001	0111001	11111001

FPE base-15 alphabet

Cards are encoded with the special ISO 7811 modified 5-bit ASCII encoding for track 2. This data type allows parity checking of the digits. Many systems require this encoding to be converted into standard data types for processing. Other data fields may use FPE base-15 encoding and would use this same alphabet when performing VFPE. These data types support values of 0 through 9 and A through F.

VFPE requires translation of the characters of the FPE alphabet number prior to encryption. Therefore, any of the data types shown in Table 255 on page 615 are supported. Decryption may use the same or a different data type than the original encoding. This alphabet requires the following values to be used in the VFPE algorithm:

```
Number of characters in alphabet('n'): 15
```

Table 255. FPE base-15 alphabet						
FPE alphabet number	**ISO 7811 modified 5-bit ASCII encoding**		**Normal data type encoding**			
	Character	**Binary**	**Character**	**4-bit binary coded decimal**	**7-bit ASCII**	**8-bit EBCDIC**
0	0	10000	0	0000	0110000	11110000
1	1	00001	1	0001	0110001	11110001
2	2	00010	2	0010	0110010	11110010
3	3	10011	3	0011	0110011	11110011
4	4	00100	4	0100	0110100	11110100
5	5	10101	5	0101	0110101	11110101
6	6	10110	6	0110	0110110	11110110
7	7	00111	7	0111	0110111	11110111
8	8	01000	8	1000	0111000	11111000
9	9	11001	9	1001	0111001	11111001

Table 255. FPE base-15 alphabet (continued)						
FPE alphabet number	**ISO 7811 modified 5-bit ASCII encoding**		**Normal data type encoding**			
	Character	**Binary**	**Character**	**4-bit binary coded decimal**	**7-bit ASCII**	**8-bit EBCDIC**
10	:	11010	A	1010	1000001	11000001
11	;	01011	B	1011	1000010	11000010
12	<	11100	C	1100	1000011	11000011
13	=	01101	D	1101	1000100	11000100
14	>	01110	E	1110	1000101	11000101

FPE track 1 cardholder name alphabet

This alphabet requires the following values to be used in the VFPE algorithm:

```
Number of characters in alphabet('n'): 45
```

Table 256. FPE track 1 cardholder name alphabet				
FPE alphabet number	**Character**	**ISO 7811 modified 7-bit ASCII**	**Standard data types 7-bit ASCII**	**Standard data types 8-bit ASCII**
0	space	1000000	0100000	01000000
1	#	1000011	0100011	01111011
2	$	0000100	0100100	01011011
3	(	0001000	0101000	01001101
4	)	1001001	0101001	01011101
5	-	0001101	0101101	01100000
6	0	0010000	0110000	11110000
7	1	1010001	0110001	11110001
8	2	1010010	0110010	11110010
9	3	0010011	0110011	11110011
10	4	1010100	0110100	11110100
11	5	0010101	0110101	11110101
12	6	0010110	0110110	11110110
13	7	1010111	0110111	11110111
14	8	1011000	0111000	11111000
15	9	0011001	0111001	11111001
16	A	1100001	1000001	11000001
17	B	1100010	1000010	11000010
18	C	0100011	1000011	11000011

Table 256. FPE track 1 cardholder name alphabet (continued)

FPE alphabet number	Character	ISO 7811 modified 7-bit ASCII	Standard data types 7-bit ASCII	Standard data types 8-bit ASCII
19	D	1100100	1000100	11000100
20	E	0100101	1000101	11000101
21	F	0100110	1000110	11000110
22	G	1100111	1000111	11000111
23	H	1101000	1001000	11001000
24	I	0101001	1001001	11001001
25	J	0101010	1001010	11010001
26	K	1101011	1001011	11010010
27	L	0101100	1001100	11010011
28	M	1101101	1001101	11010100
29	N	1101110	1001110	11010101
30	O	0101111	1001111	11010110
31	P	1110000	1010000	11010111
32	Q	0110001	1010001	11011000
33	R	0110010	1010010	11011001
34	S	1110011	1010011	11100010
35	T	0110100	1010100	11100011
36	U	1110101	1010101	11100100
37	V	1110110	1010110	11100101
38	W	0110111	1010111	11100110
39	X	0111000	1011000	11100111
40	Y	1111001	1011001	11101000
41	Z	1111010	1011010	11101001
42	[	0111011	1011011	10111010
43	\	1111100	1011100	11100000
44	]	111110	1011101	10111011

FPE track 1 discretionary data alphabet

This alphabet requires the following values to be used in the VFPE algorithm:

```
Number of characters in alphabet('n'): 47
```

Table 257. FPE track 1 discretionary data alphabet

FPE alphabet number	Character	ISO 7811 modified 7-bit ASCII	Standard data types 7-bit ASCII	Standard data types 8-bit ASCII
0	space	1000000	0100000	01000000
1	#	1000011	0100011	01111011
2	$	0000100	0100100	01011011
3	(	0001000	0101000	01001101
4	)	1001001	0101001	01011101
5	-	0001101	0101101	01100000
6	.	0001110	0101110	01001011
7	/	1001111	0101111	01100001
8	0	0010000	0110000	11110000
9	1	1010001	0110001	11110001
10	2	1010010	0110010	11110010
11	3	0010011	0110011	11110011
12	4	1010100	0110100	11110100
13	5	0010101	0110101	11110101
14	6	0010110	0110110	11110110
15	7	1010111	0110111	11110111
16	8	1011000	0111000	11111000
17	9	0011001	0111001	11111001
18	A	1100001	1000001	11000001
19	B	1100010	1000010	11000010
20	C	0100011	1000011	11000011
21	D	1100100	1000100	11000100
22	E	0100101	1000101	11000101
23	F	0100110	1000110	11000110
24	G	1100111	1000111	11000111
25	H	1101000	1001000	11001000
26	I	0101001	1001001	11001001
27	J	0101010	1001010	11010001
28	K	1101011	1001011	11010010
29	L	0101100	1001100	11010011
30	M	1101101	1001101	11010100
31	N	1101110	1001110	11010101
32	O	0101111	1001111	11010110

Table 257. FPE track 1 discretionary data alphabet (continued)				
FPE alphabet number	**Character**	**ISO 7811 modified 7-bit ASCII**	**Standard data types 7-bit ASCII**	**Standard data types 8-bit ASCII**
33	P	1110000	1010000	11010111
34	Q	0110001	1010001	11011000
35	R	0110010	1010010	11011001
36	S	1110011	1010011	11100010
37	T	0110100	1010100	11100011
38	U	1110101	1010101	11100100
39	V	1110110	1010110	11100101
40	W	0110111	1010111	11100110
41	X	0111000	1011000	11100111
42	Y	1111001	1011001	11101000
43	Z	1111010	1011010	11101001
44	[	0111011	1011011	10111010
45	\	1111100	1011100	11100000
46	]	0111110	1011101	10111011

VFPE track 2 discretionary data alphabet

This alphabet requires the following values to be used in the VFPE algorithm:

```
Number of characters in alphabet('n'): 10
```

Table 258. VFPE track 2 discretionary data alphabet						
VFPE alphabet number	**Character**	**ISO 7811 modified 5-bit ASCII**	**ISO 7811 modified 7-bit ASCII**	**Normal data type encoding**		
				4-bit	**7-bit ASCII**	**8-bit EBCDIC**
0	0	10000	0010000	0000	0110000	11110000
1	1	00001	1010001	0001	0110001	11110001
2	2	00010	1010010	0010	0110010	11110010
3	3	10011	0010011	0011	0110011	11110011
4	4	00100	1010100	0100	0110100	11110100
5	5	10101	0010101	0101	0110101	11110101
6	6	10110	0010110	0110	0110110	11110110
7	7	00111	1010111	0111	0110111	11110111
8	8	01000	1011000	1000	0111000	11111000
9	9	11001	0011001	1001	0111001	11111001

Authentication Parameter Generate (CSNBAPG and CSNEAPG)

The Authentication Parameter Generate callable service generates an authentication parameter (AP) and returns it encrypted using the key supplied in the AP_encrypting_key_identifier parameter.

The callable service name for AMODE(64) is CSNEAPG.

Format

```
CALL CSNBAPG(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             inbound_PIN_encrypting_key_identifier_length,
             inbound_PIN_encrypting_key_identifier,
             encrypted_PIN_block,
             issuer_domestic_code,
             card_secure_code,
             PAN_data,
             AP_encrypting_key_identifier_length,
             AP_encrypting_key_identifier,
             AP_value)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 0, 1, or 2.

rule_array

Direction	Type
Input	String

The keywords that provide control information to the callable service. The following table provides a list. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 259. Authentication Parameter Generate Rule Array Keywords

Keyword	Meaning
AP Protection Method (One, optional)	
ENCRYPT	Specifies the AP value should be returned encrypted under the *AP_encrypting_key_identifier* parameter. This is the default.
CLEAR	Specifies the AP value should be returned in the clear.
AP Value Format (One, optional)	
BCD	Specifies the output format of the AP as binary coded decimal. This is the default.

inbound_PIN_encrypting_key_identifier_length

Direction	Type
Input	Integer

Length of the *inbound_PIN_encrypting_key_identifier* field in bytes.

When the *inbound_PIN_encrypting_key_identifier* parameter contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

inbound_PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to unwrap the PIN block. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte DES key token of key type IPINENC.

For X9.143 keys, the identifier is a key block of a DES PIN-encrypting key: key usage P0, algorithm T, and mode of use D.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key

encrypted_PIN_block

Direction	Type
Input	String

The ISO-0 PIN block encrypted with the inbound_PIN_encrypting_key_identifier. The PIN within the PIN block must be a 5 digit value.

issuer_domestic_code

Direction	Type
Input	Alphanumeric Character String

A 5 byte alphanumeric character string.

card_secure_code

Direction	Type
Input	String

An 8 byte string of digits grouped into two 4 byte sections. The 4 digits in a section cannot all be zero; for example, the value “0000” is invalid.

PAN_data

Direction	Type
Input	String

The personal account number (PAN). Must be 12 characters long.

AP_encrypting_key_identifier_length

Direction	Type
Input	Integer

The length of the *AP_encrypting_key_identifier* field in bytes.

This value is 64 when a label is supplied.

When the key identifier is a key token or block, the value is the length of the token.

The maximum value is 9992.

The value may be 0 when the 'CLEAR' rule array option is specified.

AP_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to wrap the *AP_value*. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

If the AP Protection Method was specified as CLEAR in the *rule_array* parameter, this parameter is ignored.

For CCA key tokens, the identifier is a 64-byte DES key token of a double-length or triple-length DATA key.

For X9.143 keys, the identifier is a key block of a DES data-encrypting key: key usage D0, algorithm T, and mode of use B or E.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

AP_value

Direction	Type
Output	String

An 8 byte character string containing the generated authentication parameter.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 260. Access Control Points for Authentication Parameter Generate (CSNBAPG and CSNEAPG)

Access control point	Restrictions
Authentication Parameter Generate	None
Authentication Parameter Generate - Clear	Allow AP value to be returned in the clear

When the **Disallow translation from DES wrapping to weaker DES wrapping** access control point is enabled, this service will fail if the input key-encrypting key identifier is stronger than the output key-encrypting key identifier.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 261. Authentication Parameter Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 261. Authentication Parameter Generate required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Clear PIN Encrypt (CSNBCPE and CSNECPE)

The Clear PIN Encrypt callable service formats a PIN into one of these PIN block formats and encrypts the results. You can use this service to create an encrypted PIN block for transmission. With the RANDOM keyword, you can have the service generate random PIN numbers.

Note: A clear PIN is a sensitive piece of information. Ensure that your application program and system design provide adequate protection for any clear PIN value.

- IBM 3621 format
- IBM 3624 format
- ISO-0 format (same as the ANSI X9.8, VISA-1, and ECI formats)
- ISO-1 format (same as the ECI-4 format)
- ISO-2 format
- ISO-3 format
- ISO-4 format
- IBM 4704 encrypting PINPAD (4704-EPP) format
- VISA 2 format
- VISA 3 format
- VISA 4 format
- ECI2 format
- ECI3 format

An enhanced PIN security mode is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the Enhanced PIN Security access control point in the domain role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

The callable service name for AMODE(64) invocation is CSNECPE.

Format

```
CALL CSNBCPE(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              PIN_encrypting_key_identifier,
              rule_array_count,
              rule_array,
              clear_PIN,
              PIN_profile,
              PAN_data,
              sequence_number,
              encrypted_PIN_block )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is defined in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to encrypt the PIN block. The key identifier is a variable-length operational key token or key block or the 64-byte key label of an operational token or block in key storage.

When the PIN block format is ISO-4, the key is an AES key:

- For CCA keys, the variable-length symmetric key-token must have a token algorithm of AES and a key type of PINPROT. In addition, the key usage fields must have the encryption operation set so that the key can be used for encryption (ENCRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN function usage CPINENC must be enabled.
- For X9.143 keys, the identifier is a key block of an AES PIN-encrypting key: key usage P0, algorithm A, and mode of use E.

When the PIN block format is other than ISO-4, the key is a DES key:

- For CCA keys, the identifier is a 64-byte DES key token of key type OPINENC with key usage attribute CPINENC enabled in the control vector.
- For X9.143 keys, the identifier is a key block of a DES PIN-encrypting key: key usage P0, algorithm T, and mode of use E.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. Valid values are 0 and 1.

rule_array

Direction	Type
Input	Character String

Keywords that provide control information to the callable service. The keyword is left-justified in an 8-byte field, and padded on the right with blanks. All keywords must be in contiguous storage. The rule array keywords are shown as follows:

Table 262. Process Rules for the Clear PIN Encryption Callable Service

Process Rule	Description
ENCRYPT	This is the default. Use of this keyword is optional.
RANDOM	Causes the service to generate a random PIN value. The length of the PIN is based on the value in the *clear_PIN* variable. Set the value of the clear PIN to zero and use as many digits as the desired random PIN; pad the remainder of the clear PIN variable with space characters.

clear_PIN

Direction	Type
Input	String

A 16-character string with the clear PIN. The value in this variable must be left-justified and padded on the right with space characters.

PIN_profile

Direction	Type
Input	String

A 24-byte string containing three 8-byte elements with a PIN block format keyword, the format control keyword, NONE, and a pad digit as required by certain formats.See “The PIN profile” on page 609 for additional information.

PAN_data

Direction	Type
Input	String

A primary account number (PAN) in character format. The service uses this parameter if the PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 12-byte value in application storage. The information in this parameter will be ignored, but the parameter must be specified.

When using the ISO-0, ISO-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 – 19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Offset	Length	Description
0	2	Length of the PAN data field, *p*.
2	p	10 to 19 bytes of PAN data.
2+*p*	0-9	Padding.

sequence_number

Direction	Type
Input	Integer

The sequence number used by certain PIN block formats.

- For the 3621 PIN block format, provide a value in the range from 0 to 65535.
- For the 4704-EPP PIN block format, provide a value in the range from 0 to 255.
- For other PIN block formats, provide a value of 99999.

encrypted_PIN_block

Direction	Type
Output	String

The field that receives the 8-byte or 16-byte encrypted PIN block. When the PIN block format is ISO-4, the PIN block will be 16 bytes long. For all other formats, the PIN block will be 8 bytes long.

Restrictions

The format control specified in the PIN profile must be NONE.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the Clear PIN Encrypt service and the label of the *PIN_encrypting_key_identifier*.

Access control point

The **Clear PIN Encrypt** access control point controls the function of this service.

An enhanced PIN security mode is available for extracting PINs from a 3621 or 3624 encrypted PIN-block and formatting an encrypted PIN block into IBM 3621 or 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** access control.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *PIN_profile* parameter is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 263. Clear PIN Encrypt required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.

Table 263. Clear PIN Encrypt required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Clear PIN Generate (CSNBPGN and CSNEPGN)

Use the Clear PIN Generate callable service to generate a clear PIN, a PIN validation value (PVV), or an offset according to an algorithm. You supply the algorithm or process rule using the *rule_array* parameter.

- IBM 3624 (IBM-PIN or IBM-PINO)
- VISA PIN validation value (VISA-PVV)
- Interbank PIN (INBK-PIN)

The callable service can execute only when ICSF is in special secure mode. This mode is described in "Special secure mode" on page 10.

For guidance information about VISA, see their appropriate publications.

The callable service name for AMODE(64) invocation is CSNEPGN.

Format

```
CALL CSNBPGN(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              PIN_generating_key_identifier,
              rule_array_count,
              rule_array,
              PIN_length,
              PIN_check_length,
              data_array,
              returned_result )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is defined in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

PIN_generating_key_identifier

Direction	Type
Input/Output	Character String

The identifier of the PIN-generating key to generate the PIN. The key identifier is a variable-length operational key token or key block or the 64-byte key label of an operational token or block in key storage.

For CCA key tokens, the identifier is a 64-byte DES key token of key type PINGEN with key usage attribute CPINGEN enabled in the control vector.

For X9.143 key blocks, the identifier is a key block of a DES PIN-generation key:

- For process rule keywords IBM-PIN and IBM-PINO: key usage V1, algorithm T, and mode of use C or G.
- For process rule keyword VISA-PVV: key usage V2, algorithm T, and mode of use C or G.
- For process rule keywords GBP-PIN and INBK-PIN: key usage V0, algorithm T, and mode of use C or G.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

rule_array_count

Direction	Type
Input	Integer

The number of process rules specified in the *rule_array* parameter. The value must be 1.

rule_array

Direction	Type
Input	Character String

The process rule provides control information to the callable service. Specify one of the values in Table 264 on page 631. The keyword is left-justified in an 8-byte field, and padded on the right with blanks.

Table 264. Process Rules for the Clear PIN Generate Callable Service

Process Rule	Description
GBP-PIN	The IBM German Bank Pool PIN, which uses the institution PINGEN key to generate an institution PIN (IPIN).
IBM-PIN	The IBM 3624 PIN, which is an institution-assigned PIN. It does not calculate the PIN offset.
IBM-PINO	The IBM 3624 PIN offset, which is a customer-selected PIN and calculates the PIN offset (the output).
INBK-PIN	The Interbank PIN is generated.
VISA-PVV	The VISA PIN validation value. Input is the customer PIN.

PIN_length

Direction	Type
Input	Integer

The length of the PIN used for the IBM algorithms only, IBM-PIN or IBM-PINO. Otherwise, this parameter is ignored. Specify an integer from 4 through 16.

PIN_check_length

Direction	Type
Input	Integer

The length of the PIN offset used for the IBM-PINO process rule only. Otherwise, this parameter is ignored. Specify an integer from 4 through 16.

Note: The PIN check length must be less than or equal to the integer specified in the *PIN_length* parameter.

data_array

Direction	Type
Input	String

Three 16-byte data elements required by the corresponding *rule_array* parameter. The data array consists of three 16-byte fields or elements whose specification depends on the process rule. If a process rule only requires one or two 16-byte fields, then the rest of the data array is ignored by the callable service. Table 265 on page 632 describes the array elements.

Table 265. Array Elements for the Clear PIN Generate Callable Service	
Array Element	**Description**
Clear_PIN	Clear user selected PIN of 4 to 12 digits of 0 through 9. Left-justified and padded with spaces. For IBM-PINO, this is the clear customer PIN (CSPIN). For IBM-PIN and GBP-PIN, this field is ignored.
Decimalization_table	Decimalization table for IBM and GBP only. Sixteen digits of 0 through 9. **Note:** If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the domain role, this table must match one of the active decimalization tables in the coprocessors.
Trans_sec_parm	For VISA only, the leftmost sixteen digits. Eleven digits of the personal account number (PAN). One digit key index. Four digits of customer selected PIN. For Interbank only, sixteen digits. Eleven right-most digits of the personal account number (PAN). A constant of 6. One digit key selector index. Three digits of PIN validation data.
Validation_data	Validation data for IBM and IBM German Bank Pool padded to 16 bytes. One to sixteen characters of hexadecimal account data left-justified and padded on the right with blanks.

Table 266 on page 632 lists the data array elements required by the process rule (*rule_array* parameter). The numbers refer to the process rule's position within the array.

Table 266. Array Elements Required by the Process Rule					
Process Rule	**IBM-PIN**	**IBM-PINO**	**GBP-PIN**	**VISA-PVV**	**INBK-PIN**
Decimalization_table	1	1	1		
Validation_data	2	2	2		
Clear_PIN		3			
Trans_sec_parm				1	1

returned_result

Direction	**Type**
Output	Character String

The 16-byte generated output, left-justified and padded on the right with blanks.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If you are using the IBM 3624 PIN and IBM German Bank Pool PIN algorithms, you can supply an unencrypted customer selected PIN to generate a PIN offset.

Access control points

This table shows the access control points in the domain role that control the function of this service.

Table 267. Required access control points for Clear PIN Generate

Rule array keywords	Access control point
IBM-PIN **IBM-PINO**	Clear PIN Generate - 3624
GBP-PIN	Clear PIN Generate - GBP
VISA-PVV	Clear PIN Generate - VISA PVV
INBK-PIN	Clear PIN Generate - Interbank

If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the domain role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 268. Clear PIN Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.

Table 268. Clear PIN Generate required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Related information

PIN algorithms are shown in "PIN formats and algorithms" on page 1631.

Clear PIN Generate Alternate (CSNBCPA and CSNECPA)

Use the Clear PIN Generate Alternate service to generate a clear VISA PVV (PIN validation value) from an input encrypted PIN block or to produce a 3624 offset from a customer-selected encrypted PIN. The PIN block can be encrypted under either an input PIN-encrypting key (IPINENC) or an output PIN-encrypting key (OPINENC). When the input encrypted PIN block is in ISO-4 format, the PIN-encrypting key is an AES PINPROT key.

An enhanced PIN security mode is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. To do this, you must enable the Enhanced PIN Security access control point in the domain role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.

An enhanced PIN security mode on the CEX3C and later is available to implement restrictions required by the ANSI X9.8 PIN standard. To enforce these restrictions, you must enable the **ANSI X9.8 PIN - Enforce PIN block restrictions** control point in the domain role.

The callable service name for AMODE(64) invocation is CSNECPA.

Format

```
CALL CSNBCPA(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             PIN_encryption_key_identifier,
             PIN_generation_key_identifier,
             PIN_profile,
             PAN_data,
             encrypted_PIN_block,
             rule_array_count,
             rule_array,
             PIN_check_length,
             data_array,
             returned_PVV)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

PIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to encrypt the PIN block. The key identifier is a variable-length operational key token or key block or the 64-byte key label of an operational token in key storage.

When the PIN block format is ISO-4, the key is an AES key:

- For CCA keys, the variable-length symmetric key-token must have a token algorithm of AES and a key type of PINPROT. In addition, the key usage fields may indicate that the key can be used for encryption (ENCRYPT) with PIN function usage EPINGEN or decryption (DECRYPT) with PIN function usage CPINGENA, the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, and PIN block format usage must be ISO-4.
- For X9.143 keys, the identifier is a key block of an AES PIN-encrypting key: key usage P0, algorithm A, and mode of use D or E.

When the PIN block format is other than ISO-4, the key is a DES key:

- For CCA keys, the identifier is a 64-byte DES key token of key type IPINENC or OPINENC.
- For X9.143 keys, the identifier is a variable-length key block of a TDES PIN-encrypting key: key usage P0, algorithm T, and mode of use D or E.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

PIN_generation_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-generating key to generate the PIN. The key identifier is a variable-length operational key token or key block or the 64-byte key label of an operational token or block in key storage.

For CCA key tokens, the identifier is a 64-byte DES key token of key type PINGEN with key usage attribute PINGEN enabled in the control vector.

For X9.143 keys, the identifier is a variable-length key block of a TDES PIN-generating key:

- For process rule keyword IBM-PINO: key usage V1, algorithm T, and mode of use C or G.
- For process rule keyword VISA-PVV: key usage V2, algorithm T, and mode of use C or G.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

PIN_profile

Direction	Type
Input	Character String

The three 8-byte character elements that contain information necessary to extract a PIN from a formatted PIN block. The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the Clear PIN Generate Alternate service. See "The PIN profile" on page 609 for additional information.

PAN_data

Direction	Type
Input	String

A primary account number (PAN) in character format. The service uses this parameter if the PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 12-byte value in application storage. The information in this parameter will be ignored, but the parameter must be specified.

When using the ISO-0, ISO-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 – 19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Table 269. PAN data structure

Offset	Length	Description
0	2	Length of the PAN data field, p.
2	p	10 to 19 bytes of PAN data.
2+p	0-9	Padding.

encrypted_PIN_block

Direction	Type
Input	String

The field that receives the 8-byte or 16-byte encrypted PIN block. When the PIN block format is ISO-4, the PIN block will be 16 bytes long. For all other formats, the PIN block will be 8 bytes long. The service uses the key that is specified in the *PIN_encryption_key_identifier* parameter to encrypt the block.

rule_array_count

Direction	Type
Input	Integer

The number of process rules specified in the *rule_array* parameter. The value may be 1, 2, or 3.

rule_array

Direction	Type
Input	Character String

The process rule for the PIN generation algorithm. Specify IBM-PINO or "VISA-PVV" (the VISA PIN verification value) in an 8-byte field, left-justified, and padded with blanks. The *rule_array* points to an array of one or two 8-byte elements as follows:

Table 270. Rule Array Elements for the Clear PIN Generate Alternate Service

Rule Array Element	Function of Rule Array keyword
1	PIN calculation method
2	PIN extraction method

The first element in the rule array must specify one of the keywords that indicate the PIN calculation method as shown:

Table 271. Rule Array Keywords (First Element) for the Clear PIN Generate Alternate Service

PIN Calculation Method Keyword	Meaning
IBM-PINO	This keyword specifies use of the IBM 3624 PIN Offset calculation method.
VISA-PVV	This keyword specifies use of the VISA PVV calculation method.

If the second element in the rule array is provided, one of the PIN extraction method keywords shown in Table 250 on page 610 may be specified for the given PIN block format. See "PIN block format and PIN extraction method keywords" on page 610 for additional information. If the default extraction method for a PIN block format is desired, you may code the rule array count value as 1.

The PIN extraction methods operate as follows:

PINBLOCK
Specifies that the service use one of these:

- the PIN length, if the PIN block contains a PIN length field
- the PIN delimiter character, if the PIN block contains a PIN delimiter character.

PADDIGIT
Specifies that the service use the pad value in the PIN profile to identify the end of the PIN.

HEXDIGIT
Specifies that the service use the first occurrence of a digit in the range from X'A' to X'F' as the pad value to determine the PIN length.

PINLENxx
Specifies that the service use the length specified in the keyword, where xx can range from 4 to 16 digits, to identify the PIN.

PADEXIST
Specifies that the service use the character in the 16th position of the PIN block as the value of the pad value.

PIN_check_length

Direction	Type
Input	Integer

The length of the PIN offset used for the IBM-PINO process rule only. Otherwise, this parameter is ignored. Specify an integer from 4 through 16.

data_array

Direction	Type
Input	String

Three 16-byte elements. Table 272 on page 638 describes the format when IBM-PINO is specified. Table 273 on page 638 describes the format when VISA-PVV is specified.

Table 272. Data Array Elements for the Clear PIN Generate Alternate Service (IBM-PINO)

Array Element	Description
decimalization_table	This element contains the decimalization table of 16 characters (0 to 9) that are used to convert hexadecimal digits (X'0' to X'F') of the enciphered validation data to the decimal digits X'0' to X'9'). **Note:** If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the domain role, this table must match one of the active decimalization tables in the coprocessors.
validation_data	This element contains one to 16 characters of account data. The data must be left justified and padded on the right with space characters.
Reserved-3	This field is ignored, but you must specify it.

When using the VISA-PVV keyword, identify these elements in the data array.

Table 273. Data Array Elements for the Clear PIN Generate Alternate Service (VISA-PVV)

Array Element	Description
Trans_sec_parm	For VISA-PVV only, the leftmost twelve digits. Eleven digits of the personal account number (PAN). One digit key index. The rest of the field is ignored.
Reserved-2	This field is ignored, but you must specify it.
Reserved-3	This field is ignored, but you must specify it.

returned_PVV

Direction	Type
Output	Character

A 16-byte area that contains the result left-justified and padded with blanks. When VISA-PVV, this is a 4 byte value. When IBM-PINO, the value is the same length as the clear PIN.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Use of the Visa-PVV PIN-calculation method will always output four digits rather than padding the output with binary zeros to the length of the PIN.

Access control points

This table shows the access control points in the domain role that control the function of this service.

Table 274. Required access control points for Clear PIN Generate Alternate

Rule array keywords	Access control point
IBM-PINO	Clear PIN Generate Alternate - 3624 Offset
VISA-PVV	Clear PIN Generate Alternate - VISA PVV

An enhanced PIN security mode is available for extracting PINs from a 3621 or 3624 encrypted PIN-block and formatting an encrypted PIN block into IBM 3621 or 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** access control.

When the **ANSI X9.8 PIN - Enforce PIN block restrictions** control is enabled in the domain role, the PIN block format in the *input_PIN_profile* parameter must be ISO-0 or ISO-3.

If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the domain role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *PIN_profile* parameter is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 275. Clear PIN Generate Alternate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. X9.143 key blocks are not supported.

Table 275. Clear PIN Generate Alternate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require the July 2019 or later licensed internal code (LIC). PIN block format ISO-4 requires the October 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	PIN block format ISO-4 requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

CVV Key Combine (CSNBCKC and CSNECKC)

Use this callable service to combine 2 single length CCA internal key tokens into 1 double-length CCA key token containing a CVVKEY-A key type for use with the VISA CVV Service Generate or VISA CVV Service Verify callable services. This combined double-length key satisfies current VISA requirements and eases translation between TR-31 and CCA formats for CVV keys.

The callable service name for AMODE(64) is CSNECKC.

Format

```
CALL CSNBCKC(
          return_code,
          reason_code,
          exit_data_length,
          exit_data,
          rule_array_count,
          rule_array,
          key_a_identifier_length,
```

```
        key_a_identifier,
        key_b_identifier_length,
        key_b_identifier,
        output_key_identifier_length,
        output_key_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 0, 1, or 2.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The *rule_array* keywords for this callable service are shown in the following table.

Table 276. Keywords for CVV Key Combine Rule Array Control Information	
Keyword	**Meaning**
Key Wrapping Method (Optional)	
USECONFG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.
WRAP-ECB	Specifies that the original wrapping method is to be used.
Translation Control (Optional)	
ENH-ONLY	Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the original method. This restricts translation to the original method. If the keyword is not specified translation to the original method will be allowed. This turns on bit 56 in the control vector. This keyword is not valid if processing a zero CV data key. This is the default when the wrapping method is WRAPENH3. **Note:** If the default wrapping method is ECB mode, but the enhanced mode and the ENH-ONLY restriction are desired for a particular key token, combine the ENH-ONLY keyword with the WRAP-ENH keyword. If this is not done, then an error will be returned because ENH-ONLY will conflict with the default wrapping mode if the default wrapping method is ECB mode.

key_a_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *key_a_identifier* parameter, in bytes. The value must be 64.

key_a_identifier

Direction	Type
Input	String

This parameter contains a 64-byte internal key token or a label of a single-length zero CV DATA key, a DATA key with the MAC gen and/or verify bits on, or a CVVKEY-A key. The internal key token contains the key-A key that encrypts information in the CVV process.

key_b_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *key_b_identifier* parameter, in bytes. The value in this parameter must be 64.

key_b_identifier

Direction	Type
Input	String

This parameter contains a 64-byte internal key token or a label of a single-length zero CV DATA key, a DATA key with the MAC gen and/or verify bits on, or a CVVKEY-B key. The internal key token contains the key-B key that decrypts information in the CVV process.

output_key_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *output_key_identifier* parameter, in bytes. The value in this parameter must be 64.

output_key_identifier

Direction	Type
Output	String

This parameter contains the output key token. It is a double-length MAC key with CV bits 0-3 set to 0010 to indicate a CVVKEY-A key.

Restrictions

None.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

If key-A and key-B have different CV values for either the Export bit (CV bit 17) or the TR-31 Translate bit (CV bit 57), then the keys cannot be combined and an error is returned (8 / 39).

Both key-A and key-B must be usable in the same role for either the CVV Generate or CVV Verify service, otherwise an error occurs.

Both key-A and key-B must be usable for the same service (CVV Generate or CVV Verify). It is not acceptable to combine a Generate and a Verify key.

If key-A or key-B is a Generate-Only key and the pair pass all criteria to be combined as a single output key, the resulting CV in the output token will indicate a double-length Generate-Only key capability.

This following tables show the action taken by the service for different combinations of input key types.

Table 277. Key type combinations for the CVV Key Combine callable service

Action taken based on key types of the 2 input keys		**8-byte input key provided as right-half (key-B) of 16 B CVV key**			
		CVVKEY-A	**CVVKEY-B**	**DATA key**	**ANY-MAC key**
8-byte input key provided as left-half (key-A) of 16 B CVV key	CVVKEY-A	Always reject	Always allow	Conditional allow*	Conditional allow*
	CVVKEY-B	Always reject	Always reject	Always reject	Always reject
	DATA key	Always reject	Conditional allow*	Always allow	Conditional allow*
	ANY-MAC key	Always reject	Conditional allow*	Conditional allow*	Always allow
* – Requires Access Control Point “CVV Key Combine – Permit mixed key types” enabled					

Table 278. WRAPENH3 Key-wrapping matrix for CVV_Key_Combine

key-A or key-B is wrapped using WRAPENH3 method	**WRAPENH3 (by keyword or by default)**	**Resulting form of output key or error**
No	No to both	ECB wrapped.
Yes	No to both	Wrap type conflict, 8/2161 (X'871') because ECB wrapping is requested so overall wrapping is weakened.
No	Yes to both	WRAPENH3, CV bit 56 = B'1' (required for WRAPENH3).
Yes	Yes to both	

There are restrictions on the available wrapping methods for the output key derived from the wrapping methods employed and CV restrictions of the input keys. These are detailed in the following table.

Table 279. Wrapping combinations for the CVV Combine Callable Service

key-A OR key-B uses WRAP-ENH wrapping method	**key-A OR key-B has enhanced-only bit (CV bit 56) set to 1 (implies WRAP-ENH for that token)**	**WRAP-ENH keyword passed or WRAP-ENH is default wrapping method**	**ENH-ONLY keyword passed**	**Outcome (form of output key or error)**
no	no	no to both	no	output is ECB wrapped
yes	no	no to both	no	error
no	no	yes to either	no	output is ENH wrapped, bit 56 not set
yes	no	yes to either	no	output is ENH wrapped, bit 56 not set

Table 279. Wrapping combinations for the CVV Combine Callable Service (continued)				
key-A OR key-B uses WRAP-ENH wrapping method	**key-A OR key-B has enhanced-only bit (CV bit 56) set to 1 (implies WRAP-ENH for that token)**	**WRAP-ENH keyword passed or WRAP-ENH is default wrapping method**	**ENH-ONLY keyword passed**	**Outcome (form of output key or error)**
no	no	yes to either	yes	output is ENH wrapped, bit 56 is set
yes	no	yes to either	yes	output is ENH wrapped, bit 56 is set
yes	yes	yes to either	no	output is ENH wrapped, bit 56 is set
yes	yes	yes to either	yes	output is ENH wrapped, bit 56 is set
no	no	no to both	yes	error
yes	no	no to both	yes	error
yes	yes	no to both	no	error
yes	yes	no to both	yes	error

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the generated key.

Access control points

The **CVV Key Combine** access control point controls the function of this service.

The key types of the key_a_identifier and key_b_identifier must be the same unless the **CVV Key Combine – Permit mixed key types** access control point is enabled. This means both key identifiers must be DATA keys or both must be MAC keys when the access control point is disabled. When enabled, DATA keys can be used with MAC keys.

When the key wrapping method keyword specifies a wrapping method that is not the default method, the **CVV Key Combine - Allow wrapping override keywords** access control point must be enabled.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 280. CVV Key Combine required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC).

Table 280. CVV Key Combine required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC).
	Crypto Express7 CCA Coprocessor	Rule array keyword WRAPENH3 requires the May 2021 or later licensed internal code (LIC).
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

EMV Scripting Service (CSNBESC and CSNEESC)

The EMV Scripting Service is a mechanism for sending commands to an EMV payment card. The commands are used to update card parameters including potentially the PIN. Commands may be encrypted for confidentiality or MAC'd for integrity or both.

Scripts are generated by the issuer, or the issuer's agent, when a transaction is received from a payment card. This service receives the script as input, encrypts, MAC's it or both, and returns either the encrypted script, the MAC, or both. The output is intended to be sent back to the payment card along with the response.

This service performs the following EMV scripting functions:

- Scripting with integrity.

  The message is MAC'd with a session key derived from the issuer master key specified in the *issuer_integrity_master_key_identifier* parameter.
- Scripting with confidentiality (for protection of scripts that may or may not contain a PIN).

  The message is encrypted with a session key derived from the issuer master key specified in the *issuer_confidentiality_master_key_identifier* parameter.
- Scripting with confidentiality and integrity (for protection of scripts that may or may not contain a PIN).

  The message is first encrypted (for confidentiality) with a session key derived from the issuer master key specified in the *issuer_confidentiality_master_key_identifier* parameter and then it is MAC'd (for integrity) with a session key derived from the issuer master key specified in the *issuer_integrity_master_key_identifier* parameter.
- PIN change/unblock.

  Visa PIN change/unblocking as described in VISA Integrated Circuit Card Specification, v1.4.0. The PIN can be changed by either specifying the new PIN only or specifying both the current PIN and the new PIN. See *new_PIN_encrypting_key_identifier* and *current_PIN_encrypting_key_identifier* parameters for additional information. The message is encrypted with a session key derived from the issuer master key specified in the *issuer_confidentiality_master_key_identifier* parameter.

This service can be used in the following specific brand modes:

- Visa
- MasterCard
- EMV

The callable service name for AMODE(64) invocation is CSNEESC.

Format

```
CALL CSNBESC(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             issuer_integrity_master_key_identifier_length,
             issuer_integrity_master_key_identifier,
             issuer_confidentiality_master_key_identifier_length,
             issuer_confidentiality_master_key_identifier,
             new_PIN_encrypting_key_identifier_length,
             new_PIN_encrypting_key_identifier,
             current_PIN_encrypting_key_identifier_length,
             current_PIN_encrypting_key_identifier,
             new_PIN_block,
             current_PIN_block,
             pan_length,
             pan,
             pan_seq_number,
             atc,
             unpredictable_number,
             input_message_length,
             input_message,
             PIN_offset,
             PIN_format,
             output_message_length,
             output_message,
             mac_length,
             mac,
             reserved1_length,
             reserved1,
             reserved2_length,
             reserved2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 3 and the maximum value is 5.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 281. Rule array keywords for EMV Scripting Service

Keyword	Meaning
Algorithm (Required)	
TDES	Specifies the use of Triple-DES.
Action (One required)	
SMINT	Secure messaging with integrity.
SMCON	Secure messaging with confidentiality. Only MC and EMV key modes supported.
SMCONPIN	Secure messaging with confidentiality for commands containing a PIN.
SMCONINT	Secure messaging with both confidentiality and integrity. Only MC and EMV key modes supported.
SMCIPIN	Secure messaging with both confidentiality and integrity for commands containing a PIN.
VISAPIN	Visa PIN change/unblocking as described in VISA Integrated Circuit Card Specification, v1.4.0.
Key mode (One required). Defines the key derivation mechanism.	

Table 281. Rule array keywords for EMV Scripting Service (continued)	
Keyword	**Meaning**
VISA	Specifies to use the Visa Cryptogram Version 10 key derivation. Not valid with SMCON and SMCONINT.
MC	Specifies to use the MasterCard M/CHIP 2.1 key derivation. The *random_number* parameter is used. Padding is according to EMV rules.
EMV	Specifies to use the session key derivation as described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.3.
Control flag (Optional)	
APPANSEQ	Specifies to append the PAN sequence number when the card specific master key is derived. See the descriptions of *pan* and *pan_seq_number*. The default is not to append the PAN sequence number.
Branch Factor (One optional, valid only with key mode EMV). The branching factor is to be used in EMV session key derivation.	
TDESEMV2	Specifies a branch factor of 2 for a height of 16. This is the default.
TDESEMV4	Specifies a branch factor of 4 for a height of 8.

issuer_integrity_master_key_identifier_length

Direction	**Type**
Input	Integer

Specifies the length of the *issuer_integrity_master_key_identifier* parameter in bytes. The value must be 0 or 64. When the action keywords SMCON or SMCONPIN is specified, the value must be 0 and the *issuer_integrity_master_key_identifier* parameter is ignored. Otherwise, the value must be 64 and a key token or label must be supplied in the *issuer_integrity_master_key_identifier* parameter.

issuer_integrity_master_key_identifier

Direction	**Type**
Input/Output	String

The 64-byte DES key identifier (either an internal token or key label) for the issuer master key to be used for secure messaging with integrity (actions SMINT, SMCONINT, and SMCIPIN) or the issuer master key for authentication (action VISAPIN).

The issuer master key is the DES key from which the card specific keys and session keys for scripting are derived.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

Table 282. EMV Scripting Service: Key requirements

Key mode keyword	Key derivation used	Key type	Subtype	Key generating bits
VISA	SESS-XOR	DKYGENKY	0	Must be ‘0010’ (keyword DMAC).
MC	MasterCard M/Chip 2.1	DKYGENKY	1	Must be ‘0010’ (keyword DMAC).
EMV	EMV key derivation	DKYGENKY	0	Must be ‘0010’ (keyword DMAC).

Note: For action VISAPIN, this is the issuer master key for authentication. The key is used for preparing the material used to form the PIN block for PIN change/unblock. The card specific key is derived using the same PAN data as for the other issuer master keys. The key must be of type DKYGENKY, subtype 0, and the key generating bits must be '0010' (keyword DMAC).

issuer_confidentiality_master_key_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *issuer_confidentiality_master_key_identifier* parameter in bytes. The value must be 0 or 64. When action keyword SMINT is specified, the value must be 0 and the *issuer_confidentiality_master_key_identifier* parameter is ignored. Otherwise, the value must be 64 and a key token or label must be supplied in the *issuer_confidentiality_master_key_identifier* parameter.

issuer_confidentiality_master_key_identifier

Direction	Type
Input/Output	String

The 64-byte DES key identifier (either an internal token or key label) for the issuer master key to be used for secure messaging with confidentiality (actions SMCON, SMCONPIN, SMCONINT, SMCIPIN, VISAPIN). The issuer master key is the DES key from which card specific keys and session keys for scripting are derived.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

Table 283. Key type requirements for actions SMCON and SMCONINT

Key mode keyword	Key derivation used	Key type	Subtype	Key generating bits
VISA	SESS-XOR	DKYGENKY	0	Must be ‘0001’ (keyword DDATA).
MC	MasterCard M/Chip 2.1	DKYGENKY	1	Must be ‘0001’ (keyword DDATA).
EMV	EMV key derivation	DKYGENKY	0	Must be ‘0001’ (keyword DDATA).

Table 284. Key type requirements for actions SMCONPIN, SMCIPIN, and VISAPIN

Key mode keyword	Key derivation used	Key type	Subtype	Key generating bits
VISA	SESS-XOR	DKYGENKY	0	Must be '1001' (keyword DMPIN).
MC	MasterCard M/Chip 2.1	DKYGENKY	1	Must be '1001' (keyword DMPIN).
EMV	EMV key derivation	DKYGENKY	0	Must be '1001' (keyword DMPIN).

new_PIN_encrypting_key_identifier_length

Direction	Type
Input	Integer

Specifies the length of the *new_PIN_encrypting_key_identifier* parameter in bytes. The value must be 0 or 64. When action keyword VISAPIN is specified, the value must be 64. Otherwise, the value must be 0 and the *new_PIN_encrypting_key_identifier* parameter is ignored.

new_PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The 64-byte DES key identifier (either an internal token or key label) for the key that encrypts the new PIN block. The key type can be either IPINENC or OPINENC. There are separate Cryptographic Coprocessor access points required based on the rules selected. For additional information, see "Cryptographic services used by EMV Scripting Service" on page 655 and "Access control points" on page 656.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

current_PIN_encrypting_key_identifier_length

Direction	Type
Input	Integer

Specifies the length of the *current_PIN_encrypting_key_identifier* parameter in bytes. The value must be 0 or 64. When action keyword SMCONPIN, SMCIPIN, or VISAPIN is specified, the value must be 64. Otherwise, the value must be 0 and the *current_PIN_encrypting_key_identifier* parameter is ignored.

current_PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The 64-byte DES key identifier (either an internal token or key label) for the key that encrypts the PIN block with the current PIN.

For actions SMCONPIN and SMCIPIN, the key type must be IPINENC. For action VISAPIN, the key type can be either IPINENC or OPINENC.

There are separate Cryptographic Coprocessor access points required depending on the rules selected. For additional information, see "Cryptographic services used by EMV Scripting Service" on page 655 and "Access control points" on page 656.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

new_PIN_block

Direction	Type
Input	String

The 8-byte PIN block encrypted by the *new_PIN_encrypting_key_identifier*.

For VISAPIN, this is the new PIN. All PIN block formats are supported. This parameter is ignored for all other action keywords.

current_PIN_block

Direction	Type
Input	String

The 8-byte PIN block for the current PIN encrypted by the *current_PIN_encrypting_key_identifier*.

When VISAPIN is specified, this is the current PIN. All Pin block formats are supported.

When SMCONPIN or SMCIPIN is specified, PIN block formats supported are ISO-0, ISO-1, and ISO-2.

When the action keyword is SMCON, SMCONINT, or SMINT, this parameter is ignored.

pan_length

Direction	Type
Input	Integer

Length in bytes of the *pan* parameter. The value must be 10.

pan

Direction	Type
Input	String

The 10-byte EMV card's Primary Account Number. The data must be in compressed numeric format and right justified in a 10-byte field, padded to the left with zeroes. For example, PAN 1234567890 must be provided as x'00000000001234567890'.

This data is used in combination with the PAN sequence number to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

pan_seq_number

Direction	Type
Input	String

The 1-byte sequence number of the EMV card's Primary Account Number. If the APPANSEQ control flag rule array keyword was specified, this PAN sequence number is used in combination with the PAN to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

atc

Direction	Type
Input	String

The 2-byte application transaction counter that is used for session key derivation. See the key mode rules for more information on session key derivation.

This parameter must be 2 bytes.

Note: The first byte is the high-order byte and the second byte is the low order byte.

unpredicatable_number

Direction	Type
Input	String

The 8-byte random number for secure messaging with MasterCard M/CHIP 2.1 processing. Only used when key mode is MC. Otherwise, this parameter is ignored.

input_message_length

Direction	Type
Input	Integer

The length of the message supplied in the *input_message* parameter.

This value must be between 8 and 320, inclusive. For key mode rule VISA, the value must be between 8 and 255, inclusive.

input_message

Direction	Type
Input	String

The message to be secured. Padding is as follows:

For action VISAPIN:
To the left one byte that contains the length of the PIN block (which is 8 bytes) and with an '80' byte followed by a number of '00' bytes until the length is 16 bytes. The output PIN block is 16 bytes for this action. Encryption is done in ECB mode.

For scripting with confidentiality:
The message is only padded if it is not a multiple of eight bytes. EMV padding is used.

For scripting with integrity:
The message is only padded if it is not a multiple of eight bytes. EMV padding is used.

The formatting of the message to the EMV card is the responsibility of the application.

PIN_offset

Direction	Type
Input	Integer

The offset in the message to be secured where the PIN block is to be placed. The first position has offset 0.

This parameter is ignored when the VISAPIN, SMINT, and SMCON keywords are specified.

PIN_format

Direction	Type
Input	String

The 17 bytes consists of two 8 byte fields for input and output PIN block format respectively and a 1 byte for the PAD digit, in case a PAD digit is needed for one of the PIN formats.

The following PIN formats are supported:

For actions SMCONPIN and SMCIPIN:
ISO-0, ISO-1, and ISO-2. Both 8 byte blocks must be fully specified.

For action VISAPIN:
All PIN formats are supported. See *ICSF Application Programmer's Guide*. Only the first 8 bytes (and maybe the PAD digit) are used for this action.

The pad digit must be specified only when specifying “3624“ for the PIN format.

This parameter is ignored when the SMINT and SMCON keywords are specified.

output_message_length

Direction	Type
Input	Integer

On input, the length of the buffer to receive the processed message. The value must be at least as long as the input message plus any padding.

On output, the actual length of the message returned in the *output_message_length* parameter.

This parameter is ignored when keyword SMINT is specified.

output_message

Direction	Type
Input	String

The encrypted message when the action keyword is SMCON, SMCONPIN, SMCONINT, SMCIPIN, or VISAPIN.

The formatting of the message to the EMV card is the responsibility of the application.

mac_length

Direction	Type
Input	Integer

The number of bytes of the MAC that are to be returned in the *mac* parameter. The MAC is returned for secure messaging with integrity. Values 4, 6, and 8 are supported.

This parameter is ignored when keyword SMCOM, SMCONPIN, or VISAPIN is specified.

mac

Direction	Type
Input	String

The MAC value that is calculated for the actions SMINT, SMCONINT, and SMCIPIN.

Note that for actions SMCONINT and SMCIPIN, the result of the integrity protection, the MAC, is output in this parameter and the result of the encryption is in the output message parameter.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Cryptographic services used by EMV Scripting Service

The following CCA cryptographic services are used by EMV Scripting Service:

For action SMINT:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate
- CSNBMGN - MAC Generate

For action SMCON:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate
- CSNBENC - Encipher

For action SMCONPIN:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate
- CSNBSPN - Secure Message for PINs

For action SMCONINT:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate

- CSNBMGN - MAC Generate
- CSNBENC - Encipher

For action SMCIPIN:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate
- CSNBMGN - MAC Generate
- CSNBSPN - Secure Message for PINs

For action VISAPIN:

- CSNBKTB - Key Token Build
- CSNBPCU - VISA PIN change/unblock

The caller does not require authorization to each of these services, only to the EMV Scripting Service. Additionally, the caller must have the required access control points enabled.

Access control points

The following access control points must be enabled to use the EMV Scripting Service:

For action SMINT:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - TDES-XOR
- Diversified Key Generate - SESS-XOR
- Diversified Key Generate - TDESEMV2/TDESEMV4
- MAC Generate

For action SMCON:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - TDES-XOR
- Diversified Key Generate - SESS-XOR
- Diversified Key Generate - TDESEMV2/TDESEMV4
- Encipher - DES

For action SMCONPIN:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - TDES-XOR
- Diversified Key Generate - SESS-XOR
- Diversified Key Generate - TDESEMV2/TDESEMV4
- Secure Messaging for PINs

For action SMCONINT:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - TDES-XOR
- Diversified Key Generate - TDESEMV2/TDESEMV4
- MAC Generate
- Encipher - DES

For action SMCIPIN:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - TDES-XOR

- Diversified Key Generate - TDESEMV2/TDESEMV4
- MAC Generate
- Secure Messaging for PINs

For action VISAPIN:

- PIN change/unblock - change EMV PIN with OPINENC
- PIN change/unblock - change EMV PIN with IPINENC

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 285. EMV Scripting Service required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

EMV Transaction (ARQC/ARPC) Service (CSNBEAC and CSNEEAC)

The EMV Transaction (ARQC/ARPC) Service simplifies EMV Authorization Request Cryptogram (ARQC) and Authorization Response Cryptogram (ARPC) transaction processing. An ARQC is generated by the EMV card upon request from the point of sales terminal to obtain authorization for payment. The ARQC is forwarded across the payment network to the issuer for verification. After the issuer has verified the ARQC, the issuer generates an ARPC (the response). The ARPC is sent back through the payment network to the point of sales terminal to authorize the transaction.

The EMV Transaction (ARQC/ARPC) Service performs the following EMV functions:

- Verifying the Authorization Request Cryptogram (ARQC).
- Generating the Authorization Response Cryptogram (ARPC).
- Both verifying the ARQC and generating the ARPC.

This service can be used in the following specific brand modes:

- Visa
- MasterCard
- EMV

The callable service name for AMODE(64) invocation is CSNEEAC.

Format

```
CALL CSNBEAC(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             issuer_master_key_identifier_length,
             issuer_master_key_identifier,
             issuer_ARPC_master_key_identifier_length,
             issuer_ARPC_master_key_identifier,
             pan_length,
             pan,
             pan_seq_number,
             cryptogram_info_length,
             cryptogram_info,
             atc,
             arc_or_csu,
             arqc,
             arpc,
             unpredictable_number,
             optional_data1_length,
             optional_data1,
             reserved2_length,
             reserved2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 3 and the maximum value is 6.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 286. Rule array keywords for EMV Transaction (ARQC/ARPC) Service

Keyword	Meaning
Algorithm (Required)	
TDES	Specifies the use of Triple-DES.
Action (One required)	
VERARQC	Specifies to verify the input Authorization Request Cryptogram.
GENARPC	Specifies to generate the Authorization Response Cryptogram from the input Authorization Request Cryptogram and Authorization Response Code (ARC).
VERGEN	Specifies to both verify the Authorization Request Cryptogram and generate the Authorization Response Cryptogram.
Key mode (One required). Defines the key derivation mechanism.	

Table 286. Rule array keywords for EMV Transaction (ARQC/ARPC) Service (continued)	
Keyword	**Meaning**
VISA	Specifies to use either the Visa Cryptogram Version 10 (CVN10) or Cryptogram Version 18 (CVN18) key derivation. • For CVN10, the card's master key is used as the session key (the keys are the same for each session). • For CVN18, the card's master key is used to derive the session key. See Visa specification, Appendix D2. Padding is determined by the Cryptogram version used.
MC	Specifies to use the MasterCard M/CHIP 2.1 key derivation. The ATC and an unpredictable number are 3DES encrypted with the card's master key. The card's master key is used when generating the ARPC. EMV padding rules apply.
EMV	Specifies to use the session key derivation as described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.1 (EMV4.1) Book 2, Annex A1.3. Use this key mode for Visa Cryptogram Version 14 and MasterCard M/CHIP 4. EMV padding rules apply.
Control flag (Optional)	
APPANSEQ	Specifies to append the PAN sequence number when the card specific master key is derived. See the descriptions of *pan* and *pan_seq_number*. The default is not to append the PAN sequence number. If PAN length rule PAN-19 is specified, APPANSEQ must also be specified.
Branch Factor (One optional, valid only with key mode EMV). The branching factor is to be used in EMV session key derivation.	
TDESEMV2	Specifies a branch factor of 2 for a height of 16. This is the default.
TDESEMV4	Specifies a branch factor of 4 for a height of 8.
Cryptogram version number (Optional. Only valid with key mode VISA.)	
CVN10	Specifies to use the Visa Cryptogram Version 10 processing method. This is the default for key mode VISA.
CVN18	Specifies to use the Visa Cryptogram Version 18 processing method. This is only allowed for output key type AC.
PAN length (Optional. Only valid with Cryptogram version number CVN18.)	
PAN-16	Specifies that only the rightmost 16 digits of the PAN are used to derive the Unique DEA Keys using the method described in Visa ICC Card Specification V1.6, Appendix D.7. This is the default.
PAN-19	Specifies that the rightmost 19 digits of the PAN are used to derive the Unique DEA Keys using the method described in EMV Book 2, Section A1.4.1, Option B.

issuer_master_key_identifier_length

Direction	Type
Input	Integer

Specifies the length of the *issuer_master_key_identifier* parameter in bytes. The value must be 64.

issuer_master_key_identifier

Direction	Type
Input/Output	String

A 64-byte DES key identifier (either an internal token or key label) for the issuer master key for Application Cryptograms (AC). The issuer master key is the DES key from which the card specific keys are derived and from the card specific keys, the session keys for application cryptograms are derived.

When using the key mode of VISA or EMV, this key is used for both the verification of the ARQC (VERARQC and VERGEN) and the generation of the ARPC (GENARPC and VERGEN).

When using the key mode of MC, this key is used only for the verification of the ARQC (VERARQC and VERGEN). The *issuer_ARPC_master_key_identifier* must supply the ARPC generation key (GENARPC and VERGEN).

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

The key algorithm must be DES, the key type must be DKYGENKY, and the key usage attributes must be DMAC. For VISA CVN10 and EMV, the key subtype must be DKYL0. For VISA CVN18 and MC, the key subtype must be DKYL1.

Note: For MasterCard M/Chip 2.1, you need the issuer master key in two versions: one of each subtype (DKYL1 and DKYL0). If action VERARQC or VERGEN is specified, this key must be the subtype DKYL1 and is used to derive the session key for ARQC verification. The key to be used for ARPC generation (GENARPC and VERGEN) must be specified in the *issuer_ARPC_master_key_identifier* input parameter.

issuer_ARPC_master_key_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *issuer_ARPC_master_key_identifier* parameter in bytes. When the key mode keyword MC is specified, the value must be 64. Otherwise, the value must be zero.

issuer_ARPC_master_key_identifier

Direction	Type
Input/Output	String

The 64-byte CCA DES key identifier (either an internal token or key label) for the issuer master key for Application Response Cryptograms (ARPC) when using the MC key mode. The issuer ARPC master key is the DES key from which a session key for ARPC generation is derived.

Only used when action is GENARPC or VERGEN and key mode is MC, where this key is the issuer master key to be used for deriving the key to use for ARPC generation.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

Key Type Requirements: The key type must be DKYGENKY, the subtype must be DKYL0, and the key usage must specify that a MAC key will be derived (keyword DMAC).

Note: For MasterCard M/Chip 2.1, you need the issuer master key in two versions: one of each subtype (DKYL1 and DKYL0). If the action GENARPC or VERGEN is specified, this key must be the subtype DKYL0 and is used to derive the session key for ARPC generation. The key to be used for ARQC verification (VERARQC and VERGEN) must be specified in the *issuer_master_key_identifier* input parameter.

pan_length

Direction	Type
Input	Integer

Length in bytes of the *pan* parameter. The value must be 10.

pan

Direction	Type
Input	String

The 10-byte EMV card's Primary Account Number. The data must be in compressed numeric format and right justified in a 10-byte field, padded to the left with zeroes. For example, PAN 1234567890 must be provided as x'0000000001234567890'.

This data is used in combination with the PAN sequence number to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

pan_seq_number

Direction	Type
Input	String

The 1-byte sequence number of the EMV card's Primary Account Number. If the APPANSEQ control flag rule array keyword was specified, this PAN sequence number is used in combination with the PAN to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

cryptogram_info_length

Direction	Type
Input	Integer

The length of the cryptogram information supplied in the *cryptogram_info* parameter. This value must be between 1 and 252 inclusive.

cryptogram_info

Direction	Type
Input	String

The cryptogram information on which the ARQC is generated. The data must not be padded on input.

The cryptogram information will be padded as follows:

Key derivation mechanism	Padding
VISA CVN10	Visa ICC Card Specification V1.4.0, Appendix D.2 and D.3.

Key derivation mechanism	Padding
VISA CVN18	Visa ICC Card Specification V1.6, Appendix D.2 and D.4.
MC, EMV	EMV, Book 2, Annex A1.2.

atc

Direction	Type
Input	String

The 2-byte application transaction counter that is used for session key derivation. See the key mode rules for more information on session key derivation.

This parameter must be 2 bytes.

Note: The first byte is the high-order byte and the second byte is the low order byte.

arc_or_csu

Direction	Type
Input	String

For VISA CVN10, MC, and EMV, this parameter must be the 2-byte authorization response code that is used for generating the ARPC.

For VISA CVN18, this parameter must be the 4-byte card status update that is used for generating the ARPC.

This parameter is always read, but the contents are ignored if neither action GENARPC nor VERGEN is specified.

Note: The first byte is the highest-order byte and the last byte is the lowest-order byte.

arqc

Direction	Type
Input	String

The 8-byte authorization request cryptogram received from the payment card.

This parameter must be 8 bytes.

arpc

Direction	Type
Output	String

For VISA CVN10, MC, and EMV, this parameter is the 8-byte authorization response cryptogram to be sent back to the payment card. The ARPC is obtained by enciphering the ARQC XOR-ed with the ARC (See also EMV, Book 2, 8.2).

For VISA CVN18, this parameter is the 4-byte authorization response cryptogram to be sent back to the payment card. The ARPC is obtained by performing the processing described in Visa ICC Card Specification V1.6, Appendix D.4.3.

unpredictable_number

Direction	Type
Input	String

The 4-byte unpredictable number used in the MasterCard M/Chip 2.1 session key derivation scheme.

The data in this field will not be reformatted by the API before use.

This parameter must be 4 bytes.

optional_data1_length

Direction	Type
Input	Integer

Length in bytes of the *optional_data1* parameter.

For VISA CVN10, MC, and EMV or for VISA CVN18 with action VERARQC, the value must be 0.

For VISA CVN18 with action GENARPC or VERGEN, this is the length of the Proprietary Authentication Data used in generating the ARPC. The 'Proprietary Authentication Data included' bit of the CSU (provided in the *arc_or_csu* parameter) determines the valid value or values of this parameter.

- If the bit is set to 1b, the value of this parameter must be between 1 and 8, inclusive.
- If the bit is set to 0b, the value of this parameter must 0.

optional_data1

Direction	Type
Input	String

If the length in the *optional_data1_length* parameter is 0, this field is ignored.

Otherwise, this field contains the Proprietary Authentication Data used in generating the ARPC for VISA CVN18.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Cryptographic services used by EMV Transaction (ARQC/ARPC) Service

The following CCA cryptographic services are used by EMV Transaction (ARQC/ARPC) Service:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate
- CSNBMGN - MAC Generate
- CSNBMVR - MAC Verify

The caller does not require authorization to each of these services, only to the EMV Transaction (ARQC/ARPC) Service. Additionally, the caller must have the required access control points enabled.

Access control points

The following access control points must be enabled to use the EMV Transaction (ARQC/ARPC) Service:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - SESS-XOR
- Diversified Key Generate - TDESEMV2/TDESEMV4
- MAC Generate
- MAC Verify

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 287. EMV Transaction (ARQC/ARPC) Service required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

EMV Verification Functions (CSNBEVF and CSNEEVF)

Provides additional functions used by MasterCard for their EMV cards in addition to application cryptograms and scripting.

This service performs the following EMV scripting functions:

- Verification of data authentication codes.
- Verification of dynamic numbers.

- Decryption of encrypted counters.

This service can be used in the following specific brand modes:

- MasterCard
- EMV

The callable service name for AMODE(64) invocation is CSNEEVF.

Format

```
CALL CSNBEVF(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_identifier_length,
              key_identifier,
              pan_length,
              pan,
              pan_seq_number,
              atc,
              unpredictable_number,
              data_length,
              data,
              reserved1_length,
              reserved1,
              reserved2_length,
              reserved2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 3 and the maximum value is 5.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 288. Rule array keywords for EMV Verification Functions

Keyword	Meaning
Algorithm (Required)	
TDES	Specifies the use of Triple-DES.
Function to be performed (One required)	
DYNVER	Specifies to verify the dynamic number.
DACVER	Specifies to verify the data authentication code.
DECCNT	Specifies to decrypt the encrypted counters.
Key mode (One required). Defines the key derivation mechanism.	
MC	Specifies to use the MasterCard M/CHIP 2.1 mode. The ATC and an unpredictable number is encrypted with the card's master key. The card's master key is used when generating the ARPC. Padding is according to EMV.
EMV	Specifies to use the session key derivation as described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.3. Use this key mode for Visa Cryptogram Version 14 and MasterCard M/CHIP 4. EMV padding rules apply.
Control flag (Optional)	
APPANSEQ	Specifies to append the PAN sequence number when the card specific master key is derived. See the descriptions of *pan* and *pan_seq_number*. The default is not to append the PAN sequence number.
Branch Factor (One optional, valid only with key mode EMV). The branching factor is to be used in EMV session key derivation.	
TDESEMV2	Specifies a branch factor of 2 for a height of 16. This is the default.
TDESEMV4	Specifies a branch factor of 4 for a height of 8.

key_identifier_length

Direction	Type
Input	Integer

Specifies the length of the *key_identifier* parameter in bytes. The value must be 64.

key_identifier

Direction	Type
Input/Output	String

The 64-byte CCA DES key identifier (either an internal token or key label) of the key to perform the specified function.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

Table 289. EMV Verification Functions: Key requirements

Function	Key mode	Key type	Subtype	Key usage
DACVER	N/A	Must be a double-length DATA session key.	N/A	N/A
DECCNT	MC	DKYGENKY	DKYL1	Must specify a DATA key to be derived (DDATA).
DECCNT	EMV	DKYGENKY	DKYL0	Must specify a DATA key to be derived (DDATA).
DYNVER	N/A	DKYGENKY	DKYL0	Must specify a DATA key to be derived (DDATA).

pan_length

Direction	Type
Input	Integer

Length in bytes of the *pan* parameter. The value must be 10.

pan

Direction	Type
Input	String

The 10-byte EMV card's Primary Account Number. The data must be in compressed numeric format and right justified in a 10-byte field, padded to the left with zeroes. For example, PAN 1234567890 must be provided as x'0000000001234567890'.

This data is used in combination with the PAN sequence number to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

pan_seq_number

Direction	Type
Input	String

The 1-byte sequence number of the EMV card's Primary Account Number. If the APPANSEQ control flag rule array keyword was specified, this PAN sequence number is used in combination with the PAN to derive the card's master key. The exact set of rules is described in EMV Integrated Circuit Card Specification for Payment Systems Version 4.2 (EMV4.2) Book 2, Annex A1.4.

If the APPANSEQ keyword was not specified, the PAN sequence number is not appended to the PAN when deriving the card's specific master key.

atc

Direction	Type
Input	String

The 2-byte application transaction counter that is used for session key derivation. See the key mode rules for more information on session key derivation.

This parameter must be 2 bytes.

Note: The first byte is the high-order byte and the second byte is the low order byte.

unpredicatable_number

Direction	Type
Input	String

The 4-byte unpredictable number used in the MasterCard M/Chip 2.1 session key derivation scheme.

The data in this field will not be reformatted before use.

This parameter must be 4 bytes.

data_length

Direction	Type
Input/Output	Integer

Specifies the length of the DATA parameter, in bytes. The value must be 2 for function DACVER and 8 for functions DYNVER and DECCNT.

data

Direction	Type
Input/Output	String

For function DACVER:
The two leftmost bytes are used from the input as the DAC to be verified. On output, the field contains the correct DAC in the case of a mismatch.

For function DYNVER:
All 8 bytes are used. On input, this is the dynamic number to be verified. On output, it contains the correct dynamic number in the case of a mismatch.

For function DECCNT:
All 8 bytes are used. On input, this is the 8-byte encrypted counters. On output, the decrypted counters are returned.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Cryptographic services used by EMV Verification Functions

The following CCA cryptographic services are used by EMV Verification Functions:

For action DACVER:

- CSNBENC - Encipher

For action DYNVER:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate
- CSNBENC - Encipher

For action DECCNT:

- CSNBKTB - Key Token Build
- CSNBDKG - Diversified Key Generate
- CSNBDEC - Decipher

The caller does not require authorization to each of these services, only to the EMV Verification Functions. Additionally, the caller must have the required access control points enabled.

Access control points

The following access control points must be enabled to use the EMV Verification Functions:

For action DACVER:

- Encipher - DES

For action DYNVER:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - SESS-XOR
- Encipher - DES

For action DECCNT:

- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - SESS-XOR
- Diversified Key Generate - TDESEMV2/TDESEMV4
- Decipher - DES

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 290. EMV Verification Functions required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Encrypted PIN Generate (CSNBEPG and CSNEEPG)

The Encrypted PIN Generate callable service formats a PIN and encrypts the PIN block. To generate the PIN, the service uses one of these PIN calculation methods:

- IBM 3624 PIN
- IBM German Bank Pool Institution PIN
- Interbank PIN

To format the PIN, the service uses one of these PIN block formats:

- IBM 3621 format
- IBM 3624 format
- ISO-0 format (same as the ANSI X9.8, VISA-1, and ECI-1 formats)
- ISO-1 format (same as the ECI-4 format)

- ISO-2 format
- ISO-3 format
- ISO-4 format
- IBM 4704 encrypting PINPAD (4704-EPP) format
- VISA 2 format
- VISA 3 format
- VISA 4 format
- ECI-2 format
- ECI-3 format

An enhanced PIN security mode is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the Enhanced PIN Security access control point in the domain role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

The callable service name for AMODE(64) invocation is CSNEEPG.

Format

```
CALL CSNBEPG(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              PIN_generating_key_identifier,
              outbound_PIN_encrypting_key_identifier,
              rule_array_count,
              rule_array,
              PIN_length,
              data_array,
              PIN_profile,
              PAN_data,
              sequence_number,
              encrypted_PIN_block )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is defined in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

PIN_generating_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-generating key to generate the PIN. The key identifier is a variable-length operational key token or key block or the 64-byte key label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte DES key token of key type PINGEN with key usage attribute EPINGEN enabled in the control vector

For X9.143 keys, the identifier is a variable-length key block of a TDES PIN- generation key:

- For process rule keyword IBM-PIN: key usage V1, algorithm T, and mode of use C or G.
- For process rule keywords GBP-PIN and INBK-PIN: key usage V0, algorithm T, and mode of use C or G.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

outbound_PIN_encrypting_key_identifier

Direction	Type
Input	String

The identifier of the PIN-encrypting key to encrypt the PIN block. The key identifier is a variable-length operational key token or key block or the 64-byte key label of an operational token or block in key storage.

When the PIN block format is ISO-4, the key is an AES key:

- For CCA keys, the identifier is a variable-length symmetric key-token must have a token algorithm of AES and a key type of PINPROT. In addition, the key usage fields must have the encryption operation set so that the key can be used for encryption (ENCRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN function usage EPINGEN must be enabled.
- For X9.143 keys, the identifier is a key block of an AES PIN-encrypting key: key usage P0, algorithm A, and mode of use E.

When the PIN block format is other than ISO-4, the key is a DES key:

- For CCA keys, the identifier is a 64-byte DES key token of key type OPINENC with key usage attribute EPINGEN enabled in the control vector.
- For X9.143 keys, the identifier is a key block of a DES PIN-encrypting key: key usage P0, algorithm T, and mode of use E.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 1.

rule_array

Direction	Type
Input	Character String

Keywords that provide control information to the callable service. Each keyword is left-justified in an 8-byte field, and padded on the right with blanks. All keywords must be in contiguous storage. The rule array keywords are shown as follows:

Table 291. Process Rules for the Encrypted PIN Generate Callable Service

Process Rule	Description
GBP-PIN	This keyword specifies the IBM German Bank Pool Institution PIN calculation method is to be used to generate a PIN.
IBM-PIN	This keyword specifies the IBM 3624 PIN calculation method is to be used to generate a PIN.
INBK-PIN	This keyword specifies the Interbank PIN calculation method is to be used to generate a PIN.

PIN_length

Direction	Type
Input	Integer

A integer defining the PIN length for those PIN calculation methods with variable length PINs; otherwise, the variable should be set to zero.

data_array

Direction	Type
Input	String

Three 16-byte character strings, which are equivalent to a single 48-byte string. The values in the data array depend on the keyword for the PIN calculation method. Each element is not always used, but you must always declare a complete data array. The numeric characters in each 16-byte string must be from 1 to 16 bytes in length, uppercase, left-justified, and padded on the right with space characters. Table 292 on page 675 describes the array elements.

Table 292. Array Elements for the Encrypted PIN Generate Callable Service	
Array Element	**Description**
Decimalization_table	Decimalization table for IBM and GBP only. Sixteen characters that are used to map the hexadecimal digits (X'0' to X'F') of the encrypted validation data to decimal digits (X'0' to X'9'). **Note:** If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the domain role, this table must match one of the active decimalization tables in the coprocessors.
Trans_sec_parm	For Interbank only, sixteen digits. Eleven right-most digits of the personal account number (PAN). A constant of 6. One digit key selector index. Three digits of PIN validation data.
Validation_data	Validation data for IBM and IBM German Bank Pool padded to 16 bytes. One to sixteen characters of hexadecimal account data left-justified and padded on the right with blanks.

Table 293 on page 675 lists the data array elements required by the process rule (*rule_array* parameter). The numbers refer to the process rule's position within the array.

Table 293. Array Elements Required by the Process Rule			
Process Rule	**IBM-PIN**	**GBP-PIN**	**INBK-PIN**
Decimalization_table	1	1	
Validation_data	2	2	
Trans_sec_parm			1

PIN_profile

Direction	**Type**
Input	String array

A 24-byte string containing the PIN profile including the PIN block format. See “The PIN profile” on page 609 for additional information.

PAN_data

Direction	**Type**
Input	String

A primary account number (PAN) in character format. The service uses this parameter if the PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 12-byte value in application storage. The information in this parameter will be ignored, but the parameter must be specified.

When using the ISO-0, ISO-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 – 19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Table 294. PAN data structure		
Offset	**Length**	**Description**
0	2	Length of the PAN data field, p.
2	p	10 to 19 bytes of PAN data.
2+p	0-9	Padding.

sequence_number

Direction	Type
Input	Integer

The sequence number used by certain PIN block formats.

- For the 3621 PIN block format, provide a value in the range from 0 to 65535.
- For the 4704-EPP PIN block format, provide a value in the range from 0 to 255.
- For other PIN block formats, provide a value of 99999.

encrypted_PIN_block

Direction	Type
Output	String

The field that receives the 8-byte or 16-byte encrypted PIN block. When the PIN block format is ISO-4, the PIN block will be 16 bytes long. For all other formats, the PIN block will be 8 bytes long.

Restrictions

The format control specified in the PIN profile must be NONE.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the Encrypted PIN Generate service and any key labels specified as input.

Access control points

This table shows the access control points in the domain role that control the function of this service.

Table 295. Required access control points for Encrypted PIN Generate	
Rule array keywords	**Access control point**
IBM-PIN	Encrypted PIN Generate - 3624
GBP-PIN	Encrypted PIN Generate - GBP
INBK-PIN	Encrypted PIN Generate - Interbank

If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the domain role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

An enhanced PIN security mode is available for extracting PINs from a 3621 or 3624 encrypted PIN-block and formatting an encrypted PIN block into IBM 3621 or 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length

of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** access control.

When the **Prohibit translation from DES wrapping to weaker DES wrapping** access control point is enabled in the domain role, this service will fail if the *outbound_PIN_encrypting_key_identifier* is stronger than the *PIN_generating_key_identifier*.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *PIN_profile* parameter is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 296. Encrypted PIN Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require the July 2019 or later licensed internal code (LIC). PIN block format ISO-4 requires the October 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	PIN block format ISO-4 requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 296. Encrypted PIN Generate required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Encrypted PIN Translate (CSNBPTR and CSNEPTR)

Note: This service has been deprecated. New applications should use the Encrypted PIN Translate2 service that is described in "Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)" on page 685 instead of this service.

Use the encrypted PIN translate callable service to reencipher a PIN block from one PIN-encrypting key to another and, optionally, to change the PIN block format, such as the pad digit or sequence number.

The unique-key-per-transaction key derivation for single and double-length keys is available for the encrypted PIN translate service. This support is available for the *input_PIN_encrypting_key_identifier* and the *output_PIN_encrypting_key_identifier* parameters for both REFORMAT and TRANSLAT process rules. The rule_array keyword determines which PIN key or keys are derived key or keys.

The encrypted PIN translate service can be used for unique-key-per-transaction key derivation.

An enhanced PIN security mode is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the Enhanced PIN Security access control point in the domain role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

The enhanced PIN security mode also extracts PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. You must enable the Enhanced PIN Security access control point in the domain role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. As with formatting an encrypted PIN block, no other PIN-block consistency checking will occur.

An enhanced PIN security mode on the CEX3C and later is available to implement restrictions required by the ANSI X9.8 PIN standard. To enforce these restrictions, you must enable the following control points in the domain role.

- ANSI X9.8 PIN - Enforce PIN block restrictions
- ANSI X9.8 PIN - Allow modification of PAN
- ANSI X9.8 PIN - Allow only ANSI PIN blocks

The callable service name for AMODE(64) invocation is CSNEPTR.

Format

```
CALL CSNBPTR(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             input_PIN_encrypting_key_identifier,
             output_PIN_encrypting_key_identifier,
             input_PIN_profile,
             PAN_data_in,
             PIN_block_in,
             rule_array_count,
```

```
            rule_array,
            output_PIN_profile,
            PAN_data_out,
            sequence_number,
            PIN_block_out )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

input_PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The input PIN-encrypting key (IPINENC) for the *PIN_block_in* parameter specified as a 64-byte internal key token or a key label. If keyword UKPTIPIN, UKPTBOTH, DUKPT-IP or DUKPT-BH is specified in the *rule_array*, then the *input_PIN_encrypting_key_identifier* must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

output_PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The output PIN-encrypting key (OPINENC) for the *PIN_block_out* parameter specified as a 64-byte internal key token or a key label. If keyword UKPTOPIN, UKPTBOTH, DUKPT-OP or DUKPT-BH is

specified in the *rule_array*, then the *output_PIN_encrypting_key_identifier* must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

input_PIN_profile

Direction	Type
Input	Character String

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile depending on whether the PIN block is being enciphered or deciphered by the callable service. See "The PIN profile" on page 609 for additional information.

The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the Encrypted PIN translate callable service with a process rule (*rule_array* parameter) of REFORMAT. If the process rule is TRANSLAT, the pad digit is ignored.

PAN_data_in

Direction	Type
Input	Character String

The personal account number (PAN) if the process rule (*rule_array* parameter) is REFORMAT and the input PIN format is ISO-0 or VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.

For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit.

For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

PIN_block_in

Direction	Type
Input	String

The 8-byte enciphered PIN block that contains the PIN to be translated.

rule_array_count

Direction	Type
Input	Integer

The number of process rules specified in the *rule_array* parameter. The value may be 1, 2 or 3.

rule_array

Direction	Type
Input	Character String

The process rule for the callable service.

Table 297. Keywords for Encrypted PIN Translate	
Keyword	**Meaning**
Processing Rules (required)	
REFORMAT	Changes the PIN format, the contents of the PIN block, and the PIN-encrypting key.
TRANSLAT	Changes the PIN-encrypting key only. It does not change the PIN format and the contents of the PIN block.
PIN Block Format and PIN Extraction Method (optional)	See "PIN block format and PIN extraction method keywords" on page 610 for additional information and a list of PIN block formats and PIN extraction method keywords. **Note:** If a PIN extraction method is not specified, the first one listed in Table 250 on page 610 for the PIN block format will be the default.
DUKPT Keywords - Single length key derivation (optional)	
UKPTIPIN	The *input_PIN_encrypting_key_identifier* is derived as a single length key. The *input_PIN_encrypting_key_identifier* must be a KEYGENKY key with the UKPT usage bit enabled. The *input_PIN_profile* must be 48 bytes and contain the key serial number.
UKPTOPIN	The *output_PIN_encrypting_key_identifier* is derived as a single length key. The *output_PIN_encrypting_key_identifier* must be a KEYGENKY key with the UKPT usage bit enabled. The *output_PIN_profile* must be 48 bytes and contain the key serial number.
UKPTBOTH	Both the *input_PIN_encrypting_key_identifier* and the *output_PIN_encrypting_key_identifier* are derived as a single length key. Both the *input_PIN_encrypting_key_identifier* and the *output_PIN_encrypting_key_identifier* must be KEYGENKY keys with the UKPT usage bit enabled. Both the *input_PIN_profile* and the *output_PIN_profile* must be 48 bytes and contain the respective key serial number.
DUKPT Keywords - Double length key derivation (optional) - requires May 2004 or later version of Licensed Internal Code (LIC)	
DUKPT-IP	The *input_PIN_encrypting_key_identifier* is derived as a double length key. The *input_PIN_encrypting_key_identifier* must be a KEYGENKY key with the UKPT usage bit enabled. The *input_PIN_profile* must be 48 bytes and contain the key serial number.
DUKPT-OP	The *output_PIN_encrypting_key_identifier* is derived as a double length key. The *output_PIN_encrypting_key_identifier* must be a KEYGENKY key with the UKPT usage bit enabled. The *output_PIN_profile* must be 48 bytes and contain the key serial number.

Table 297. Keywords for Encrypted PIN Translate (continued)	
Keyword	**Meaning**
DUKPT-BH	Both the *input_PIN_encrypting_key_identifier* and the *output_PIN_encrypting_key_identifier* are derived as a double length key. Both the *input_PIN_encrypting_key_identifier* and the *output_PIN_encrypting_key_identifier* must be KEYGENKY keys with the UKPT usage bit enabled. Both the *input_PIN_profile* and the *output_PIN_profile* must be 48 bytes and contain the respective key serial number.

output_PIN_profile

Direction	Type
Input	Character String

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile, depending on whether the PIN block is being enciphered or deciphered by the callable service.

- If you choose the REFORMAT processing rule in the *rule_array* parameter, the input PIN profile and output PIN profile can have different PIN block formats.
- If you specify UKPTOPIN or UKPTBOTH in the *rule_array* parameter, then the *output_PIN_profile* is extended to a 48-byte field and must contain the current key serial number. See "The PIN profile" on page 609 for additional information.
- If you specify DUKPT-OP or DUKPT-BH in the *rule_array* parameter, then the *output_PIN_profile* is extended to a 48-byte field and must contain the current key serial number. See "The PIN profile" on page 609 for additional information.

PAN_data_out

Direction	Type
Input	Character String

The personal account number (PAN) if the process rule (*rule_array* parameter) is REFORMAT and the output PIN format is ISO-0 or VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.

For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit.

For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

sequence_number

Direction	Type
Input	Integer

The sequence number if the process rule (*rule_array* parameter) is REFORMAT and the output PIN block format is 3621 or 4704-EPP only. Specify the integer value 99999. Otherwise, this parameter is ignored.

PIN_block_out

Direction	Type
Output	String

The 8-byte output PIN block that is reenciphered.

Restrictions

PAD digit restricted to non-decimal digit when Enhanced PIN Security access control point is enabled and if the output PIN profile specifies 3624 or 3621 as the PIN-block format.

The format control specified in the PIN profile must be NONE.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Some PIN block formats are known by several names. This table shows the additional names.

Table 298. Additional Names for PIN Formats

PIN Format	Additional Name
ISO-0	ANSI X9.8, VISA format 1, ECI format 1
ISO-1	ECI format 4

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 299. Required access control points for Encrypted PIN Translate

Processing rule	Access control point
TRANSLAT	Encrypted PIN Translate - Translate
REFORMAT	Encrypted PIN Translate - Reformat

If any of the Unique Key per Transaction rule array keywords are specified, the **DUKPT - PIN Verify, PIN Translate** access control point must be enabled.

An enhanced PIN security mode is available for extracting PINs from a 3621 or 3624 encrypted PIN-block and formatting an encrypted PIN block into IBM 3621 or 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** access control.

When the **Encrypted PIN Translate - Translate PIN Check** access control is enabled, checking of the PIN block is performed. The checking is the same as the checking done when the REFORMAT keyword is specified.

When the **General ISO PIN Error Mode** access control is enabled, the return code will be a general PIN block error (return code 8 reason code 2514) instead of some of the PIN block errors return code. The use of a general return code can prevent the abuse of PIN processing error messages due to information leakage derived from the return code reason codes returned under various conditions. For more details, see “PIN block error processing mode” on page 609.

Three additional access controls should be considered: **ANSI X9.8 PIN - Enforce PIN block restrictions, ANSI X9.8 PIN - Allow modification of PAN, and ANSI X9.8 PIN - Allow only ANSI PIN blocks**. These three access controls affect how PIN processing is performed as described below. The access controls will affect this and other PIN processing services if enabled.

1. Enable the **ANSI X9.8 PIN - Enforce PIN block restrictions** access control to apply additional restrictions to PIN processing as follows:
   - Do not translate or reformat a non-ISO PIN block into an ISO PIN block. Specifically, do not allow an IBM 3624 PIN-block format in the *output_PIN_profile* variable when the PIN-block format in the *input_PIN_profile* variable is not IBM 3624.
   - Constrain use of ISO-2 PIN blocks to offline PIN verification and PIN change operations in integrated circuit card environments only. Specifically, do not allow ISO-2 input or output PIN blocks.
   - Do not translate or reformat a PIN-block format that includes a PAN into a PIN-block format that does not include a PAN. Specifically, do not allow an ISO-1 PIN-block format in the *output_PIN_profile* variable when the PIN-block format in the *input_PIN_profile* variable is ISO-0, ISO-3, or ISO-4.
   - Do not allow a change of PAN data. Specifically, when performing translations between PIN block formats that both include PAN data, do not allow the *input_PAN_data* and *output_PAN_data* variables to be different from the PAN data enciphered in the input PIN-block.
2. Enable the **ANSI X9.8 PIN - Allow modification of PAN** access control to override the restriction to not allow a change of PAN data. This override is applicable only when either the **ANSI X9.8 PIN - Enforce PIN block restrictions** control, the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control, or both are enabled. This override is to support account number changes in issuing environments. The **ANSI X9.8 PIN - Allow modification of PAN** control has no effect if neither the **ANSI X9.8 PIN - Enforce PIN block restrictions** control nor the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control is enabled. This rule does not apply for CSNBPTRE, and PAN changes are not allowed.
3. Enable the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control to apply a more restrictive variation of the **ANSI X9.8 PIN - Enforce PIN block restrictions** control. In addition to the previously described restrictions of the **ANSI X9.8 PIN - Enforce PIN block restrictions** control, this control also restricts the *input_PIN_profile* and the *output_PIN_profile* to contain only ISO-0, ISO-1, ISO-3, and ISO-4 PIN block formats. Specifically, the IBM 3624 PIN-block format is not allowed with this command. The **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control overrides the **ANSI X9.8 PIN - Enforce PIN block restrictions** control.

When the **Prohibit translation from DES wrapping to weaker DES wrapping** access control point is enabled in the domain role, this service will fail if the *input_PIN_encrypting_key_identifier* is stronger than the *output_PIN_encrypting_key_identifier*.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *input_PIN_profile* and *output_PIN_profile* parameters is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 300. Encrypted PIN Translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.

Table 300. Encrypted PIN Translate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Encrypted PIN Translate2 (CSNBPTR2 and CSNEPTR2)

Use the Encrypted PIN Translate2 callable service to reencipher a PIN block from one PIN-encrypting key to another and, optionally, to change the PIN block format, such as the pad digit or sequence number.

This callable service performs all of the function that the Encrypted PIN Translate service performs with the addition of ISO-4 PIN block support and PAN change authentication support.

The derived unique-key-per-transaction (DUKPT) algorithm is available. Both DES-DUKPT (ANSI x9.24-1 2007) and AES-DUKPT (ANSI x9.24-3 2017) are supported. This support is available for the *input_PIN_encrypting_key_identifier* and the *output_PIN_encrypting_key_identifier* parameters for both REFORMAT and TRANSLAT process rules. The *rule_array* keyword determines which PIN key or PIN keys are derived keys.

The callable service operates in one of two modes, either translate or reformat:

- In translate mode, the callable service decrypts a PIN block using an input key that you supply or that is derived from other information that you supply. The cleartext information is then encrypted using an output key that you supply or that is derived from other information that you supply. The cleartext is not examined.
- In reformat mode, the callable service performs the translate-mode functions and, in addition, processes the cleartext information. Following rules that you specify, the PIN is recovered from the input cleartext PIN-block and formatted into an output PIN-block for encryption.

PAN change authentication allows the caller to specify an authentication value, additional authentication data, and a MAC verify key, which is used to verify the authentication value. If the verification passes, the PAN change request is allowed. The AES CMAC method is used to generate the MAC.

When the PAN format rule is PANAUTAS, the PAN data must be ASCII character data.

```
Authentication value = CMAC( (Old PAN) || (New PAN) || (Optional additional authentication
data) )
```

When the PAN format rule is PANAUTI4, the PAN data is formatted according to ISO 9564-1 Plain text primary account number field format.

```
Authentication value = CMAC( (Old PAN) ISO 9564 FMT || (New PAN) ISO 9564 FMT || (Optional
additional authentication data) )
```

The PAN change authentication support is only allowed when input PIN block format and the output PIN block format are both ISO-4 and an appropriate access control is enabled in the domain role.

PAN change authentication support is only allowed when the input and output PIN-block formats are both ISO-4 and the **Encrypted PIN Translate2 – Permit ISO-4 Reformat w/ PAN Chg** access control is enabled. Certain restrictions apply when selecting a PAN change request. Whenever **Encrypted PIN Translate2 – Permit ISO-4 Reformat w/ PAN Chg** is enabled in the active role, only authenticated PAN change requests are allowed. No other REFORMAT requests are allowed if **Encrypted PIN Translate2 – Permit ISO-4 Reformat w/ PAN Chg** is enabled.

The callable service name for AMODE(64) invocation is CSNEPTR2.

Format

```
CALL CSNBPTR2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             input_PIN_encrypting_key_identifier_length,
             input_PIN_encrypting_key_identifier,
             output_PIN_encrypting_key_identifier_length,
             output_PIN_encrypting_key_identifier,
             authentication_key_identifier_length,
             authentication_key_identifier,
             input_PIN_profile_length,
             input_PIN_profile,
             input_PAN_data_length,
             input_PAN_data,
             input_PIN_block_length,
             input_PIN_block,
             output_PIN_profile_length,
             output_PIN_profile,
             output_PAN_data_length,
             output_PAN_data,
             authentication_data_length,
             authentication_data,
             output_PIN_block_length,
             output_PIN_block,
             reserved1_length,
             reserved1,
             reserved2_length,
             reserved2,
             reserved3_length,
             reserved3 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords that you supplied in the *rule_array* parameter. Values are 1 through 5.

rule_array

Direction	Type
Input	Character String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 301. Keywords for Encrypted PIN Translate2

Keyword	Meaning
Mode (one, required)	
REFORMAT	Specifies that either the PIN-block format or the PIN-block encryption, or both, are to be changed. If the PIN-extraction method is not chosen by default, another element in the rule array must specify one of the keywords that indicates a PIN-extraction method.
TRANSLAT	Specifies that only PIN-block encryption is changed. The first 24 bytes of PIN profiles are ignored for all formats except ISO-4. The input PIN profile must be supplied for ISO-4 PIN blocks.
PAN change option (One, optional) Only valid with REFORMAT and for ISO-4 PIN block processing.	
PAN-CHG	Specifies that a PAN change has been requested.

Table 301. Keywords for Encrypted PIN Translate2 (continued)	
Keyword	**Meaning**
PAN format option (one, optional) Only valid with REFORMAT and for ISO-4 PIN block processing.	
PANAUTAS	Specifies to format the PAN data using the original ASCII format when verifying the CMAC of the authentication data.
PANAUTI4	Specifies to format the PAN data according to ISO 9564-1 Plain text primary account number field format when verifying the CMAC of the authentication data.
DES DUKPT (one, optional). Valid for DES keys only. See Table 304 on page 697 for valid DUKPT keyword combinations.	
UKPTIPIN	Specifies the use of DUKPT input-key derivation and PIN-block decryption, Single-DES method. This keyword cannot be specified with ADUKPTBH or ADUKPTIP.
UKPTOPIN	Specifies the use of DUKPT output-key derivation and PIN-block encryption, Single-DES method. This keyword cannot be specified with ADUKPTBH or ADUKPTOP.
UKPTBOTH	Specifies the use of DUKPT key-derivation and PIN-block ciphering for both input and output processing, Single-DES method. This keyword cannot be specified with any of the keywords in the AES DUKPT group.
DUKPT-IP	Specifies the use of DUKPT input-key derivation and PIN-block decryption, Triple-DES method. This keyword cannot be specified with ADUKPTBH or ADUKPTIP.
DUKPT-OP	Specifies the use of DUKPT output-key derivation and PIN-block encryption, Triple-DES method. This keyword cannot be specified with ADUKPTBH or ADUKPTOP.
DUKPT-BH	Specifies the use of DUKPT key-derivation and PIN-block ciphering for both input and output processing, Triple-DES method. This keyword cannot be specified with any of the keywords in the AES DUKPT group.
AES DUKPT (one, optional). Valid for AES keys only. See Table 304 on page 697 for valid DUKPT keyword combinations.	
ADUKPTBH	Specifies the use of DUKPT key-derivation and PIN-block ciphering for both input and output processing. AES DUKPT method. This keyword cannot be specified with any of the keywords in the DES DUKPT group.
ADUKPTIP	Specifies the use of DUKPT key-derivation and PIN-block ciphering for input processing. AES DUKPT method. This keyword cannot be specified with UKPTIPIN, UKPTBOTH, DUKPT-IP, or DUKPT-BH
ADUKPTOP	Specifies the use of DUKPT key-derivation and PIN-block ciphering for output processing. AES DUKPT method. This keyword cannot be specified with UKPTOPIN, UKPTBOTH, DUKPT-OP, or DUKPT-BH.

Table 301. Keywords for Encrypted PIN Translate2 (continued)	
Keyword	**Meaning**
PIN-extraction method (one, optional). See "PIN block format and PIN extraction method keywords" on page 610 for additional information and a list of PIN block formats and PIN extraction method keywords. **Note:** If a PIN extraction method is not specified, the first one listed in Table 250 on page 610 for the PIN block format will be the default.	

input_PIN_encrypting_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_PIN_encrypting_key_identifier* parameter.

If the *input_PIN_encrypting_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

input_PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to decrypt the input PIN block or the key-generating key to be used to derive the key to decrypt the input PIN block. The key identifier is a variable-length operational key token or key block or the key label of an operational token or block in key storage.

For CCA key tokens:

For DES keys

If you do not use the DUKPT process or specified the UKPTOPIN or DUKPT-OP rule array keyword, the key is a DES 64-byte PIN block encrypting key of type IPINENC and has one or both of the TRANSLAT and REFORMAT key usage bits enabled as appropriate for the requested mode.

If you use the DUKPT process for the input PIN block by specifying the UKPTIPIN, UKPTBOTH, DUKPT-IP, or DUKPT-BH rule array keyword, the key is the DES 64-byte base derivation key of KEYGENKY key type with key usage UKPT enabled.

For AES keys (ISO-4 PIN blocks)

If you do not use the DUKPT process or specified the ADUKPTOP rule array keyword, this key is an AES variable-length PIN block encrypting key of type PINPROT with one or both of the PINXLATE and REFORMAT key usage field bits enabled as appropriate for the requested mode. The key usage fields must have the decryption operation set so that the key can be used for decryption (DECRYPT), but not encryption, and the encryption mode of Cipher Block Chaining (CBC) must be specified.

If you use the DUKPT process for the input PIN block by specifying the ADUKPTIP or ADUKPTBH rule array keywords and the *input_PIN_profile* contains AES-DUKPT derivation data, this key is an AES variable-length DKYGENKY key with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1.

For X9.143 key blocks:

For DES keys

If you do not use the DUKPT process or specified the UKPTOPIN or DUKPT-OP rule array keyword, the key is TDES PIN-encrypting key (key usage P0, algorithm T, and mode of use D).

If you use the DUKPT process for the input PIN block by specifying the UKPTIPIN, UKPTBOTH, DUKPT-IP, or DUKPT-BH rule array keyword, the key is the TDES base derivation key (key usage B0, algorithm T, and mode of use X).

For AES keys (ISO-4 PIN blocks)

If you do not use the DUKPT process or specified the ADUKPTOP rule array keyword, this key is an AES PIN-encrypting key (key usage P0, algorithm A, and mode of use D).

If you use the DUKPT process for the input PIN block by specifying the ADUKPTIP or ADUKPTBH rule array keywords and the *input_PIN_profile* contains AES-DUKPT derivation data, this key is an AES base derivation key (key usage B0, algorithm A, and mode of use X).

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

output_PIN_encrypting_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *output_PIN_encrypting_key_identifier* parameter.

If the *output_PIN_encrypting_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

output_PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to encrypt the output PIN block or the key-generating key to be used to derive the key to encrypt the output PIN block. The key identifier is an operational token or the key label of an operational token in key storage.

For CCA key tokens:

For DES keys

If you do not use the DUKPT process or specified the UKPTIPIN or DUKPT-IP rule array keyword, the key is a DES 64-byte PIN block encrypting key of type OPINENC and has one or both of the TRANSLAT and REFORMAT key usage bits enabled as appropriate for the requested mode.

If you use the DUKPT process for the output PIN-block by specifying the UKPTOPIN, UKPTBOTH, DUKPT-OP, or DUKPT-BH rule array keyword, the key is the DES 64-byte base derivation key of KEYGENKY key type with key usage UKPT enabled.

For AES keys (ISO-4 PIN block)

If you do not use the DUKPT process or you specify the ADUKPTIP rule array keyword, this key is an AES variable-length PIN block encrypting key of type PINPROT with one or both of the PINXLATE and REFORMAT key usage field bits enabled as appropriate for the requested mode. The key usage fields must have the encryption operation set so that the key can be used for encryption (ENCRYPT), but not decryption, and the encryption mode of Cipher Block Chaining (CBC) must be specified.

If you use the DUKPT process for the output PIN block by specifying the ADUKPTOP or ADUKPTBH rule array keywords and the *output_PIN_profile* contains AES-DUKPT derivation data, this key is an AES variable-length DKYGENKY key with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1.

For X9.143 key blocks:

For DES keys

If you do not use the DUKPT process or specified the UKPTIPIN or DUKPT-IP rule array keyword, the key is TDES PIN-encrypting key (key usage P0, algorithm T, and mode of use E).

If you use the DUKPT process for the input PIN block by specifying the UKPTOPIN, UKPTBOTH, DUKPT-OP, or DUKPT-BH rule array keyword, the key is the TDES base derivation key (key usage B0, algorithm T, and mode of use X).

For AES keys (ISO-4 PIN blocks)

If you do not use the DUKPT process or specified the ADUKPTIP rule array keyword, this key is an AES PIN-encrypting key (key usage P0, algorithm A, and mode of use E).

If you use the DUKPT process for the input PIN block by specifying the ADUKPTOP or ADUKPTBH rule array keywords and the *output_PIN_profile* contains AES-DUKPT derivation data, this key is an AES base derivation key (key usage B0, algorithm A, and mode of use X).

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

authentication_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *authentication_key_identifier* parameter.

When the PAN change option keyword PAN-CHG is not specified, the value must be zero.

When the PAN change option keyword PAN-CHG is specified, and the *authentication_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

authentication_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify the CMAC in the *authentication_data* parameter. The key identifier is an operational token or the key label of an operational token in key storage. When *authentication_key_identifier_length* is zero, this parameter is ignored.

For CCA keys, the key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, VERIFY. When **Encrypted PIN Translate2 - Permit ISO-4 to ISO-4 PTR2AUTH** is enabled, the AES MAC key must have key usage VERIFY, CMAC, and PTR2AUTH enabled.

For X9.143 keys, the identifier is a variable-length key block of an AES MAC key: key usage M6, algorithm A, and mode of use V.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

input_PIN_profile_length

Direction	Type
Input	Integer

Specifies the length of the *input_PIN_profile* parameter in bytes.

Table 302. Supported Encrypted PIN Translate2 PIN profile lengths	
Pin profile	**Length**
PIN-block format only.	24
PIN-block format and CKSN extension used for DES-DUKPT.	48
PIN-block format and single block of derivation data extension used for AES-DUKPT.	44

input_PIN_profile

Direction	Type
Input	String

The 24, 44, or 48 byte input PIN profile. The profile consists of three 8-byte character strings with information defining the input PIN-block format and optionally followed by either an additional 24 bytes containing the input CKSN extension or an additional 20 bytes containing the input derivation data extension. See "The PIN profile" on page 609 for additional information.

If the rule array keyword UKPTBOTH or UKPTIPIN is specified, CKSN extension must be included in the *input_PIN_profile*. Single-DES DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block.

If the rule array keyword DUKPT-BH or DUKPT-IP is specified, CKSN extension must be included in the *input_PIN_profile*. The Triple-DES DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block.

If the rule array keyword ADUKPTBH or ADUKPTIP is specified, the AES-DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block when the derivation data extension is included in the *input_PIN_profile*. See Table 684 on page 1719 for the layout of the AES-DUKPT derivation data extension. The algorithm indicator must be set to X'0000' (2-key TDES), X'0001' (3-key TDES), X'0002' (AES-128), X'0003' (AES-192), or X'0004' (AES-256). The key usage indicator must be set to X'1000' (PIN Encryption).

input_PAN_data_length

Direction	Type
Input	Integer

Specifies the length of the *input_PAN_data* parameter in bytes.

When the TRANSLAT mode rule is specified, the value must be 0 except when the PIN block format is ISO-4. When the format is ISO-4, the value must be 10 - 19.

When the REFORMAT keyword is specified:

- If the input PIN block format is ISO-0, ISO-3, or VISA-4, the value must be 12.
- If the input PIN block format is ISO-4, the value must be 10 - 19.
- Otherwise, the value must be 0.

input_PAN_data

Direction	Type
Input	String

The primary account number (PAN) data used to format the input PIN block. This service uses this data to recover the PIN from the PIN block when the format uses the PAN data.

When the TRANSLAT mode rule is specified, this parameter is ignored except when the PIN block format is ISO-4. When the format is ISO-4, this parameter is required.

When the REFORMAT keyword is specified and the input PIN profile specifies ISO-0, ISO-3, ISO-4, or VISA-4 for the PIN block format, this parameter is required.

When the profile specifies the ISO-0, ISO-3, or VISA-4 block format, the 12 rightmost digits of the PAN, excluding the check digit, are used to format the output PIN block.

When the PIN block format is ISO-4, the PAN is used to format the output PIN block. The PAN check digit is included in the formation. The PAN check digit is excluded in the test used to determine if the PAN of an ISO-4 PIN block is equivalent to a PAN that is in a non-ISO format 4 PIN block.

input_PIN_block_length

Direction	Type
Input	Integer

Specifies the length of the *input_PIN_block* parameter in bytes. The value must be 8 for DES PIN-encrypting key and 16 for AES PIN-encrypting key.

input_PIN_block

Direction	Type
Input	String

The 8-byte or 16-byte enciphered PIN block that contains the PIN to be processed.

output_PIN_profile_length

Direction	Type
Input	Integer

Specifies the length of the *output_PIN_profile* parameter in bytes. See Table 302 on page 692 for the supported PIN profile lengths.

output_PIN_profile

Direction	Type
Input	String

The 24, 44, or 48 byte output PIN profile. The profile contains three 8-byte character strings with information defining the PIN-block format and optionally followed by either an additional 24 bytes containing the input CKSN extension or an additional 20 bytes containing the input derivation data extension. See “The PIN profile” on page 609 for additional information.

If the rule array keyword UKPTBOTH or UKPTOPIN is specified, CKSN extension must be included in the *output_PIN_profile*. Single-DES DUKPT algorithm will be used to derive the DUKPT key used to encrypt the output PIN block.

If the rule array keyword DUKPT-BH or DUKPT-OP is specified, CKSN extension must be included in the *output_PIN_profile*. The Triple-DES DUKPT algorithm will be used to derive the DUKPT key used to encrypt the output PIN block.

If the rule array keyword ADUKPTBH or ADUKPTOP is specified, the AES-DUKPT algorithm will be used to derive the DUKPT key used to encrypt the output PIN block when the derivation data extension is included in the *output_PIN_profile*. See Table 684 on page 1719 for the layout of the AES-DUKPT derivation data extension. The algorithm indicator must be set to X'0000' (2-key TDES), X'0001' (3-key TDES), X'0002' (AES-128), X'0003' (AES-192), or X'0004' (AES-256). The key usage indicator must be set to X'1000' (PIN Encryption).

When the mode rule is TRANSLAT, the first 24 bytes of this parameter are ignored.

When the mode rule is REFORMAT in the rule array, the input PIN profile and output PIN profile can have different PIN block formats.

When UKPTOPIN or UKPTBOTH is specified, the parameter is extended to a 48-byte field and must contain the output current key serial number.

When DUKPT-OP or DUKPT-BH is specified, the parameter is extended to a 48-byte field and must contain the output current key serial number.

output_PAN_data_length

Direction	Type
Input	Integer

Specifies the length of the *output_PAN_data* parameter in bytes.

When the mode rule is TRANSLAT, this parameter is ignored.

When the mode rule is REFORMAT:

- If the output PIN block format is ISO-0, ISO-3, or VISA-4, the value must be 12.
- If the format is ISO-4, the value must be 10 - 19.
- Otherwise, the value must be zero.

output_PAN_data

Direction	Type
Input	String

The primary account number (PAN) data used to format the output PIN block. When the *output_PAN_data_length* is zero, this parameter is ignored. When the mode rule is TRANSLAT, the parameter is ignored.

This service uses this data to format the output PIN block if you specify the REFORMAT keyword and the output PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the PIN block format.

For the ISO-4 format, the PAN-CHG rule must be specified in the rule array and the authentication data must be supplied.

When using the ISO-0, ISO-3, or VISA-4 PIN-block format, use the 12 rightmost digits of PAN, excluding the check digit. When using the ISO-4 PIN-block format, the PAN check digit is included in the formation of the PIN blocks.

authentication_data_length

Direction	Type
Input	Integer

Specifies the length of the *authentication_data* parameter in bytes. When the PAN change option keyword PAN-CHG is specified, the value must be 12 – 276. Otherwise, the value must be zero.

authentication_data

Direction	Type
Input	String

The MAC that must be verified to authorize a PAN change operation. When the *authentication_data_length* is zero, this parameter is ignored.

The parameter contains a length-value structure with the following format:

Offset	Length	Description
0	2	Length of the CMAC, *n*. The CMAC can be 8 – 16 bytes long,
2	*n*	CMAC
2 + *n*	2	Length of the optional additional authentication data.
4 + *n*	0 - 256	Optional additional authentication data.

The additional authentication data length can be 0. If a PAN change is requested, the CMAC length can be 8 to 16 bytes. The service creates a CMAC over the old PAN data, new PAN data, and additional authentication data.

Note: The PAN data must be ASCII character data when calculating the authentication value.

This MAC is compared to the CMAC in this parameter for length specified. If the values match, the PAN change request is honored.

output_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length of the *output_PIN_block* parameter in bytes. The value must be at least 8 bytes for DES PIN blocks and 16 for AES PIN blocks. On output, the value is updated with the actual number of bytes returned.

output_PIN_block

Direction	Type
Output	String

The 8 or 16 byte reformatted PIN block.

reserved1_length

Direction	Type
Input/Output	Integer

Length of the *reserved1* parameter in bytes. The value must be 0.

reserved1

Direction	Type
Input	String

This parameter is ignored.

reserved2_length

Direction	Type
Input	Integer

Length of the *reserved2* parameter in bytes. The value must be 0.

reserved2

Direction	Type
Input	String

This parameter is ignored.

reserved3_length

Direction	Type
Input	Integer

Length of the *reserved3* parameter in bytes. The value must be 0.

reserved3

Direction	Type
Input	String

This parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

The inbound and outbound PIN encrypting key usage requirements are defined in the table below:

Table 303. Key usage requirements for PIN encrypting keys

Input PIN format	Output PIN format	Authenticated PAN-change allowed	Inbound key (key usage)	Outbound key (key usage)	Authentication key (key usage)
ISO-0	ISO-4	No	DES IPINENC (REFORMAT)	AES PINPROT (ENCRYPT, ISO-4, REFORMAT)	N/A
ISO-1	ISO-4	No	DES IPINENC (REFORMAT)	AES PINPROT (ENCRYPT, ISO-4, REFORMAT)	N/A
ISO-1	ISO-4	No	DES IPINENC (REFORMAT)	AES PINPROT (ENCRYPT, REFORMAT, ISO-4, RFMT1TO4)	N/A
ISO-4	ISO-0	No	AES PINPROT (DECRYPT, ISO-4, REFORMAT)	DES OPINENC (REFORMAT)	N/A
ISO-4	ISO-1	No	AES PINPROT (DECRYPT, ISO-4, REFORMAT)	DES OPINENC (REFORMAT)	N/A

Table 303. Key usage requirements for PIN encrypting keys (continued)

Input PIN format	Output PIN format	Authenticated PAN-change allowed	Inbound key (key usage)	Outbound key (key usage)	Authentication key (key usage)
ISO-4	ISO-1	No	AES PINPROT (DECRYPT, ISO-4, REFORMAT, RFMT4TO1)	DES OPINENC (REFORMAT)	N/A
ISO-4	ISO-4	No	AES PINPROT (DECRYPT, ISO-4, PINXLATE)	AES PINPROT (ENCRYPT, ISO-4, PINXLATE)	N/A
ISO-4	ISO-4	Yes	AES PINPROT (DECRYPT, ISO-4, PINXLATE)	AES PINPROT (ENCRYPT, ISO-4, PINXLATE)	AES MAC (CMAC, VERIFY) or (CMAC, VERIFY, PTR2AUTH)

Table 304. Valid encrypted PIN Translate2 DUKPT keyword combinations

DUKPT keyword combination	Input PIN encrypting key	Output PIN encrypting key
UKPTIPIN ADUKPTOP	Single DES DUKPT	AES DUKPT
DUKPT-IP ADUKPTOP	Triple DES DUKPT	AES DUKPT
ADUKPTIP UKPTOPIN	AES DUKPT	Single DES DUKPT
ADUKPTIP DUKPT-OP	AES DUKPT	Triple DES DUKPT
UKPTBOTH	Single DES DUKPT	Single DES DUKPT
DUKPT-BH	Triple DES DUKPT	Triple DES DUKPT
ADUKPTBH	AES DUKPT	AES DUKPT
UKPTIPIN	Single DES DUKPT	Static
UKPTOPIN	Static	Single DES DUKPT
DUKPT-IP	Triple DES DUKPT	Static
DUKPT-OP	Static	Triple DES DUKPT
ADUKPTIP	AES DUKPT	Static
ADUKPTOP	Static	AES DUKPT

Access control points

The following table shows the access control points in the domain role that control the function of this service. When the input or output PIN format in the PIN profile is ISO-4, the **Encrypted PIN Translate2 – REFORMAT/TRANSLATE** access controls are used. When neither the input nor output PIN format in the PIN profile is ISO-4, the **Encrypted PIN Translate – REFORMAT/TRANSLATE** access controls are used.

Table 305. Required access control points for Encrypted PIN Translate2	
Processing rule	**Access control point**
TRANSLAT	• Encrypted PIN Translate - TRANSLAT • Encrypted PIN Translate2 - TRANSLAT
REFORMAT	• Encrypted PIN Translate - REFORMAT • Encrypted PIN Translate2 - REFORMAT

Table 306. Required access controls for ISO-4 PIN blocks			
Input PIN format	**Output PIN format**	**Authenticated PAN-change allowed**	**Access control name**
ISO-0	ISO-4	No	Encrypted PIN Translate2 – Permit ISO-0 to ISO-4 Reformat.
ISO-1	ISO-4	No	Encrypted PIN Translate2 – Permit ISO-1 to ISO-4 Reformat (see note 1).
ISO-1	ISO-4	No	Encrypted PIN Translate2 – Permit ISO-1 to ISO-4 RFMT1TO4 (see note 1).
ISO-4	ISO-0	No	Encrypted PIN Translate2 – Permit ISO-4 to ISO-0 Reformat.
ISO-4	ISO-1	No	Encrypted PIN Translate2 – Permit ISO-4 to ISO-1 Reformat (see note 2).
ISO-4	ISO-1	No	Encrypted PIN Translate2 – Permit ISO-4 to ISO-1 RFMT4TO1 (see note 2).
ISO-4	ISO-4	No	Encrypted PIN Translate2 – Permit ISO-4 to ISO-4 Translate.
ISO-4	ISO-4	Yes	Encrypted PIN Translate2 – Permit ISO-4 Reformat with PAN Change (see note 3).
ISO-4	ISO-4	Yes	Encrypted PIN Translate2 - Permit ISO-4 to ISO-4 PTR2AUTH (see note 3).

Notes:

1. When enabled, the **Encrypted PIN Translate2 – Permit ISO-1 to ISO-4 RFMT1TO4** control has the effect of disallowing REFORMAT requests from ISO-1 to ISO-4 PIN blocks unless the outbound PIN encrypting key has the RFMT1TO4 key-usage field bit enabled in the AES key-token.
2. When enabled, the **Encrypted PIN Translate2 – Permit ISO-4 to ISO-1 RFMT4TO1** control has the effect of disallowing REFORMAT requests from ISO-4 to ISO-1 PIN blocks unless the inbound PIN encrypting key has the RFMT4TO1 key-usage field bit enabled in the AES key-token.

3. When enabled, the **Encrypted PIN Translate2 – Permit ISO-4 to ISO-4 PTR2AUTH** control has the effect of disallowing REFORMAT requests from ISO-4 to ISO-4 PIN blocks unless the outbound PIN encrypting key has the PTR2AUTH key-usage field bit enabled in the AES key-token.

If any of the Unique Key per Transaction rule array keywords are specified, the **DUKPT - PIN Verify, PIN Translate** access control point must be enabled.

An enhanced PIN security mode is available for extracting PINs from a 3621 or 3624 encrypted PIN-block and formatting an encrypted PIN block into IBM 3621 or 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** access control.

When the **Encrypted PIN Translate - Translate PIN Check** access control is enabled, checking of the PIN block is performed. The checking is the same as the checking done when the REFORMAT keyword is specified.

When the **General ISO PIN Error Mode** access control is enabled, the return code will be a general PIN block error (return code 8 reason code 2514) instead of some of the PIN block errors return code. The use of a general return code can prevent the abuse of PIN processing error messages due to information leakage derived from the return code reason codes returned under various conditions. For more details, see "PIN block error processing mode" on page 609.

Three additional access controls should be considered: **ANSI X9.8 PIN - Enforce PIN block restrictions, ANSI X9.8 PIN - Allow modification of PAN, and ANSI X9.8 PIN - Allow only ANSI PIN blocks**. These three access controls affect how PIN processing is performed as described below. The access controls will affect this and other PIN processing services if enabled.

1. Enable the **ANSI X9.8 PIN - Enforce PIN block restrictions** access control to apply additional restrictions to PIN processing as follows:
   - Do not translate or reformat a non-ISO PIN block into an ISO PIN block. Specifically, do not allow an IBM 3624 PIN-block format in the *output_PIN_profile* variable when the PIN-block format in the *input_PIN_profile* variable is not IBM 3624.
   - Constrain use of ISO-2 PIN blocks to offline PIN verification and PIN change operations in integrated circuit card environments only. Specifically, do not allow ISO-2 input or output PIN blocks.
   - Do not translate or reformat a PIN-block format that includes a PAN into a PIN-block format that does not include a PAN. Specifically, do not allow an ISO-1 PIN-block format in the *output_PIN_profile* variable when the PIN-block format in the *input_PIN_profile* variable is ISO-0, ISO-3, or ISO-4.
   - Do not allow a change of PAN data. Specifically, when performing translations between PIN block formats that both include PAN data, do not allow the *input_PAN_data* and *output_PAN_data* variables to be different from the PAN data enciphered in the input PIN-block.
2. Enable the **ANSI X9.8 PIN - Allow modification of PAN** access control to override the restriction to not allow a change of PAN data. This override is applicable only when either the **ANSI X9.8 PIN - Enforce PIN block restrictions** control, the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control, or both are enabled. This override is to support account number changes in issuing environments. The **ANSI X9.8 PIN - Allow modification of PAN** control has no effect if neither the **ANSI X9.8 PIN - Enforce PIN block restrictions** control nor the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control is enabled. This rule does not apply for CSNBPTRE, and PAN changes are not allowed.
3. Enable the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control to apply a more restrictive variation of the **ANSI X9.8 PIN - Enforce PIN block restrictions** control. In addition to the previously described restrictions of the **ANSI X9.8 PIN - Enforce PIN block restrictions** control, this control also restricts the *input_PIN_profile* and the *output_PIN_profile* to contain only ISO-0, ISO-1, ISO-3, and ISO-4 PIN block formats. Specifically, the IBM 3624 PIN-block format is not allowed with this command. The **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control overrides the **ANSI X9.8 PIN - Enforce PIN block restrictions** control.

When the **Disallow translation from AES wrapping to DES wrapping** access control point is enabled in the domain role, this service fails if the *input_PIN_encrypting_key_identifier* is an AES key and the *output_PIN_encrypting_key_identifier* is a DES key.

When the **Disallow translation from AES wrapping to weaker AES wrapping** access control point is enabled in the domain role, this service fails if the *input_PIN_encrypting_key_identifier* is stronger than the *output_PIN_encrypting_key_identifier*.

When the **Disallow translation from DES wrapping to weaker DES wrapping** access control point is enabled in the domain role, this service fails if the *input_PIN_encrypting_key_identifier* is stronger than the *output_PIN_encrypting_key_identifier*.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *input_PIN_profile* and *output_PIN_profile* parameters is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 307. Encrypted PIN Translate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	This service requires the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	This service requires the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. Support for key usage attribute RFMT4TO1 for AES PINPROT keys requires the November 2019 or later licensed internal code. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	This service requires the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens and PANAUTAS and PANAUTI4 keywords require a CEX6C with the July 2019 or later licensed internal code (LIC). Support for key usage attribute RFMT4TO1 for AES PINPROT keys requires the November 2019 or later licensed internal code. The AES-DUKPT algorithm requires the October 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 307. Encrypted PIN Translate2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Support for key usage attribute RFMT4TO1 for AES PINPROT keys requires the October 2019 or later licensed internal code. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Support for key usage attribute RFMT4TO1 for AES PINPROT keys requires the October 2019 or later licensed internal code. The AES-DUKPT algorithm requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Encrypted PIN Translate Enhanced (CSNBPTRE and CSNEPTRE)

Use the Encrypted PIN Translate Enhanced callable service to change the format of a PIN block where the PAN field is enciphered using format preserving encryption. The service supports translation of PIN blocks whose PAN information has been enciphered using the Visa Merchant Data Secure (VMDS) standard and Visa Format Preserving Encryption (VFPE) encryption methods. Change of PAN data is not allowed.

PIN blocks are sometimes formatted using the PAN information. For this service, either the input PIN block profile or the output PIN block profile must specify a PIN block format that incorporates a PAN. The PIN block formats which incorporate a PAN are ISO-0, ISO-3, and Visa Format 4. Change of PAN data is not allowed.

Unique-key-per-transaction key derivation support is available for the *input_PIN_encrypting_key_identifier*, *output_PIN_encrypting_key_identifier*, and the *PAN_key_identifier* parameters. Optional rule array keywords determines which keys are to be derived and which key identifier parameters contains the key-generating key.

VMDS enciphered PAN data can be enciphered using DUKPT key management or static TDES key management. The enciphered PAN could be enciphered with the CBC or VFPE mode. The VMDS standard requires that the same key management scheme and type of keys be used for both the PIN and the PAN.

For VMDS, the following pairings are supported:

Table 308. VMDS pairings for enciphered PAN data

Function	Source		Target	
	Key management	**VMDS option**	**Key management**	**VMDS option**
Translation	DUKPT	Standard CBC	Static TDES non-DUKPT	Standard CBC
		VFPE		
	Static TDES non-DUKPT	Standard CBC		

The VMDS standard refers to double length, non-DUKPT keys as Zone Encryption keys.

To use this service, specify the following information:

- The mode of operation with a keyword in the rule array: **REFORMAT**.
- Optionally, the method of PIN extraction with a rule-array keyword.
- The Input and Output PIN block encrypting keys, or the key-encrypting keys used to derive the PIN block enciphering keys (rule array keywords **DUKPT-IP**, **DUKPT-OP**, or **DUKPT-BH**).
- The PAN-encrypting key or base key used to derive the PAN-encrypting key (rule array keywords **IN-DUKPT**, **OUTDUKPT**, or **STATIC**).
- The Input PIN block.
- The Input and Output PIN profiles. For UKPT processing, the profiles are extended to 48 bytes with a 24-byte current-key serial number (CKSN) extension.
- The Input PAN data as required by the selected PIN-block formats.
- An output PIN-block sequence number. Specify a value of 99999.
- For VMDS processing, you must also specify:
  - Processing algorithm: **VMDS**.
  - PAN input character set: **PAN8BITA** or **PAN4BITX**.
  - PAN input data encryption algorithm: **TDES**.
  - PAN input data mode: **CBC** or **VFPE**.
  - If using VFPE mode encryption, check digit compliance indicator.
  - If using CBC mode encryption, the data decryption key is needed to recover the enciphered PAN.

The callable service name for AMODE(64) invocation is CSNEPTRE.

Format

```
CALL CSNBPTRE(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              input_PIN_key_identifier_length,
              input_PIN_key_identifier,
              output_PIN_key_identifier_length,
              output_PIN_key_identifier,
              PAN_key_identifier_length,
              PAN_key_identifier,
              input_PIN_profile_length,
              input_PIN_profile,
              PAN_data_length,
              PAN_data,
              input_PIN_block_length,
              input_PIN_block,
              output_PIN_profile_length,
              output_PIN_profile,
              sequence_number,
```

```
                output_PIN_block_length,
                output_PIN_block,
                reserved1_length,
                reserved1,
                reserved2_length,
                reserved2 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be between 6 and 10 inclusive.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 309. Rule array keywords for Encrypted PIN Translate Enhanced

Keyword	Meaning
Mode (required)	
REFORMAT	Specifies that either or both the PIN-block format and the PIN-block encryption are to be changed. If the PIN-extraction method is not chosen by default, another element in the rule array must specify one of the keywords that indicates a PIN-extraction method.
Processing method (required)	
VMDS	Specifies the Visa Merchant Data Secure method be used for processing.
Input PAN data key management method (one required) These keywords are used to define the PAN-encrypting key used to decrypt the *PAN_data* parameter.	
IN-DUKPT	Specifies that the key to be used to decrypt the PAN data is to be derived using the key specified in the *input_PIN_key_identifier*. See the description of the *input_PIN_key_identifier* for the requirements of the key. The DUKPT-BH, DUKPT-IP, ADUKPTBH, or ADUKPTIP keyword is required.
OUTDUKPT	Specifies that the key to be used to decrypt the PAN data is to be derived using the key specified in the *output_PIN_key_identifier*. See the description of the *output_PIN_key_identifier* for the requirements of the key. The DUKPT-BH, DUKPT-OP, ADUKPTBH, or ADUKPTOP keyword is required.
STATIC	Specifies that key is supplied in the *PAN_key_identifier* parameter is to be used to decrypt the PAN data.
Input data algorithm (one required)	
TDES	Specifies Triple-DES encryption was used for the PAN data.
Input data mode (one required)	
CBC	Specifies CBC mode encryption was used for the PAN data. This is the mode for the Standard Encryption option.
VFPE	Specifies Visa format preserving mode encryption was used for the PAN data.
PAN input character set (one required)	
PAN4BITX	Specifies the PAN data character set is 4-bit hexadecimal. Two digits per byte. Not valid with the CBC rule.
PAN8BITA	Specifies the PAN data character set is normal ASCII represented in binary format. Not valid with CBC rule.
PAN-EBLK	Specifies the PAN data is in a CBC encrypted block. Valid only with CBC rule.
PAN check digit compliance (one required if mode VFPE and PAN input character set keyword present; otherwise, not allowed)	
CMPCKDGT	Last digit of the PAN data contains a compliant check digit per ISO/IEC 7812-1.

Table 309. Rule array keywords for Encrypted PIN Translate Enhanced (continued)	
Keyword	**Meaning**
NONCKDGT	Last digit of the PAN data contains does not contain a compliant check digit per ISO/IEC 7812-1.
DES DUKPT (one optional). These keywords are for the PIN-encrypting keys. See Table 311 on page 711 for valid DUKPT keyword combinations.	
DUKPT-IP	Specifies that the input PIN-encrypting key is to be derived using the key-generating key specified in the *input_PIN_key_identifier* parameter. See the description of the *input_PIN_key_identifier* parameter for the requirements of the key. This keyword cannot be specified with ADUKPTBH or ADUKPTIP.
DUKPT-OP	Specifies that the output PIN-encrypting key is to be derived using the key-generating key specified in the *output_PIN_key_identifier* parameter. See the description of the *output_PIN_key_identifier* parameter for the requirements of the key. This keyword cannot be specified with ADUKPTBH or ADUKPTOP.
DUKPT-BH	Specifies that the input and output PIN-encrypting keys are to be derived using the key-generating key specified in the respective parameters. See the descriptions of the *input_PIN_key_identifier* and *output_PIN_key_identifier* parameters for the requirements of the keys. This keyword cannot be specified with any of the keywords in the AES DUKPT group.
AES DUKPT (one optional). See Table 311 on page 711 for valid DUKPT keyword combinations.	
ADUKPTBH	Specifies that the input and output PIN-encrypting keys are to be derived with the AES DUKPT algorithm using the key-generating key specified in the respective parameters. See the descriptions of the *input_PIN_key_identifier* and *output_PIN_key_identifier* parameters for the requirements of the keys. This keyword cannot be specified with any of the keywords in the DES DUKPT group.
ADUKPTIP	Specifies the use of DUKPT input-key derivation and PIN-block decryption. AES DUKPT method. Specifies that the input PIN-encrypting key is to be derived using the key-generating key specified in the *input_PIN_key_identifier* parameter. See the description of the *input_PIN_key_identifier* parameter for the requirements of the key. This keyword cannot be specified with DUKPT-BH or DUKPT-IP.
ADUKPTOP	Specifies that the output PIN-encrypting key is to be derived with the AES DUKPT algorithm using the key-generating key specified in the *output_PIN_key_identifier* parameter. See the description of the *output_PIN_key_identifier* parameter for the requirements of the key. This keyword cannot be specified with DUKPT-BH or DUKPT-OP.
PIN-extraction method (one, optional). See "PIN block format and PIN extraction method keywords" on page 610 for additional information and a list of PIN block formats and PIN extraction method keywords.	
Note: If a PIN extraction method is not specified, the first one method listed in Table 250 on page 610 for the PIN block format will be the default method.	

input_PIN_key_identifier_length

Direction	Type
Input	Integer

Specifies the length of the *input_PIN_key_identifier* parameter in bytes.

If the *input_PIN_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

input_PIN_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to decrypt the input PIN block or the base derivation key to be used to derive the key to decrypt the input PIN block. The base derivation key can optionally be used to derive the key to decrypt the PAN data. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA keys

If you do not use the DUKPT process or you specify the ADUKPTOP rule array keyword, the 64-byte key token must contain the PIN-encrypting key to be used to decipher the input PIN block. The key algorithm must be DES, the key type must be IPINENC, and the key usage REFORMAT bit must be enabled.

If you use the DES-DUKPT process for the input PIN block by specifying the DUKPT-IP or DUKPT-BH rule array keyword, the key token must contain the base derivation key to derive the PIN-encrypting key. If you have also specified the IN-DUKPT keyword, the key will be used to derive the key to decrypt the PAN data. For DES-DUKPT, a DES 64-byte KEYGENKY key with the UKPT key usage bit enabled must be supplied.

If you use the AES-DUKPT process for the input PIN block by specifying the ADUKPTIP or ADUKPTBH rule array keywords, the key token must contain the base derivation key to derive the PIN-encrypting key. If you have also specified the IN-DUKPT keyword, the key will be used to derive the key to decrypt the PAN data. For AES-DUKPT, an AES variable-length DKYGENKY key with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1 must be supplied.

For X9.143 (TR-31) key blocks

If you do not use the DUKPT process or specified the ADUKPTOP rule array keyword, the variable-length key block must contain the PIN-encrypting key to be used to decipher the input PIN block (key usage P0, algorithm T, and mode of use D).

If you use the DES-DUKPT process for the input PIN block by specifying the DUKPT-IP or DUKPT-BH rule array keyword, the key token must contain the base derivation key to derive the PIN-encrypting key. If you have also specified the IN-DUKPT keyword, the key will be used to derive the key to decrypt the PAN data (key usage B0, algorithm T, and mode of use X).

If you use the AES-DUKPT process for the input PIN block by specifying the ADUKPTIP or ADUKPTBH rule array keywords, the key token must contain the base derivation key to derive the PIN-encrypting key. If you have also specified the IN-DUKPT keyword, the key will be used to derive the key to decrypt the PAN data (key usage B0, algorithm A, and mode of use X).

If the token or key block supplied was encrypted under the old master key, the token or key block will be returned encrypted under the current master key.

output_PIN_key_identifier_length

Direction	Type
Input/Output	Integer

Specifies the length of the *output_PIN_key_identifier* parameter in bytes.

If the *output_PIN_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

output_PIN_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to encrypt the output PIN block or the base derivation key to be used to derive the key to encrypt the output PIN block. The base derivation key can optionally be used to derive the key to encrypt the PAN data. The key identifier is an operational key token or key block or the key label of an operational token orblock in key storage.

For CCA keys

If you do not use the DUKPT process or specified the DUKPT-IP or ADUKPTIP rule array keyword, the 64-byte key token must contain the PIN-encrypting key to be used to encipher the output PIN block. The key algorithm must be DES, the key type must be OPINENC, and the key usage REFORMAT bit must be enabled.

If you use the DES DUKPT process for the output PIN block by specifying the DUKPT-OP or DUKPT-BH rule array keyword, the 64-byte key token must contain the base derivation key to derive the PIN-encrypting key. If you have also specified the OUTDUKPT keyword, the key will be used to derive the key to encrypt the PAN data. For DES-DUKPT, a DES KEYGENKY key with the UKPT key usage bit enabled must be supplied.

If you use the AES DUKPT process for the output PIN block by specifying the ADUKPTOP or ADUKPTBH rule array keyword, the key token must contain the base derivation key to derive the PIN-encrypting key. If you have also specified the OUTDUKPT keyword, the key will be used to derive the key to encrypt the PAN data. For AES-DUKPT, an AES DKYGENKY key with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1 must be supplied.

For X9.143 (TR-31) key blocks

If you do not use the DUKPT process or specified the DUKPT-IP or ADUKPTIP rule array keyword, the key token must contain the PIN-encrypting key to be used to encipher the output PIN block (key usage P0, algorithm T, and mode of use E).

If you use the DES DUKPT process for the output PIN block by specifying the DUKPT-OP or DUKPT-BH rule array keyword, the key token must contain the base derivation key to derive the PIN-encrypting key. If you have also specified the OUTDUKPT keyword, the key will be used to derive the key to encrypt the PAN data (key usage B0, algorithm T, and mode of use X).

If you use the AES DUKPT process for the output PIN block by specifying the ADUKPTOP or ADUKPTBH rule array keyword, the key token must contain the base derivation key to derive the PIN-encrypting key. If you have also specified the OUTDUKPT keyword, the key will be used to derive the key to encrypt the PAN data (key usage B0, algorithm A, and mode of use X).

If the token or key block supplied was encrypted under the old master key, the token or key block will be returned encrypted under the current master key.

PAN_key_identifier_length

Direction	Type
Input	Integer

Specifies the length of the *PAN_key_identifier* parameter in bytes.

When the PAN key management method keyword is STATIC:

- And the identifier is a label, the value is 64.
- And the identifier is a key token or block, the value must be between the actual length of the token and 9992.

Otherwise, the value is 0.

PAN_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the PAN data. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

When *PAN_key_identifier_length* is zero, this parameter is ignored.

For CCA key tokens, the identifier is a 64-byte DES key token of a double-length data-encrypting key of key type CIPHER or DECIPHER.

For X9.143 keys, the identifier is a variable-length key block of a TDES data-encrypting key: key usage D0, algorithm T, and mode of use B or D.

If the token or key block supplied was encrypted under the old master key, the token or key block will be returned encrypted under the current master key.

input_PIN_profile_length

Direction	Type
Input	Integer

Specifies the length of the *input_PIN_profile* parameter in bytes. Set the length according to the following table:

Table 310. Supported Encrypted PIN Translate Enhanced PIN profile lengths

Pin profile	Length
PIN-block format only.	24
PIN-block format and CKSN extension used for DES-DUKPT.	48
PIN-block format and single block of derivation data extension used for AES-DUKPT.	44

input_PIN_profile

Direction	Type
Input	String

The 24, 44, or 48 byte input PIN profile. The profile consists of three 8-byte character strings with information defining the input PIN-block format.

When the keywords DUKPT-BH, DUKPT-IP, ADUKPTIP, ADUKPTBH, or IN-DUKPT are specified, additional bytes must be present containing the CKSN or derivation data extension. The DES DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block when the CKSN extension is included in the *input_PIN_profile*. The AES DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block when the derivation data extension is included in the *input_PIN_profile*. See Table 684 on page 1719 for the layout of the AES-DUKPT derivation data

extension. The algorithm indicator must be set to either X'0000' (2-key TDES) or X'0001' (3-key TDES). The key usage indicator must be set to X'1000' (PIN Encryption).

PAN_data_length

Direction	Type
Input	Integer

When the input data mode keyword is CBC, this parameter specifies the length in bytes of the *PAN_data* parameter. The value must be 16.

When the input data mode keyword is VFPE, this parameter specifies the number of PAN digits. The value will be between 15 and 19 inclusive.

PAN_data

Direction	Type
Input	String

The enciphered primary account number (PAN) to be used to reformat the PIN block format. For VFPE mode, if the PAN contains an odd number of 4-bit digits, the data is left justified in the PAN variable and the rightmost 4 bits are ignored.

This service uses this data to recover the PIN from the PIN block if you specify the REFORMAT keyword and the input PIN profile specifies the ISO-0, VISA-4, or ISO-3 keyword for the PIN block format. If the output PIN profile specifies the ISO-0, VISA-4, or ISO-3 keyword for the PIN block format, the 12 rightmost digits of the PAN, excluding the check digit, are used to format the output PIN block.

input_PIN_block_length

Direction	Type
Input	Integer

Specifies the length of the *input_PIN_block* parameter in bytes. The value must be 8.

input_PIN_block

Direction	Type
Input	String

The 8-byte enciphered PIN block that contains the PIN to be processed.

output_PIN_profile_length

Direction	Type
Input	Integer

Specifies the length of the *output_PIN_profile* parameter in bytes. See Table 310 on page 708 for the supported PIN profile lengths.

output_PIN_profile

Direction	Type
Input	String

The 24, 44, or 48 byte PIN profile for the output PIN block. The profile contains three 8-byte character strings with information defining the PIN-block format.

When the keywords DUKPT-BH, DUKPT-OP, ADUKPTOP, ADUKPTBH, or OUTDUKPT are specified, additional bytes must be present containing the CKSN or derivation data extension. The DES DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block when the CKSN extension is included in the *input_PIN_profile*. The AES-DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block when the derivation data extension is included in the *input_PIN_profile*. See Table 684 on page 1719 for the layout of the AES-DUKPT derivation data extension. The algorithm indicator must be set to either X'0000' (2-key TDES) or X'0001' (3-key TDES). The key usage indicator must be set to X'1000' (PIN Encryption).

sequence_number

Direction	Type
Input	Integer

The 4 byte sequence number if the output PIN block format is 3621. Specify the integer value 99999. Otherwise, this parameter is ignored.

output_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length of the *output_PIN_block* parameter in bytes. The value must be at least 8 bytes. On output, the value is updated with the actual number of bytes returned.

output_PIN_block

Direction	Type
Output	String

The 8 byte reformatted PIN block.

reserved1_length

Direction	Type
Input	Integer

Length of the *reserved1* parameter in bytes. The value must be 0.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length of the *reserved2* parameter in bytes. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Table 311. Valid encrypted PIN Translate Enhanced DUKPT keyword combinations

DUKPT keyword combination	Input PIN encrypting key	Output PIN encrypting key
DUKPT-IP ADUKPTOP	Triple DES DUKPT	AES DUKPT
ADUKPTIP DUKPT-OP	AES DUKPT	Triple DES DUKPT
DUKPT-BH	Triple DES DUKPT	Triple DES DUKPT
ADUKPTBH	AES DUKPT	AES DUKPT
DUKPT-IP	Triple DES DUKPT	Static
DUKPT-OP	Static	Triple DES DUKPT
ADUKPTIP	AES DUKPT	Static
ADUKPTOP	Static	AES DUKPT

Access control points

The **Encrypted PIN Translate Enhanced** access control in the domain role controls the function of this service. In addition, the **Encrypted PIN Translate – Reformat** access control must be enabled when the mode rule array keyword is REFORMAT.

If any of the rule array keywords that control UKPT key derivation (IN-DUKPT, OUTDUKPT, DUKPT-OP, DUKPT-IP, and DUKPT-BH) are specified, the **UKPT - PIN Verify**, **PIN Translate** access control must be enabled.

An enhanced PIN security mode is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the **Enhanced PIN Security** access control point in the domain role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

An enhanced PIN security mode is available to implement restrictions required by the ANSI X9.8 PIN standard. To enforce these restrictions, you must enable the following control points in the domain role. See “ANSI X9.8 PIN restrictions” on page 606 for a description of these controls.

- ANSI X9.8 PIN - Enforce PIN block restrictions
- ANSI X9.8 PIN - Allow only ANSI PIN blocks

Note: The **ANSI X9.8 PIN - Allow modification of PAN** access control is not applicable to this service as the PAN is not allowed to be modified.

When the **Disallow translation from DES wrapping to weaker DES wrapping** access control point is enabled, this service will fail if the input PIN key identifier is a PIN-encrypting key and is stronger than the output PIN key identifier when it is a PIN-encrypting key.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *input_PIN_profile* and *output_PIN_profile* parameters is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 312. Encrypted PIN Translate Enhanced required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Requires the March 2016 or later licensed internal code (LIC). Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Requires the March 2016 or later licensed internal code (LIC). Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The AES-DUKPT algorithm requires the October 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	The AES-DUKPT algorithm requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Encrypted PIN Verify (CSNBPVR and CSNEPVR)

Use the Encrypted PIN verify callable service to verify that one of these customer selected trial PINs is valid:

- IBM 3624 (IBM-PIN)
- IBM 3624 PIN offset (IBM-PINO)
- IBM German Bank Pool (GBP-PIN)
- VISA PIN validation value (VISA-PVV)
- VISA PIN validation value (VISAPVV4)
- Interbank PIN (INBK-PIN)

The derived unique-key-per-transaction (DUKPT) algorithm is available. Both DES-DUKPT (ANSI x9.24-1 2007) and AES-DUKPT (ANSI x9.24-3 2017) are supported. This support is available for the input available for the *input_PIN_encrypting_key_identifier* parameter.

The unique-key-per-transaction key derivation for single and double-length keys is available for the *input_PIN_encrypting_key_identifier* parameter.

An enhanced PIN security mode is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. To do this, you must enable the Enhanced PIN Security access control point in the domain role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.

The callable service name for AMODE(64) invocation is CSNEPVR.

Format

```
CALL CSNBPVR(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             input_PIN_encrypting_key_identifier,
             PIN_verifying_key_identifier,
             input_PIN_profile,
             PAN_data,
             encrypted_PIN_block,
             rule_array_count,
             rule_array,
             PIN_check_length,
             data_array )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

input_PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the input PIN block or the key-generating key to be used to derive the key to decrypt the input PIN block. The key identifier is a variable-length operational key token or key block or the 64-byte key label of an operational token or block in key storage.

If none of the 'Derived unique key per transaction' rule array keywords are used, the key token must contain the input PIN-block encrypting key to be used to decrypt the input PIN-block. The key may be a DES key (all PIN block formats except ISO-4) or an AES key (PIN block format ISO-4).

For CCA key tokens

- For DES keys, the control vector in the key token must specify the IPINENC key-type with EPINVER bit (CV bit 19) set to B'1'.
- For AES keys, the variable-length symmetric key token must have a token algorithm of AES and a key type of PINPROT. In addition, the key usage fields must indicate that the key can be used for decryption (DECRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN function usage CPINENC must be enabled.

For X9.143 (TR-31) key blocks (variable-length)

- For DES keys, (key usage P0, algorithm T, and mode of use D).
- For AES keys, (key usage P0, algorithm A, and mode of use D).

If one of the 'Derived unique key per transaction' (DUKPT) rule array keywords is used:

For CCA key tokens

- If you use the DES DUKPT process specified with the UKPTIPIN or DUKPT-IP keyword for the input PIN-block, specify the 64-byte base derivation key as a KEYGENKY key type with the UKPT bit (CV bit 18) set to B'1'.
- If you use the AES DUKPT process specified with the ADUKPTIP keyword for the input PIN-block, specify the variable-length base derivation key as an AES DKYGENKY key with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1.

For X9.143 (TR-31) key blocks (variable-length)

- For DES keys, (key usage B0, algorithm T, and mode of use X).

- For AES keys, (key usage B0, algorithm A, and mode of use X).

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

PIN_verifying_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify PIN. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA key tokens, the identifier is a 64-byte DES key token of key type PINVER with key usage attribute EPINVER enabled in the control vector.

For X9.143 (TR-31) key, the identifier is a variable-length key block of a TDES PIN-verifying key:

- For process rule keywords IBM-PIN and IBM-PINO: key usage V1, algorithm T, and mode of use C or V.
- For process rule keywords VISA-PVV and VISAPVV4: key usage V2, algorithm T, and mode of use C or V.
- For process rule keywords GBP-PIN and INBK-PIN: key usage V0, algorithm T, and mode of use C or V.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

input_PIN_profile

Direction	Type
Input/Output	Character string

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile depending on whether the PIN block is being enciphered or deciphered by the callable service.

If you specify UKPTIPIN or DUKPT-IP in the *rule_array* parameter, the *input_PIN_profile* must be 48 bytes and must contain the current key serial number. See "The PIN profile" on page 609 for additional information.

If you specify ADUKPTIP in the *rule_array* parameter, the *input_PIN_profile* must be 44 bytes and must contain the AES-DUKPT derivation data extension. See "The PIN profile" on page 609 for additional information.

The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the encrypted PIN verify callable service.

PAN_data

Direction	Type
Input	Character String

The primary account number (PAN) in character format. This parameter must always be provided. The service uses this parameter if the PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 12-byte value in application storage whose contents will be ignored.

When using the ISO-0, ISO-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 – 19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Table 313. PAN data structure

Offset	Length	Description
0	2	Length of the PAN data field, p.
2	p	10 to 19 bytes of PAN data.
2+p	0-9	Padding.

encrypted_PIN_block

Direction	Type
Input	String

The enciphered PIN block that contains the PIN to be verified. When the PIN block format is ISO-4, the PIN block will be 16 bytes long. For all other formats, the PIN block will be 8 bytes long.

rule_array_count

Direction	Type
Input	Integer

The number of process rules specified in the *rule_array* parameter. The value may be 1, 2 or 3.

rule_array

Direction	Type
Input	Character String

The process rule for the PIN verify algorithm.

Table 314. Keywords for Encrypted PIN Verify

Keyword	Meaning
Algorithm Value (required)	
GBP-PIN	The IBM German Bank Pool PIN. It verifies the PIN entered by the customer and compares that PIN with the institution generated PIN by using an institution key.
IBM-PIN	The IBM 3624 PIN, which is an institution-assigned PIN. It does not calculate the PIN offset.
IBM-PINO	The IBM 3624 PIN offset, which is a customer-selected PIN and calculates the PIN offset.
INBK-PIN	The Interbank PIN verify algorithm.
VISA-PVV	The VISA PIN verify value.

Table 314. Keywords for Encrypted PIN Verify (continued)	
Keyword	**Meaning**
VISAPVV4	The VISA PIN verify value. If the length is 4 digits, normal processing for VISA-PVV will occur. If the length is greater than 4 digits, the service will fail.
PIN Block Format and PIN Extraction Method (optional)	See "PIN block format and PIN extraction method keywords" on page 610 for additional information and a list of PIN block formats and PIN extraction method keywords. **Note:** If a PIN extraction method is not specified, the first one listed in Table 250 on page 610 for the PIN block format will be the default.
DUKPT Rule (one optional)	
UKPTIPIN	The *input_PIN_encrypting_key_identifier* is derived as a single length key.
DUKPT-IP	The *input_PIN_encrypting_key_identifier* is to be derived using the TDES-DUKPT algorithm.
ADUKPTIP	The *input_PIN_encrypting_key_identifier* is to be derived using the AES-DUKPT algorithm.

PIN_check_length

Direction	Type
Input	Integer

The PIN check length for the IBM-PIN or IBM-PINO process rules only. Otherwise, it is ignored. Specify the rightmost digits, 4 through 16, for the PIN to be verified.

data_array

Direction	Type
Input	String

Three 16-byte elements required by the corresponding *rule_array* parameter. The data array consists of three 16-byte fields whose specification depend on the process rule. If a process rule only requires one or two 16-byte fields, then the rest of the data array is ignored by the callable service. Table 315 on page 717 describes the array elements.

Table 315. Array Elements for the Encrypted PIN Verify Callable Service	
Array Element	**Description**
Decimalization_table	Decimalization table for IBM and GBP only. Sixteen decimal digits of 0 through 9. **Note:** If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the domain role, this table must match one of the active decimalization tables in the coprocessors.

Table 315. Array Elements for the Encrypted PIN Verify Callable Service (continued)	
Array Element	**Description**
PIN_offset	Offset data for IBM-PINO. One to twelve numeric characters, 0 through 9, left-justified and padded on the right with blanks. The PIN offset length is specified in the *PIN_check_length* parameter. For IBM-PIN and GBP-PIN, the field is ignored.
trans_sec_parm	For VISA, only the leftmost twelve digits of the 16-byte field are used. These consist of the rightmost eleven digits of the personal account number (PAN) and a one-digit key index. The remaining four characters are ignored. For Interbank only, all 16 bytes are used. These consist of the rightmost eleven digits of the PAN, a constant of X'6', a one-digit key index, and three numeric digits of PIN validation data.
RPVV	For VISA-PVV only, referenced PVV (4 bytes) that is left-justified. The rest of the field is ignored.
Validation_data	Validation data for IBM and GBP padded to 16 bytes. One to sixteen characters of hexadecimal account data left-justified and padded on the right with blanks.

Table 316 on page 718 lists the data array elements required by the process rule (*rule_array* parameter). The numbers refer to the process rule's position within the array.

Table 316. Array Elements Required by the Process Rule					
Process Rule	**IBM-PIN**	**IBM-PINO**	**GBP-PIN**	**VISA-PVV**	**INBK-PIN**
Decimalization_table	1	1	1		
Validation_data	2	2	2		
PIN_offset	3	3	3		
Trans_sec_parm				1	1
RPVV				2	

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

This table shows the access control points in the domain role that control the function of this service.

Table 317. Required access control points for Encrypted PIN Verify	
Process rule	**Access control point**
IBM-PIN **IBM-PINO**	Encrypted PIN Verify - 3624
GBP-PIN	Encrypted PIN Verify - GBP
VISA-PVV	Encrypted PIN Verify - VISA PVV
INBK-PIN	Encrypted PIN Verify - Interbank

If any of the Unique Key per Transaction rule array keywords are specified, the **DUKPT - PIN Verify, PIN Translate** access control point must be enabled.

If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the domain role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *input_PIN_profile* parameter is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 318. Encrypted PIN Verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The ADUKPTIP keyword is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The ADUKPTIP keyword is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The ADUKPTIP keyword requires the October 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The ADUKPTIP keyword is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	The ADUKPTIP keyword requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 318. Encrypted PIN Verify required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Related information

"PIN formats and algorithms" on page 1631 discusses the PIN algorithms in detail.

Encrypted PIN Verify2 (CSNBPVR2 and CSNEPVR2)

The Encrypted PIN Verify2 callable service validates a customer encrypted PIN-block against a reference encrypted PIN block.

You can specify these PIN-block formats:

- IBM 3624
- ISO-0 (same as ANS X9.8, VISA-1, and ECI-1)
- ISO-1 (same as ECI-4)
- ISO-2
- ISO-3
- ISO-4

The service supports truncated customer PINs, optionally verifying an indicated number of PIN digits (minimum of 4) that is less than the number of digits for the reference PIN. When truncated PINs are compared, the order is right to left for the number of digits specified in the *PIN_check_length* parameter.

The derived unique-key-per-transaction (DUKPT) algorithm is available. Both DES-DUKPT (ANSI x9.24-1 2007) and AES-DUKPT (ANSI x9.24-3 2017) are supported. This support is available for the input available for the *input_PIN_encrypting_key_identifier* parameter and the *reference_PIN_encrypting_key_identifier* parameter.

The callable service name for AMODE(64) invocation is CSNEPVR2.

Format

```
CALL CSNBPVR2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              reference_PIN_rule_array_count,
              reference_PIN_rule_array,
              PIN_check_length,
              input_PIN_encrypting_key_identifier_length,
              input_PIN_encrypting_key_identifier,
              reference_PIN_encrypting_key_identifier_length,
              reference_PIN_encrypting_key_identifier,
              input_PIN_profile_length,
              input_PIN_profile,
              input_PIN_block_length,
              input_PIN_block,
              reference_PIN_profile_length,
```

```
                reference_PIN_profile,
                reference_PIN_block_length,
                reference_PIN_block,
                input_PAN_data,
                reference_PAN_data,
                reserved1_length,
                reserved1,
                reserved2_length,
                reserved2,
                reserved3_length,
                reserved3,
                reserved4_length,
                reserved4 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1, 2, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 319. Keywords for Encrypted PIN Verify2

Keyword	Meaning
Processing rule (one required).	
REFPIN	Specifies that the input PIN is to be compared to the reference PIN.
TRUNCPIN	Specifies that the input PIN is to be compared to a truncated version of the reference PIN for the number of digits specified by the *PIN_check_length* parameter. **Note:** The digits of the PINs are checked from the rightmost digit to the left for the number of digits specified.
Input PIN unique key per transaction (one, optional).	
UKPT	Specifies the use of the single-DES method of DUKPT key derivation and PIN-block decryption for the input PIN encrypting key.
DUKPT	Specifies the use of the triple-DES method of DUKPT key derivation and PIN-block decryption for the input PIN encrypting key.
ADUKPT	Specifies the use of the AES DUKPT method of DUKPT key-derivation and PIN-block decryption for the input PIN encrypting key.
Input PIN-extraction method (one, optional). See "PIN block format and PIN extraction method keywords" on page 610 for additional information and a list of PIN block formats and PIN extraction method keywords. **Note:** If a PIN extraction method is not specified, the first one listed in Table 250 on page 610 for the PIN block format will be the default.	

reference_PIN_rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0, 1, or 2.

reference_PIN_rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 320. Keywords for Encrypted PIN Verify2	
Keyword	**Meaning**
Reference PIN unique key per transaction (one, optional).	
UKPT	Specifies the use of the single-DES method of DUKPT key derivation and PIN-block decryption for the reference PIN encrypting key.
DUKPT	Specifies the use of the triple-DES method of DUKPT key derivation and PIN-block decryption for the reference PIN encrypting key.
ADUKPT	Specifies the use of the AES DUKPT method of DUKPT key-derivation and PIN-block decryption for the reference PIN encrypting key.
Reference PIN-extraction method (one, optional). See "PIN block format and PIN extraction method keywords" on page 610 for additional information and a list of PIN block formats and PIN extraction method keywords. **Note:** If a PIN extraction method is not specified, the first one listed in Table 250 on page 610 for the PIN block format will be the default.	

PIN_check_length

Direction	Type
Input	Integer

The number of digits of the reference PIN to compare when the TRUNCPIN keyword is specified. The value may be 4-16 inclusive and must be less or equal to the length of the reference PIN. When the REFPIN keyword is specified, the value must be zero.

input_PIN_encrypting_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_PIN_encrypting_key_identifier* parameter.

If the *input_PIN_encrypting_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

input_PIN_encrypting_key_identifier

Direction	Type
Input	String

This is either the identifier of the key to unwrap the input PIN block or the identifier of the key-generating key used to derive the key to unwrap the input PIN block. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

If none of the 'Input PIN unique key per transaction' rule array keywords are used, the key token must contain the input PIN-block encrypting key to be used to decrypt the input PIN-block. The key may be a DES key (all PIN block formats except ISO-4) or an AES key (PIN block format ISO-4).

For CCA key tokens

- For DES keys, the control vector in the 64-byte key token must specify the IPINENC key-type with EPINVER bit (CV bit 19) set to B'1'.
- For AES keys, the variable-length symmetric key token must have a token algorithm of AES and a key type of PINPROT. In addition, the key usage fields must indicate that the key can be used for decryption (DECRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN function usage EPINVER must be enabled.

For X9.143 (TR-31) key blocks (variable-length)

- For DES keys, (key usage P0, algorithm T, and mode of use D).
- For AES keys, (key usage P0, algorithm A, and mode of use D).

If one of the 'Input PIN unique key per transaction' rule array keywords is used:

For CCA key tokens

- When you use the DES DUKPT process for the input PIN-block, specify that the DES 64-byte base derivation key as a KEYGENKY key type with the UKPT bit (CV bit 18) set to B'1'.
- When you use the AES DUKPT process for the input PIN-block, specify the base derivation key as an AES DKYGENKY key with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1.

For X9.143 (TR-31) key blocks (variable-length)

- For DES keys, (key usage B0, algorithm T, and mode of use X).
- For AES keys, (key usage B0, algorithm A, and mode of use X).

When the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

reference_PIN_encrypting_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *reference_PIN_encrypting_key_identifier* parameter.

If the *reference_PIN_encrypting_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

reference_PIN_encrypting_key_identifier

Direction	Type
Input	String

This is either the identifier of the key to unwrap the reference PIN block or the identifier of the key-generating key used to derive the key to unwrap the reference PIN block. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

If none of the 'Reference PIN unique key per transaction' rule array keywords are used, the key token must contain the input PIN-block encrypting key to be used to decrypt the input PIN-block. The key may be a DES key (all PIN block formats except ISO-4) or an AES key (PIN block format ISO-4).

For CCA key tokens

- For DES keys, the control vector in the key token must specify the IPINENC key-type with EPINVER bit (CV bit 19) set to B'1'.

- For AES keys, the variable-length symmetric key token must have a token algorithm of AES and a key type of PINPROT. In addition, the key usage fields must indicate that the key can be used for decryption (DECRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN function usage EPINVER must be enabled.

For X9.143 (TR-31) key blocks (variable-length)

- For DES keys, (key usage P0, algorithm T, and mode of use D).
- For AES keys, (key usage P0, algorithm A, and mode of use D).

When any of the DUKPT rule array keywords are used for the reference PIN encrypting key:

For CCA key tokens

- When you use the DES DUKPT process for the input PIN-block, specify that the DES 64-byte base derivation key as a KEYGENKY key type with the UKPT bit (CV bit 18) set to B'1'.
- When you use the AES DUKPT process for the input PIN-block, specify the base derivation key as an AES DKYGENKY key with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1.

For X9.143 (TR-31) key blocks (variable-length)

- For DES keys, (key usage B0, algorithm T, and mode of use X).
- For AES keys, (key usage B0, algorithm A, and mode of use X).

When the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

input_PIN_profile_length

Direction	Type
Input	Integer

The length of the *input_PIN_profile* parameter in bytes.

Table 321. Supported Encrypted PIN Verify2 input PIN profile lengths

Pin profile	Length
PIN-block format only.	24
PIN-block format and CKSN extension used for DES-DUKPT.	48
PIN-block format and single block of derivation data extension used for AES-DUKPT.	44

input_PIN_profile

Direction	Type
Input	String

The three 8-byte character elements that contain information necessary to extract the PIN from a formatted PIN block. See "The PIN profile" on page 609 for additional information.

When the DUKPT keywords are specified for the input PIN encrypting key, additional bytes must be present containing the CKSN or derivation data extension. The DES DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block when the CKSN extension is included in the *input_PIN_profile*. The AES DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block when the derivation data extension is included in the *input_PIN_profile*. See Table

684 on page 1719 for the layout of the AES-DUKPT derivation data extension. The algorithm indicator must be set to either X'0000' (2-key TDES) or X'0001' (3-key TDES). The key usage indicator must be set to X'1000' (PIN Encryption).

input_PIN_block_length

Direction	Type
Input	Integer

The length of the *input_PIN_block* in bytes. The value must be 16 for ISO-4 PIN blocks and 8 for all other PIN blocks.

input_PIN_block

Direction	Type
Input	String

The encrypted PIN block containing the PIN to compare against the reference PIN.

reference_PIN_profile_length

Direction	Type
Input	Integer

The length of the *reference_PIN_profile* parameter in bytes.

Table 322. Supported Encrypted PIN Verify2 reference PIN profile lengths

Pin profile	Length
PIN-block format only.	24
PIN-block format and CKSN extension used for DES-DUKPT.	48
PIN-block format and single block of derivation data extension used for AES-DUKPT.	44

reference_PIN_profile

Direction	Type
Input	String

The three 8-byte character elements that contain information necessary to extract the reference PIN from a formatted PIN block. See “The PIN profile” on page 609 for additional information.

When the DUKPT keywords are specified for the reference PIN encrypting key, additional bytes must be present containing the CKSN or derivation data extension. The DES DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block when the CKSN extension is included in the *reference_PIN_profile*. The AES DUKPT algorithm will be used to derive the DUKPT key used to decrypt the input PIN block when the derivation data extension is included in the *reference_PIN_profile*. See Table 684 on page 1719 for the layout of the AES-DUKPT derivation data extension. The algorithm indicator must be set to either X'0000' (2-key TDES) or X'0001' (3-key TDES). The key usage indicator must be set to X'1000' (PIN Encryption).

reference_PIN_block_length

Direction	Type
Input	Integer

The length of the *reference_PIN_block* in bytes. The value must be 16 for ISO-4 PIN blocks and 8 for all other PIN blocks.

reference_PIN_block

Direction	Type
Input	String

The encrypted PIN block containing the reference PIN.

input_PAN_data

Direction	Type
Input	String

The PAN data for the *input_PIN_block*. The PAN data is used if the input PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the PIN block format. The *PAN_data* parameter is 21 bytes long.

When using the ISO-0, IS0-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 –19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Offset	Length	Description
0	2	Length of the PAN data field, *p*.
2	p	10 – 19 bytes of PAN data.
2+p	0-9	Padding.

Note: If the *reference_PAN_data* and the *input_PAN_data* are different lengths, the rightmost 12 digits must be the same excluding the check digit.

reference_PAN_data

Direction	Type
Input	String

The PAN data for the *reference_PIN_block*. The PAN data is used if the reference PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the PIN block format. The *PAN_data* parameter is 21 bytes long.

When using the ISO-0, IS0-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 –19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Offset	Length	Description
0	2	Length of the PAN data field, *p*.
2	p	10 – 19 bytes of PAN data.
2+p	0-9	Padding.

Note: If the *reference_PAN_data* and the *input_PAN_data* are different lengths, the rightmost 12 digits must be the same excluding the check digit.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input	String

This parameter is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This parameter is ignored.

reserved3_length

Direction	Type
Input	Integer

Length in bytes of the *reserved3* parameter. The value must be 0.

reserved3

Direction	Type
Input	String

This parameter is ignored.

reserved4_length

Direction	Type
Input	Integer

Length in bytes of the *reserved4* parameter. The value must be 0.

reserved4

Direction	Type
Input	String

This parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

This table shows the access control points in the domain role that control the function of this service.

Table 323. Required access control points for Encrypted PIN Verify2

Processing rule	Access control point
REFPIN	Encrypted PIN Verify2 - REFPIN
TRUNCPIN	Encrypted PIN Verify2 - TRUNCPIN

If any of the Unique Key per Transaction rule array keywords are specified, the **DUKPT - PIN Verify, PIN Translate** access control point must be enabled.

An enhanced PIN security mode is available for extracting PINs from a 3621 or 3624 encrypted PIN-block and formatting an encrypted PIN block into IBM 3621 or 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** access control.

If the **ANSI X9.8 PIN - Use stored decimalization tables only** access control point is enabled in the domain role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *input_PIN_profile* parameter is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 324. Encrypted PIN Verify2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**		This service is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**		This service is not supported. X9.143 key blocks are not supported.

Table 324. Encrypted PIN Verify2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	This service requires the CCA release 7.4 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Field Level Decipher (CSNBFLD and CSNEFLD)

Use the Field Level Decipher callable service to decrypt payment-related database fields that have been previously encrypted using the field level encipher callable service. A database in this context is any structured data area or repository such as Db2, IMS, VSAM, or any column delineated data set or file.

The callable service name for AMODE(64) invocation is CSNEFLD.

Format

```
CALL CSNBFLD(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             key_derive_data_length,
             key_derive_data,
             context_data_length,
             context_data,
             charset_parms_length,
             charset_parms,
             reserved_length,
             reserved,
             source_text_id,
             source_text_length,
             source_text,
             target_text_id,
             target_text_length,
             target_text)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 4. The maximum value is 5.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 325. Rule array keywords for Field Level Decipher

Keyword	Meaning
Algorithm (required)	
AESVFPE	AES
TDESVFPE	TDES
Key type (optional)	
KEY-CLR	Specifies that the *key_identifier* parameter contains a clear key value. KEY-CLR is the default value.

Table 325. Rule array keywords for Field Level Decipher (continued)	
Keyword	**Meaning**
KEYIDENT	Indicates that the value in the *key_identifier* parameter is either an internal key token or the label of a key token in the CKDS.
Keying method (one required)	
KEY	Indicates that the value in the *key_identifier* parameter is an encryption key and is to be only used for encryption one time with a given *context_data* value.
KEY-DRV	Indicates that the key specified by *key_identifier* is a base derivation key and may be used for encryption multiple times. For each encryption, a unique encryption subkey is derived from the base key and the *key_derive_data* parameters. The *key_derive_data* and *context_data* combination must be unique for each encryption. Furthermore, this same *key_derive_data* and *context_data* combination must be specified when decrypting the field with the CSNBFLD callable service. The *key_derive_data* length must be greater than or equal to 8 and cannot exceed 2000.
Context Method (one required)	
TWEAK	Indicates that the *context_data* parameter specifies the initialization vector (IV) to be used for encryption.
Charset (one required)	
ADIGITS	Indicates that the input characters are ASCII digits. ICSF converts the input characters to integer (ordinal) values using n equals 10 in the order below: `0123456789` The constant k is 18 for TDES and 37 for AES.
APRINT	Indicates that the input characters are printable ASCII. ICSF converts the input characters to integer (ordinal) values using n equals 95 in the order below: `!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQR STUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~` The constant k is 8 for TDES and 18 for AES.
EDIGITS	Indicates that the input characters are EBCDIC digits. ICSF converts the input characters to integer (ordinal) values using n equals 10 in the order below: `0123456789` The constant k is 18 for TDES and 37 for AES.
EPRINT	Indicates that the input characters are printable EBCDIC. ICSF converts the input characters to integer (ordinal) values using n equals 95 in the order below: `!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQR STUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~` The constant k is 8 for TDES and 18 for AES.

Table 325. Rule array keywords for Field Level Decipher (continued)	
Keyword	**Meaning**
ORDINAL	Indicates that the input characters have already been converted to ordinal value by the calling application. ICSF processes the characters using the n and k values specified in *charset_parms*. Set k equal to 0 to have the service calculate and use the optimal value.

key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *key_identifier* parameter.

For the KEY-CLR keyword, this value is the length in bytes of the clear key value only.

For the KEYIDENT keyword, when the *key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token or block and 9992.

Note: Single length DES keys are not supported.

key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to decrypt the data.

For the KEY-CLR keyword, *key_identifier* specifies the cipher key. The parameter must be left-justified.

For the KEYIDENT keyword, the key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

The data-encrypting key may be an:

- Internal clear or encrypted DES or AES DATA key in a 64-byte CCA key token.
- Encrypted DES CIPHER key in a 64-byte CCA key token or variable-length X9.143 (TR-31) key block.
- Encrypted AES CIPHER key in a variable-length CCA key token or X9.143 (TR-31) key block.

Notes:

- To use a fixed-length DES or AES encrypted CCA DATA key in the CKDS, the ICSF segment of the CSFKEYS class general resource profile that is associated with the specified key label must contain SYMCPACFWRAP(YES). To use a fixed-length DES or AES encrypted key that does not reside in the CKDS, the ICSF segment of the CSFKEYS class general resource profile CSF-PROTECTED-KEY-TOKEN (or its generic equivalent) must contain SYMCPACFWRAP(YES). For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.
- To use an encrypted fixed-length DES CIPHER CCA key token, the key token must allow decryption and export to the CPACF protected key format. For more information on creating key tokens with the specified attributes, see "Key Token Build (CSNBKTB and CSNEKTB)" on page 271.
- To use an encrypted variable-length AES CIPHER CCA key token, the key token must allow decryption, any cipher mode, and export to CPACF protected key format. For more information on creating key tokens with the specified attributes, see "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297. See the CKDS KEYS utility Key Attributes panel for information on the current attributes of a CKDS key.
- To use an encrypted variable-length DES or AES data-encryption X9.143 (TR-31) key block (key usage D0, algorithm A or T, mode of use B or D), the key block must allow decryption and export

to the CPACF protected key format (IBM optional block "10" is required). For more information on creating key blocks with the specified attributes, see “TR-31 Create (CSNBT31C and CSNET31C)” on page 938.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

key_derive_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *key_derive_data* parameter. For keying method KEY-DRV, the value must be greater than or equal to 8 and cannot exceed 2000. For keying method KEY, the value must be 0.

key_derive_data

Direction	Type
Input/Output	String

The data that is used to derive the encryption and decryption subkey when keying method KEY-DRV is selected.

context_data_length

Direction	Type
Input/Output	Integer

Specifies the length of the *context_data* parameter in bytes. For context method TWEAK, this is an input field and must be 16 for rule AESVFPE or 8 for rule TDESVFPE.

context_data

Direction	Type
Input/Output	String

For context method TWEAK:

Specifies the initialization vector (IV) to be used for encryption and decryption. This value is 8 bytes in length for TDES and 16 bytes in length for AES. The left most 7 bits must be zero. The right most *c* bits are reserved for the VFPE counter value (*c* is application specific). The section in between the left most 7 bits and the right most *c* bits is the TWEAK area. It is recommended that this TWEAK area is set to an application specific unique value. See information on the need for unique key and context data combinations.

The service increments the counter for each encryption block required (for example, incremented once for each k characters processed). The application is expected to initialize the counter value to zero. Multi-part encryption and decryption may be accomplished by setting the counter portion appropriately for subsequent calls.

For decryption, the *context_data* parameter must contain the same value that was used to encrypt the data.

Note: The caller must ensure that *c* is sufficiently large enough to account for the total number of characters being encrypted. The incrementing of the counter value should not be allowed to overflow into the TWEAK area. The service does not try to enforce this. The entire *context_data* value, minus the left most 7 bits, is treated as the VFPE counter T and incremented accordingly.

charset_parms_length

Direction	Type
Input	Integer

Contains the length of the *charset_parms* parameter in bytes. The value must be 4 for the keyword ORDINAL. Otherwise, it must be 0.

charset_parms

Direction	Type
Input/Output	String

For the charset rule ORDINAL, this parameter specifies the n and k values as two concatenated half words with n appearing first. n must be greater than or equal to 9 and less than or equal to 255. k, if non-zero, must be set appropriately, which depends on n and the algorithm (AES or TDES) being used. Setting a value too high will either cause the request to fail or affect performance. Likewise, setting a value too low could also affect performance. Set k equal to 0 to have the service calculate and use the optimal value. When k equals 0 on input, the calculated optimal value of k is returned on output.

source_text_id

Direction	Type
Input	Integer

The ALET of the *source_text* parameter to be deciphered.

reserved_length

Direction	Type
Input	Integer

Length in bytes of the *reserved* parameter. The value must be 0.

reserved

Direction	Type
Input	String

This field is ignored.

source_text_length

Direction	Type
Input	Integer

The length of the *source_text* parameter in bytes. The maximum value is 2,147,836,647. A zero value is valid.

source_text

Direction	Type
Input	String

The input text to be deciphered. The string must contain only the characters of the character set defined by the *charset* keyword. The service does not enforce this.

target_text_id

Direction	Type
Input	Integer

The ALET of the *target_text* parameter.

target_text_length

Direction	Type
Input/Output	Integer

On input, this parameter specifies the size of the storage pointed to by the *target_text* parameter. On output, this parameter has the actual length of the text stored in the buffer addressed by the *target_text* parameter. The value must not be less that the value in *source_text_length*.

target_text

Direction	Type
Output	String

The deciphered text returned by the service.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

To use a fixed-length CKDS encrypted key, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES).

To use a fixed-length encrypted key that does not reside in the CKDS, the ICSF segment of the CSFKEYS class general resource profile CSF-PROTECTED-KEY-TOKEN (or its generic equivalent) must contain SYMCPACFWRAP(YES) and the caller must have SAF access to the profile.

To use an encrypted variable-length AES CIPHER key token, the key token must allow decryption, any cipher mode, and export to CPACF protected key format.

No pre-processing or post-processing exits are enabled for this service.

The master keys need to be loaded only when using this service with an encrypted key.

The AESVFPE algorithm uses hardware if it is available. Otherwise, clear key operations are performed in software.

This service fails if execution would cause destructive overlay of the *source_text* parameter.

For multi-part encryption, the source text length should be a multiple of k bytes on each call until the last call, which can be less.

For rule ORDINAL, specifying k equals 0 indicates that the service should calculate, use, and return the optimal k value. The algorithm used to calculate the optimal k value is as follows:

- Set k equal to the highest power of n that is less than or equal to 2 to the b power (where b equals 64 for TDES and 128 for AES).
- If n to the k power does not evenly divide 2 to the b power, then set k equal to k-1.

Note: For certain values of n, subtracting 1 from k may result in a value that is, in fact, not optimal when encrypting large amounts of plain text. The service makes no attempt to distinguish these cases.

The encryption and decryption subkey is derived from the base key according to NIST SP 800-108, 'Recommendation for Key Derivation Pseudorandom Functions'. The KDF in Counter Mode variant is used as follows:

```
K1, K2, … Kn = PRF(Key,i || key_data_data || blen)
```

where

- The PRF is CMAC (See NIST SP 800-38B, 'Recommendation for Black Cipher Modes of Operation: The CMAC Mode for Authentication').
- *i* is one byte iteration variable.
- *blen* is the one byte bit length of the desired key (minus 1).

Note: The encryption and decryption subkey is always the same type and size as the base key.

Access control points

When the label of an encrypted key is specified for the *key_identifier* parameter, the appropriate access control points must be enabled.

Table 326. Access control points for Field Level Decipher

Key algorithm	Access control point
Crypto Express5 and earlier CCA coprocessors	
AESVFPE	High-performance secure AES keys
TDESVFPE	High-performance secure DES keys
Crypto Express6 and later CCA coprocessors	
All	Authenticated Key Export - EXPTSK

For AES CIPHER keys (CEX6C and later), the following access controls must be enabled:

- **Authenticated Key Export - DRVTXKEY**
- **Authenticated Key Export - EXPTSK**
- **Authenticated Key Export - SETSNKEY**

These access controls should be enabled by default, but **Authenticated Key Export - EXPTSK** may not be enabled and will cause return codes that are misleading.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 327. Field Level Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions Crypto Express 5 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. DES and AES CIPHER key tokens and triple-length DES DATA keys with a non-zero control vector are not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.

Table 327. Field Level Decipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. Encrypted variable-length AES CIPHER key tokens require a CEX6C. Triple-length DES DATA keys with a non-zero control vector require a CEX6C with the November 2018 or later licensed internal code (LIC). DES CIPHER key tokens require a CEX6C with the P41458.002 or later MCL. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Encrypted keys require a Crypto express coprocessor. DES and AES CIPHER key tokens and triple-length DES DATA keys with a non-zero control vector are not supported. X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Related information

You **cannot** overlap the source and target text fields. For example:

```
ssssss
     tttttt  is not supported.

tttttt
     ssssss  is not supported.

ssssssttttttt is supported.

ssssss
```

```
tttttt  is supported.

s represents the source text and t represents the target text.
```

Field Level Encipher (CSNBFLE and CSNEFLE)

Use the Field Level Encipher callable service to encrypt payment related database fields, preserving the format of the fields. A database in this context is any structured data area or repository such as Db2, IMS, VSAM, or any column delineated data set or file. For example, you can encrypt a 16-digit EBCDIC credit card number where the resulting cipher text would also be 16 EBCDIC digits.

This callable service implements the VISA Format Preserving Encryption algorithm, which is a counter mode stream cipher. Consequently, the service has very stringent keying requirements. These requirements must be followed exactly or the security of the algorithm may be severely compromised. There are two keying options:

The key identified by the *key_identifier* parameter is the encryption key.
The combination of the key and the *context_data* parameter (for example, the initialization vector (IV) or TWEAK) must be unique for each encryption call. This is similar to ICSF's other encryption services, such as CSNBSYE.

The key identified by the *key_identifier* parameter is a base derivation key.
The base key and the *key_derive_data* are used to derive the actual encryption subkey. The combination of the base key, key derive data, and the context data must be unique for each encryption call. For example, if you have the following database that you wish to protect by a static key:

Name	Email address	Credit card number	Last changed date and time	Other data
John Doe	jdoe@company.com	1111222233334444	2013/10/22 14.22.05	xxx
Lisa Smith	lsmith@company.com	6666777788889999	2012/07/19 09.10.16	yyy

- The credit card number column is to be encrypted.
- The email address column is the record's primary index to the database.
- The last changed date and time column is updated every time that the row is changed.

When the credit card number field is updated and needs to be encrypted, the concatenation of the email address field and last changed date and time field are supplied as the key derive data. Because the last changed date and time field changes with each update, the combination of this field with the primary index and the static key produces a unique encryption subkey. Using this technique, the context data is not needed to provide uniqueness. Therefore, it may be a constant value that does not need to be stored in the record.

Note: In order to decrypt the data at a later time, the same base key, key derive data, and the context data must be supplied to the CSNBFLD callable service. Therefore, care must be taken with respect to the last changed date and time field as that would change for other field updates. Whenever this happens, the credit card number field would have to be decrypted using the old last changed date and time field value and re-encrypted under the new value.

The callable service name for AMODE(64) invocation is CSNEFLE.

Format

```
CALL CSNBFLE(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             key_derive_data_length,
```

```
                key_derive_data,
                context_data_length,
                context_data,
                charset_parms_length,
                charset_parms,
                reserved_length,
                reserved,
                source_text_id,
                source_text_length,
                source_text,
                target_text_id,
                target_text_length,
                target_text)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 4. The maximum value is 5.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 328. Rule array keywords for Field Level Encipher

Keyword	Meaning
Algorithm (required)	
AESVFPE	AES
TDESVFPE	TDES
Key type (optional)	
KEY-CLR	Specifies that the *key_identifier* parameter contains a clear key value. KEY-CLR is the default value.
KEYIDENT	Indicates that the value in the *key_identifier* parameter is either an internal key token or the label of a key token in the CKDS.
Keying method (one required)	
KEY	Indicates that the value in the *key_identifier* parameter is an encryption key and is to be only used for encryption one time with a given *context_data* value.
KEY-DRV	Indicates that the key specified by *key_identifier* is a base derivation key and may be used for encryption multiple times. For each encryption, a unique encryption subkey is derived from the base key and the *key_derive_data* parameters. The *key_derive_data* and *context_data* combination must be unique for each encryption. Furthermore, this same *key_derive_data* and *context_data* combination must be specified when decrypting the field with the CSNBFLD callable service. The *key_derive_data* length must be greater than or equal to 8 and cannot exceed 2000.
Context Method (one required)	
TWEAK	Indicates that the *context_data* parameter specifies the initialization vector (IV) to be used for encryption.
TWEAKGEN	For AESVFPE algorithm only. Indicates that a random initialization vector (IV) is generated, used for encryption, and returned in the *context_data* parameter.
Charset (one required)	
ADIGITS	Indicates that the input characters are ASCII digits. ICSF converts the input characters to integer (ordinal) values using n equals 10 in the order below: `0123456789` The constant k is 18 for TDES and 37 for AES.

Table 328. Rule array keywords for Field Level Encipher (continued)	
Keyword	**Meaning**
APRINT	Indicates that the input characters are printable ASCII. ICSF converts the input characters to integer (ordinal) values using n equals 95 in the order below: `!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQR STUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~` The constant k is 8 for TDES and 18 for AES.
EDIGITS	Indicates that the input characters are EBCDIC digits. ICSF converts the input characters to integer (ordinal) values using n equals 10 in the order below: `0123456789` The constant k is 18 for TDES and 37 for AES.
EPRINT	Indicates that the input characters are printable EBCDIC. ICSF converts the input characters to integer (ordinal) values using n equals 95 in the order below: `!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQR STUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~` The constant k is 8 for TDES and 18 for AES.
ORDINAL	Indicates that the input characters have already been converted to ordinal value by the calling application. ICSF processes the characters using the n and k values specified in *charset_parms*. Set k equal to 0 to have the service calculate and use the optimal value.

key_identifier_length

Direction	**Type**
Input	Integer

Specifies the length in bytes of the *key_identifier* parameter.

For the KEY-CLR keyword, this value is the length in bytes of the clear key value only.

For the KEYIDENT keyword, when the *key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token or block and 9992.

Note: Single length DES keys are not supported.

key_identifier

Direction	**Type**
Input/Output	String

The identifier of the data-encrypting key to encrypt the data.

For the KEY-CLR keyword, *key_identifier* specifies the cipher key. The parameter must be left justified.

For the KEYIDENT keyword, the key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

The data-encrypting key may be an:

- Internal clear or encrypted DES or AES DATA key in a 64-byte CCA key token.
- Encrypted DES CIPHER key in a 64-byte CCA key token or variable-length X9.143 (TR-31) key block.
- Encrypted AES CIPHER key in a variable-length CCA key token or X9.143 (TR-31) key block.

Notes:

- To use a fixed-length DES or AES encrypted CCA DATA key in the CKDS, the ICSF segment of the CSFKEYS class general resource profile that is associated with the specified key label must contain SYMCPACFWRAP(YES). To use a fixed-length DES or AES encrypted key that does not reside in the CKDS, the ICSF segment of the CSFKEYS class general resource profile CSF-PROTECTED-KEY-TOKEN (or its generic equivalent) must contain SYMCPACFWRAP(YES). For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.
- To use an encrypted fixed-length DES CIPHER CCA key token, the key token must allow decryption and export to the CPACF protected key format. For more information on creating key tokens with the specified attributes, see "Key Token Build (CSNBKTB and CSNEKTB)" on page 271.
- To use an encrypted variable-length AES CIPHER CCA key token, the key token must allow encryption, any cipher mode, and export to CPACF protected key format. For more information on creating key tokens with the specified attributes, see "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297. See the CKDS KEYS utility Key Attributes panel for information on the current attributes of a CKDS key.
- To use an encrypted variable-length DES or AES data-encryption X9.143 (TR-31) key block (key usage D0, algorithm A or T, mode of use B or E), the key block must allow encryption and export to the CPACF protected key format (IBM optional block "10" is required). For more information on creating key blocks with the specified attributes, see "TR-31 Create (CSNBT31C and CSNET31C)" on page 938.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

key_derive_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *key_derive_data* parameter. For keying method KEY-DRV, the value must be greater than or equal to 8 and cannot exceed 2000. For keying method KEY, the value must be 0.

key_derive_data

Direction	Type
Input/Output	String

The data used to derive the encryption and decryption subkey when keying method KEY-DRV is selected.

context_data_length

Direction	Type
Input/Output	Integer

Specifies the length of the *context_data* parameter in bytes. For context method TWEAK, this is an input field and must be 16 for rule AESVFPE or 8 for rule TDESVFPE. For context method TWEAKGEN, the value must be greater than or equal to 16 on input and is set to 16 on output.

context_data

Direction	Type
Input/Output	String

For context method TWEAK:

Specifies the initialization vector (IV) to be used for encryption and decryption. This value is 8 bytes in length for TDES and 16 bytes in length for AES. The left most 7 bits must be zero. The right most *c* bits are reserved for the VFPE counter value (*c* is application specific). The section in between the left most 7 bits and the right most *c* bits is the TWEAK area. It is recommended that this TWEAK area is set to an application specific unique value. See the preceding information on the need for unique key and context data combinations.

The service increments the counter for each encryption block required (for example, incremented once for each k characters processed). The application is expected to initialize the counter value to zero. Multi-part encryption and decryption may be accomplished by setting the counter portion appropriately for subsequent calls.

For decryption, the *context_data* parameter must contain the same value that was used to encrypt the data.

Note: The caller must ensure that *c* is sufficiently large enough to account for the total number of characters being encrypted. The incrementing of the counter value should not be allowed to overflow into the TWEAK area. The service does not try to enforce this. The entire *context_data* value, minus the left most 7 bits, is treated as the VFPE counter T and incremented accordingly.

For context method TWEAKGEN:

For AES encryption only. A random initialization vector (IV) is generated and returned in the *context_data* parameter. The returned *context_data* value must be specified as the input *context_data* when decrypting the field with the CSNBFLD callable service.

The returned *context_data* value is 16 bytes in length and has the following format:

Byte 1
Zero

Bytes 2 through 13
Pseudo-random value

Bytes 14 through 16
Zeros (area reserved for the VFPE counter value, *c*)

The caller must preallocate the 16-byte *context_data* field to hold the return value and set *context_data_length* greater than or equal to 16.

Multi-part encryption may be accomplished by starting the encryption with the TWEAKGEN rule and then switching to the TWEAK rule for subsequent calls. Remember to set the counter portion appropriately for each call.

charset_parms_length

Direction	Type
Input	Integer

Contains the length of the *charset_parms* parameter in bytes. The value must be 4 for the keyword ORDINAL. Otherwise, it must be 0.

charset_parms

Direction	Type
Input/Output	String

For the charset rule ORDINAL, this parameter specifies the n and k values as two concatenated half words with n appearing first. n must be greater than or equal to 9 and less than or equal to 255. k, if non-zero, must be set appropriately, which depends on n and the algorithm (AES or TDES) being used. Setting a value too high will either cause the request to fail or affect performance. Likewise, setting a value too low could also affect performance. Set k equal to 0 to have the service calculate and use the optimal value. When k equals 0 on input, the calculated optimal value of k is returned on output.

source_text_id

Direction	Type
Input	Integer

The ALET of the *source_text* parameter to be enciphered.

reserved_length

Direction	Type
Input	Integer

Length in bytes of the *reserved* parameter. The value must be 0.

reserved

Direction	Type
Input	String

This field is ignored.

source_text_length

Direction	Type
Input	Integer

The length of the *source_text* parameter in bytes. The maximum value is 2,147,836,647. A zero value is valid.

source_text

Direction	Type
Input	String

The input text to be enciphered. The string must contain only the characters of the character set defined by the *charset* keyword. The service does not enforce this.

target_text_id

Direction	Type
Input	Integer

The ALET of the *target_text* parameter.

target_text_length

Direction	Type
Input/Output	Integer

On input, this parameter specifies the size of the storage pointed to by the *target_text* parameter. On output, this parameter has the actual length of the text stored in the buffer addressed by the *target_text* parameter. The value must not be less that the value in *source_text_length*.

target_text

Direction	Type
Output	String

The enciphered text returned by the service.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

To use a fixed-length CKDS encrypted key, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES).

To use a fixed-length encrypted key that does not reside in the CKDS, the ICSF segment of the CSFKEYS class general resource profile CSF-PROTECTED-KEY-TOKEN (or its generic equivalent) must contain SYMCPACFWRAP(YES) and the caller must have SAF access to the profile.

To use an encrypted variable-length AES CIPHER key token, the key token must allow encryption, any cipher mode, and export to CPACF protected key format.

No pre-processing or post-processing exits are enabled for this service.

The master keys need to be loaded only when using this service with an encrypted key.

The AESVFPE algorithm uses hardware if it is available. Otherwise, clear key operations are performed in software.

This service fails if execution would cause destructive overlay of the *source_text* parameter.

For multi-part encryption, the source text length should be a multiple of k bytes on each call until the last call, which can be less.

For rule ORDINAL, specifying k equals 0 indicates that the service should calculate, use, and return the optimal k value. The algorithm used to calculate the optimal k value is as follows:

- Set k equal to the highest power of n that is less than or equal to 2 to the b power (where b equals 64 for TDES and 128 for AES).
- If n to the k power does not evenly divide 2 to the b power, then set k equal to k-1.

Note: For certain values of n, subtracting 1 from k may result in a value that is, in fact, not optimal when encrypting large amounts of plain text. The service makes no attempt to distinguish these cases.

The encryption and decryption subkey is derived from the base key according to NIST SP 800-108, 'Recommendation for Key Derivation Pseudorandom Functions'. The KDF in Counter Mode variant is used as follows:

```
K1, K2, … Kn = PRF(Key,i || key_data_data || blen)
```

where

- The PRF is CMAC (See NIST SP 800-38B, 'Recommendation for Black Cipher Modes of Operation: The CMAC Mode for Authentication').
- *i* is one byte iteration variable.
- *blen* is the one byte bit length of the desired key (minus 1).

Note: The encryption and decryption subkey is always the same type and size as the base key.

Access control points

When the label of an encrypted key is specified for the *key_identifier* parameter, the appropriate access control points must be enabled.

Table 329. Access control points for Field Level Encipher

Key algorithm	Access control point
Crypto Express5 and earlier CCA coprocessors	
AESVFPE	High-performance secure AES keys
TDESVFPE	High-performance secure DES keys
Crypto Express6 and later CCA coprocessors	
All	Authenticated Key Export - EXPTSK

For AES CIPHER keys (CEX6C and later), the following access controls must be enabled:

- **Authenticated Key Export - DRVTXKEY**
- **Authenticated Key Export - EXPTSK**
- **Authenticated Key Export - SETSNKEY**

These access controls should be enabled by default, but **Authenticated Key Export - EXPTSK** may not be enabled and will cause return codes that are misleading.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 330. Field Level Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions Crypto Express 5 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. DES and AES CIPHER key tokens and triple-length DES DATA keys with a non-zero control vector are not supported. Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.

Table 330. Field Level Encipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. Encrypted variable-length AES CIPHER key tokens require a CEX6C. Triple-length DES DATA keys with a non-zero control vector require a CEX6C with the November 2018 or later licensed internal code (LIC). DES CIPHER key tokens require a CEX6C with the P41458.002 or later MCL. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Encrypted keys require a Crypto express coprocessor. DES and AES CIPHER key tokens and triple-length DES DATA keys with a non-zero control vector are not supported. X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Encrypted keys require a Crypto express coprocessor. X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Related information

You **cannot** overlap the source and target text fields. For example:

```
ssssss
     tttttt  is not supported.

tttttt
     ssssss  is not supported.

ssssssttttttt is supported.

ssssss
```

```
tttttt  is supported.

s represents the source text and t represents the target text.
```

Format Preserving Algorithms Decipher (CSNBFFXD and CSNEFFXD)

The Format Preserving Algorithms Decipher callable service implements Format Preserving Encryption Algorithms (FF1, FF2, and FF2.1). This service returns the decrypted data which is equal in terms of length and alphabet characters to the input plain text.

The callable service name for AMODE(64) invocation is CSNEFFXD.

Format

```
CALL CSNBFFXD(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             tweak_length,
             tweak,
             tweak_alphabet_length,
             tweak_alphabet,
             alphabet_length,
             alphabet,
             ciphertext_length,
             ciphertext,
             reserved1_length,
             reserved1,
             reserved2_length,
             reserved2,
             plaintext_length,
             plaintext)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 3.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 331. Rule array keywords for Format Preserving Algorithms Decipher control information

Keyword	Meaning
Encryption method format (one, required)	
FF1	Specifies to use the FPE FF1 algorithm to decrypt the data (original FFX).
FF2	Specifies to use the FPE FF2 algorithm to decrypt the data (original VAES).
FF2.1	Specifies to use the FPE FF2.1 algorithm to decrypt the data (new version of VAES).
Encryption Algorithm process (one, required)	
AES	Specifies use of the AES ciphering algorithm.
Alphabet (one, required)	
BASE10	Specifies that the input data will be only BASE-10 ASCII represented in binary form. Valid ASCII values are '0' through '9' (X'30' through X'39'). This applies to the tweak as well as the alphabet.
CUSTOM	Specifies that the caller can provide the alphabet for the input data as well as the tweak. The tweak alphabet is optional. The input data, tweak, and alphabets are ASCII character sets.

key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *key_identifier* parameter.

If the *key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to decrypt the clear text. The key identifier is an operational token or the key label of an operational token in key storage.

For CCA keys, the identifier is a variable-length AES key token of key type CIPHER with key usage attribute DECRYPT enabled and the desired enciphering mode of FF1, FF2, or FF2.1 enabled.

For X9.143 keys, the identifier is a variable-length key block of an AES data-encrypting key: key usage D0, algorithm A, and mode of use B or D.

When the FF1 keyword is specified, this key can be either a 128-bit or a 256-bit key. When the FF2 or FF2.1 keyword is specified, this key must be a 128-bit key.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

tweak_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *tweak* parameter.

For the FF1 keyword
The value can be between 0 and 512 inclusive.

For the FF2 or FF2.1 keyword
Minimum can be 0 and the maximum must satisfy this formula depending on the *tweak_alphabet_length* value:

```
(tweak _length x lg2(tweak_alphabet_length)) <= (15 – 2) x 8
```

tweak

Direction	Type
Input	String

The data to be used as the tweak value in the decryption calculations. When the *tweak _length* is zero, this parameter is ignored.

tweak_alphabet_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *tweak_alphabet* parameter. The value must be zero when the BASE10 keyword is specified.

When the CUSTOM keyword is specified:

For the FF1 keyword
The value must be 256.

For the FF2 or FF2.1 keyword
The value must be between 8 and 256 inclusive.

tweak_alphabet

Direction	Type
Input	String

The tweak alphabet data to be used in the decryption calculations. When the *tweak_alphabet_length* is zero, this parameter is ignored.

alphabet_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *alphabet* parameter. The value must be zero when the BASE10 keyword is specified. Otherwise, the value may be zero or between 8 and 256 inclusive.

alphabet

Direction	Type
Input	String

The alphabet data to be used in the decryption calculations. When the *alphabet_length* is zero, this parameter is ignored.

ciphertext_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *ciphertext* parameter.

For the FF1 keyword
The value must be between 2 and 504.

For the FF2 or FF2.1 keyword
The minimum is 2 and the maximum must satisfy this formula depending on the *alphabet_length* value:

```
(ciphertext_length * lg2(alphabet_length))/2 <= (15-1) * 8
```

For example, when the *alphabet_length* is 10, the maximum *ciphertext_length* is 31.

```
104/log2(10) = 31
```

ciphertext

Direction	Type
Input	String

The encrypted text to be decrypted.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input/Output	String

This parameter is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input/Output	String

This parameter is ignored.

plaintext_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *plaintext* parameter. On input, the value is the size of the buffer to receive the output *plaintext*. The value must be at least the same as the *ciphertext_length*. On output, the value is the length of the data returned in the *plaintext* parameter.

plaintext

Direction	Type
Output	String

The deciphered text returned.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS, PKDS, or TKDS.

Access control points

The **Format Preserving Algorithms Decipher** access control point controls the use of this service.

Table 332. Access controls in the domain role that control the function of the Format Preserving Algorithms Decipher service

Rule-array keyword	Access control
FF1	Format Preserving Algorithms Encipher/Decipher – Allow FF1.
FF2	Format Preserving Algorithms Encipher/Decipher – Allow FF2.
FF2.1	Format Preserving Algorithms Encipher/Decipher – Allow FF2.1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 333. Format Preserving Algorithms Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**		This service is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**		This service is not supported. X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	This service requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Format Preserving Algorithms Encipher (CSNBFFXE and CSNEFFXE)

The Format Preserving Algorithms Encipher callable service implements Format Preserving Encryption Algorithms (FF1, FF2, and FF2.1). This service returns the encrypted data which is equal in terms of length and alphabet characters to the input plain text.

The callable service name for AMODE(64) invocation is CSNEFFXE.

Format

```
CALL CSNBFFXE(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier,
             tweak_length,
             tweak,
             tweak_alphabet_length,
             tweak_alphabet,
             alphabet_length,
```

```
            alphabet,
            plaintext_length,
            plaintext,
            reserved1_length,
            reserved1,
            reserved2_length,
            reserved2,
            ciphertext_length,
            ciphertext)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 3.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 334. Rule array keywords for Format Preserving Algorithms Encipher control information	
Keyword	**Meaning**
Encryption method format (one, required)	
FF1	Specifies to use the FPE FF1 algorithm to encrypt the data (original FFX).
FF2	Specifies to use the FPE FF2 algorithm to encrypt the data (original VAES).
FF2.1	Specifies to use the FPE FF2.1 algorithm to encrypt the data (new version of VAES).
Encryption Algorithm process (one, required)	
AES	Specifies use of the AES ciphering algorithm.
Alphabet (one, required)	
BASE10	Specifies that the input data will be only BASE-10 ASCII represented in binary form. Valid ASCII values are '0' through '9' (X'30' through X'39'). This applies to the tweak as well as the alphabet.
CUSTOM	Specifies that the caller can provide the alphabet for the input data as well as the tweak. The tweak alphabet is optional. The input data, tweak, and alphabets are ASCII character sets.

key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *key_identifier* parameter.

If the *key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to encrypt the clear text. The key identifier is an operational token or the key label of an operational token in key storage.

For CCA keys, the identifier is a variable-length AES key token of key type CIPHER with key usage attribute ENCRYPT enabled and the desired enciphering mode of FF1, FF2, or FF2.1 enabled.

For X9.143 keys, the identifier is a variable-length key block of an AES data-encrypting key: key usage D0, algorithm A, and mode of use B or E.

When the FF1 keyword is specified, this key can be either a 128-bit or a 256-bit key. When the FF2 or FF2.1 keyword is specified, this key must be a 128-bit key.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

tweak_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *tweak* parameter.

For the FF1 keyword
The value can be between 0 and 512 inclusive.

For the FF2 or FF2.1 keyword
Minimum can be 0 and the maximum must satisfy this formula depending on the *tweak_alphabet_length* value:

```
(tweak _length x lg2(tweak_alphabet_length)) <= (15 – 2) x 8
```

tweak

Direction	Type
Input	String

The data to be used as the tweak value in the encryption calculations. When the *tweak _length* is zero, this parameter is ignored.

tweak_alphabet_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *tweak_alphabet* parameter. The value must be zero when the BASE10 keyword is specified.

When the CUSTOM keyword is specified:

For the FF1 keyword
The value must be 256.

For the FF2 or FF2.1 keyword
The value must be between 8 and 256 inclusive.

tweak_alphabet

Direction	Type
Input	String

The tweak alphabet data to be used in the encryption calculations. When the *tweak_alphabet_length* is zero, this parameter is ignored.

alphabet_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *alphabet* parameter. The value must be zero when the BASE10 keyword is specified. Otherwise, the value may be zero or between 8 and 256 inclusive.

alphabet

Direction	Type
Input	String

The alphabet data to be used in the encryption calculations. When the *alphabet_length* is zero, this parameter is ignored.

plaintext_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *plaintext* parameter.

For the FF1 keyword
The value must be between 2 and 504.

For the FF2 or FF2.1 keyword
The minimum is 2 and the maximum must satisfy this formula depending on the *alphabet_length* value:

```
(plaintext_length * lg2(alphabet_length))/2 <= (15-1) * 8
```

For example, when the *alphabet_length* is 10, the maximum *plaintext_length* is 31.

```
104/log2(10) = 31
```

plaintext

Direction	Type
Input	String

The input clear text.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input/Output	String

This parameter is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input/Output	String

This parameter is ignored.

ciphertext_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *ciphertext* parameter.

On input, the value is the size of the buffer to receive the output ciphertext. The value must be at least the same the *plaintext_length*.

On output, the value is the length of the data returned in the ciphertext parameter.

ciphertext

Direction	Type
Output	String

The enciphered text returned.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS, PKDS, or TKDS.

Access control points

The **Format Preserving Algorithms Encipher** access control point controls the use of this service.

Table 335. Access controls in the domain role that control the function of the Format Preserving Algorithms Encipher service

Rule-array keyword	Access control
FF1	Format Preserving Algorithms Encipher/Decipher – Allow FF1.
FF2	Format Preserving Algorithms Encipher/Decipher – Allow FF2.
FF2.1	Format Preserving Algorithms Encipher/Decipher – Allow FF2.1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 336. Format Preserving Algorithms Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**		This service is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**		This service is not supported. X9.143 key blocks are not supported.

Table 336. Format Preserving Algorithms Encipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	This service requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT)

The Format Preserving Algorithms Translate callable service implements Format Preserving Encryption Algorithms (FF1, FF2, and FF2.1). This service returns the translated data for the requested format.

The callable service name for AMODE(64) invocation is CSNEFFXT.

Format

```
CALL CSNBFFXT(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             input_key_identifier_length,
             input_key_identifier,
             output_key_identifier_length,
             output_key_identifier,
             input_tweak_length,
             input_tweak,
             input_tweak_alphabet_length,
             input_tweak_alphabet,
             input_alphabet_length,
             input_alphabet,
             output_tweak_length,
             output_tweak,
             output_tweak_alphabet_length,
             output_tweak_alphabet,
             output_alphabet_length,
             output_alphabet,
             reserved1_length,
             reserved1,
             reserved2_length,
             reserved2,
             ciphertext_length,
             ciphertext)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 5.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 337. Rule array keywords for Format Preserving Algorithms Translate control information

Keyword	Meaning
Input ciphertext format (one, required)	
ICFF1	Specifies the input ciphertext was encrypted using FPE FF1 algorithm (original FFX).

Table 337. Rule array keywords for Format Preserving Algorithms Translate control information (continued)	
Keyword	**Meaning**
ICFF2	Specifies the input ciphertext was encrypted using FPE FF2 algorithm (original VAES).
ICFF2.1	Specifies the input ciphertext was encrypted using FPE FF2.1 algorithm (new version of VAES).
Output ciphertext format (one, required)	
OCFF1	Specifies the data is to be encrypted using FPE FF1 algorithm (original FFX).
OCFF2	Specifies the data is to be encrypted using FPE FF2 algorithm (original VAES).
OCFF2.1	Specifies the data is to be encrypted using FPE FF2.1 algorithm (new version of VAES).
Encryption Algorithm process (one, required)	
AES	Specifies use of the AES ciphering algorithm.
Input alphabet (one, required)	
I-BASE10	Specifies that the input data will be only BASE-10 ASCII represented in binary form. Valid ASCII values are '0' through '9' (X'30' through X'39'). This applies to the tweak as well as the alphabet.
I-CUSTOM	Specifies that the caller can provide the alphabet for the input data as well as the tweak. The tweak alphabet is optional. The input data, tweak, and alphabets are ASCII character sets.
Output alphabet (one, required)	
O-BASE10	Specifies that the input data will be only BASE-10 ASCII represented in binary form. Valid ASCII values are '0' through '9' (X'30' through X'39'). This applies to the tweak as well as the alphabet.
O-CUSTOM	Specifies that the caller can provide the alphabet for the input data as well as the tweak. The tweak alphabet is optional. The input data, tweak, and alphabets are ASCII character sets.

input_key_identifier_length

Direction	**Type**
Input	Integer

Specifies the length in bytes of the *input_key_identifier* parameter.

If the *input_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

input_key_identifier

Direction	**Type**
Input/Output	String

The identifier of the data-encrypting key to decrypt the input cipher text. The key identifier is an operational token or the key label of an operational token in key storage.

For CCA keys, the identifier is a variable-length AES key token of key type CIPHER with key usage attribute DECRYPT enabled and the desired enciphering mode of FF1, FF2, or FF2.1 enabled.

For X9.143 keys, the identifier is a variable-length key block of an AES data-encrypting key: key usage D0, algorithm A, and mode of use B or D.

When the ICFF1 keyword is specified, this key can be either a 128-bit or a 256-bit key. When the ICFF2 or ICFF2.1 keyword is specified, this key must be a 128-bit key.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

output_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *output_key_identifier* parameter.

If the *output_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

output_key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to encrypt the plain text. The key identifier is an operational token or the key label of an operational token in key storage.

For CCA keys, the identifier is a variable-length AES key token of key type CIPHER with key usage attribute ENCRYPT enabled and the desired enciphering mode of FF1, FF2, or FF2.1 enabled.

For X9.143 keys, the identifier is a variable-length key block of an AES data-encrypting key: key usage D0, algorithm A, and mode of use B or E.

When the OCFF1 keyword is specified, this key can be either a 128-bit or a 256-bit key. When the OCFF2 or OCFF2.1 keyword is specified, this key must be a 128-bit key.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

input_tweak_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_tweak* parameter.

For the ICFF1 keyword
The value can be between 0 and 512 inclusive.

For the ICFF2 or ICFF2.1 keyword
Minimum can be 0 and the maximum must satisfy this formula depending on the *input_tweak_alphabet_length* value:

```
(input_tweak _length x lg2(input_tweak_alphabet_length)) <= (15 – 2) x 8
```

input_tweak

Direction	Type
Input	String

The data to be used as the tweak value in the input decryption calculations. When the *input_tweak_length* is zero, this parameter is ignored.

input_tweak_alphabet_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_tweak_alphabet* parameter. The value must be zero when the I-BASE10 keyword is specified.

When the I-CUSTOM keyword is specified:

For the ICFF1 keyword
The value must be 256.

For the ICFF2 or ICFF2.1 keyword
The value must be between 8 and 256 inclusive.

input_tweak_alphabet

Direction	Type
Input	String

The tweak alphabet data to be used in the input decryption calculations. When the *input_tweak_alphabet_length* is zero, this parameter is ignored.

input_alphabet_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_alphabet* parameter. The value must be zero when the I-BASE10 keyword is specified. Otherwise, the value may be zero or between 8 and 256 inclusive.

input_alphabet

Direction	Type
Input	String

The alphabet data to be used in the input decryption calculations. When the *input_alphabet_length* is zero, this parameter is ignored.

output_tweak_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *output_tweak* parameter.

For the OCFF1 keyword
The value can be between 0 and 512 inclusive.

For the OCFF2 or OCFF2.1 keyword
Minimum can be 0 and the maximum must satisfy this formula depending on the *output_tweak_alphabet_length* value:

```
(output_tweak _length x lg2(output_tweak_alphabet_length)) <= (15 – 2) x 8
```

output_tweak

Direction	Type
Input	String

The data to be used as the tweak value in the output encryption calculations. When the *output_tweak _length* is zero, this parameter is ignored.

output_tweak_alphabet_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *output_tweak_alphabet* parameter. The value must be zero when the O-BASE10 keyword is specified.

When the O-CUSTOM keyword is specified:

For the OCFF1 keyword
The value must be 256.

For the OCFF2 or OCFF2.1 keyword
The value must be between 8 and 256 inclusive.

output_tweak_alphabet

Direction	Type
Input	String

The tweak alphabet data to be used in the output encryption calculations. When the *output_tweak_alphabet_length* is zero, this parameter is ignored.

output_alphabet_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *output_alphabet* parameter. The value must be zero when the O-BASE10 keyword is specified. Otherwise, the value may be zero or between 8 and 256 inclusive.

output_alphabet

Direction	Type
Input	String

The alphabet data to be used in the output encryption calculations. When the *output_alphabet_length* is zero, this parameter is ignored.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input/Output	String

This parameter is ignored.

reserved2_length

Direction	Type
Input/Output	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input/Output	String

This parameter is ignored.

ciphertext_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *ciphertext* parameter. On input, the value is the length the input ciphertext. On output, the value is the length of the data returned in the ciphertext parameter.

For the OCFF1 keyword
The value must be between 2 and 504 inclusive.

For the OCFF2 or OCFF2.1 keyword
The minimum is 2 and the maximum must satisfy this formula depending on the *output_alphabet_length* value:

```
(ciphertext_length x lg2(output_alphabet_length))/2 <= (15-1) x 8
```

For example, when the *output_alphabet_length* is 10, the maximum *ciphertext_length* is 31.

```
104/log2(10) = 31
```

ciphertext

Direction	Type
Input/Output	String

The enciphered text to be processed.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS, PKDS, or TKDS.

Access control points

The **Format Preserving Algorithms Translate** access control point controls the use of this service.

Table 338. Access controls in the domain role that control the function of the Format Preserving Algorithms Translate service	
Rule-array keyword	**Access control**
FF1	Format Preserving Algorithms Encipher/Decipher – Allow FF1.
FF2	Format Preserving Algorithms Encipher/Decipher – Allow FF2.
FF2.1	Format Preserving Algorithms Encipher/Decipher – Allow FF2.1.

To permit the output cipher key to be weaker than the input cipher key, enable the **Format Preserving Algorithms Translate - Allow weaker output key** access control.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 339. Format Preserving Algorithms Translate required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**		This service is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**		This service is not supported. X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	This service requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

FPE Decipher (CSNBFPED and CSNEFPED)

The FPE Decipher callable service is used to decrypt payment card data for the Visa Data Secure Platform (Visa DSP) processing. This service supports two options:

- The standard encryption option.
- The Visa Format Preserving Encryption (VFPE) option.

If the standard encryption option was selected, the plain text data was formatted into blocks and then encrypted with triple-DES encryption with a static TDES key or a DUKPT double length data encryption key. For the decryption operation, the data blocks must be decrypted and unblocked to produce the plaintext. If the data was encrypted with the VFPE option, the data was encrypted in place without changing the data type or length of the field and DUKPT key management is used.

This service can be used to decrypt one or all of the following fields: the primary account number (PAN), the cardholder name, the track 1 discretionary data, or the track 2 discretionary data.

There are three decryption options:

1. Decrypt standard option with CBC mode TDES and DUKPT keys.
2. Decrypt VFPE option with DUKPT keys.
3. Decrypt standard option with CBC mode TDES and double-length TDES keys.

To use this service, you must specify the following:

- The processing method, which is limited to Visa Data Secure Platform (Visa DSP).
- The key management method, either STATIC or DUKPT.
- The algorithm, which is limited to TDES.
- The mode, either CBC or Visa Format Preserving Encryption (VFPE).
- The plaintext to be decrypted.
- The character set of each field to be decrypted using rule-array keywords.
- The base derivation key and either the key serial number for DES-DUKPT or the AES-DUKPT derivation data for AES-DUKPT, or a double-length TDES key if STATIC key management is used.
- A compliance or non-compliance indicator for the check digit of the PAN to be processed if VFPE is specified.

The service returns the decrypted fields and optionally, the DUKPT PIN key, if the DUKPT key management is selected and the PINKEY rule is specified.

The callable service name for AMODE(64) invocation is CSNEFPED.

Format

```
CALL CSNBFPED(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              enc_PAN_length,
              enc_PAN,
              enc_CH_name_length,
              enc_CH_name,
              enc_dtrack1_data_length,
              enc_dtrack1_data,
              enc_dtrack2_data_length,
              enc_dtrack2_data,
              key_identifier_length,
              key_identifier,
              derivation_data_length,
              derivation_data,
              clear_PAN_length,
              clear_PAN,
              clear_CH_name_length,
              clear_CH_name,
              clear_dtrack1_data_length,
              clear_dtrack1_data,
              clear_dtrack2_data_length,
              clear_dtrack2_data,
              DUKPT_PIN_key_identifier_length,
              DUKPT_PIN_key_identifier,
              reserved1_length,
              reserved1,
```

```
                reserved2_length,
                reserved2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 5. The maximum value is 10.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Note: At least one character set keyword is required.

Table 340. Rule array keywords for FPE Decipher	
Keyword	**Meaning**
Processing method (required)	
VMDS	Specifies that the Visa DSP method (formally known as the Visa Merchant Data Secure method) is to be used for processing.
Key management method (one required)	
STATIC	Specifies the use of double length (2-key) triple-DES symmetric keys. This is a non-DUKPT key.
DUKPT	Specifies the use of the transaction unique general purpose Data Encryption Keys generated by the DUKPT process at the point of service for data encryption. This is required if VFPE mode is specified. Otherwise, this is optional. Both DES DUKPT and AES DUKPT key derivation methods are supported. The content of the *derivation_data* parameter will determine which DUKPT method is used.
Algorithm (required)	
TDES	Specifies the use of CBC mode triple-DES encryption.
Mode (one required)	
CBC	Specifies the use of CBC mode. This is the mode for the standard encryption option.
VFPE	Specifies the use of Visa format preserving encryption.
PAN input output character set (one required if the clear_PAN_length variable is greater than 0. Otherwise, it is not allowed.)	
PAN8BITA	Specifies that the PAN data character set is ASCII represented in binary form. Valid ASCII values are '0' through '9' (X'30' through X'39').
PAN4BITX	Specifies that the PAN data character set is 4-bit hex. Two digits per byte. Valid 4-bit hexadecimal values are X'0' through X'9'.
Cardholder name input output character set (required if the clear_CH_name_length variable is greater than 0. Otherwise, it is not allowed.)	
CN8BITA	Specifies that the cardholder name character set is ASCII represented in binary format, one character per byte. See Table 256 on page 616 for valid characters.
Track_1 input output character set (required if the clear_dtrack1_data_length variable is greater than 0. Otherwise, it is not valid.)	
TK18BITA	Specifies that the track 1 discretionary data character set is ASCII represented in binary format, one character per byte. See Table 256 on page 616 for valid characters.
Track_2 input output character set (required if the clear_dtrack2_data_length variable is greater than 0. Otherwise, it is not valid.)	
TK28BITA	Specifies that the track 2 discretionary data character set is ASCII represented in binary format, one character per byte. Valid ASCII values are '0' - '9' (X'30' - X'39') and 'A' - 'F' (X'41' - X'46'.

Table 340. Rule array keywords for FPE Decipher (continued)	
Keyword	**Meaning**
PIN encryption key output selection (one, optional, if DUKPT is specified. Otherwise, it is not valid.)	
NOPINKEY	Do not return a DUKPT PIN encryption key. This is the default.
PINKEY	Return a DUKPT PIN encryption key.
PAN check digit compliance (one required if mode VFPE and the pan character set keyword is present. Otherwise, it is not allowed.)	
CMPCKDGT	Last digit of the PAN contains a compliant check digit per ISO/IEC 7812-1.
NONCKDGT	Last digit of the PAN does not contain a compliant check digit per ISO/IEC 7812-1.

enc_PAN_length

Direction	Type
Input	Integer

Specifies the number of bytes of data in the *enc_PAN* parameter if the mode is CBC or the number of PAN digits if the mode is VFPE. The value is 0 or 15 through 19 for VFPE. The value must be 0 or 16 if the standard option with CBC mode is selected. The value is zero when the PAN has not been presented for decryption.

enc_PAN

Direction	Type
Input	String

The enciphered primary account number (PAN) that is to be decrypted. For VFPE mode, if the PAN contains an odd number of 4-bit digits, the data is left justified in the PAN variable and the right-most 4 bits are ignored.

enc_CH_name_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *enc_CH_name* parameter. The input value is 0 or between 1 and 32, inclusive for VFPE. For the standard method, the input value is 0 or 2 through 32 for VFPE. For CBC mode, the input value is 0, 16, 24, 32, or 40. The value is zero when the cardholder name has not been presented for decryption.

enc_CH_name

Direction	Type
Input	String

The enciphered cardholder full name that is to be decrypted. Only characters in Table 256 on page 616 are valid.

enc_dtrack1_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *enc_dtrack1_data* parameter. The input value is 0 or between 1 and 56 inclusive for VFPE. For the standard method, the input value is 0 or 1 through 56 for VFPE. For CBC mode, the input value is 0 or 16, 24, 32, 40, 48, 56, or 64. The value is zero when the track 1 discretionary data has not been presented for decryption.

enc_dtrack1_data

Direction	Type
Input	String

The encrypted track 1 data that is to be decrypted. Only characters in Table 256 on page 616 are valid.

enc_dtrack2_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *enc_dtrack2_data* parameter. The input value is 0 or 1 through 19 for VFPE. For mode CBC, the input value is 0, 8, or 16. The value is zero when the track 2 discretionary data is not been presented for decryption.

enc_dtrack2_data

Direction	Type
Input	String

The encrypted track 2 data that is to be decrypted.

key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *key_identifier* parameter.

If the *key_identifier* parameter contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The identifier of the key that is used to either decrypt the card data (key management STATIC) or derive the *DUKPT_PIN_key_identifer* (key management DUKPT). The *key identifier* is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA key tokens

For DES-DUKPT, the key type must be KEYGENKY. In addition, it must have a control vector with bit 18 equal to B'1' (UKPT). The base derivation key is the one from which the operational keys are derived using the DUKPT algorithm defined in ANSI X9.24-1 2007.

For AES-DUKPT, this is an AES DKYGENKY with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1. The base derivation key is the one from which the operational keys are derived using the DUKPT algorithm defined in ANSI x9.24-3 2017.

For key management STATIC, (Zone Encryption Key in the Visa DSP specification), the key type must be either CIPHER or DECIPHER. For production purposes, it is recommended that the key have left and right halves that are not equal.

Note: Data keys are not supported.

For X9.143 (TR-31) key blocks

For DES-DUKPT, the key usage is B0, algorithm is T, and mode of use is X.

For AES-DUKPT, the key usage is B0, algorithm is A, and mode of use is X.

For key management STATIC, the key usage is D0, algorithm is T, and mode of use is B or D.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

derivation_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *derivation_data* parameter. To specify the DES DUKPT method, set the value to 10 if the key management method DUKPT is specified in the rule array. To specify the AES DUKPT method, set the value to 20 if the key management method DUKPT is specified in the rule array. Otherwise, this value must be 0.

derivation_data

Direction	Type
Input	String

When specifying the DES-DUKPT method, *derivation_data* contains the 80 bit (10 byte) derivation data that is used as input to the DUKPT derivation process. The derivation data contains the current key serial number (CKSN), which is composed of the 59 bit initial key serial number value concatenated with the 21 bit value of the current encryption counter, which the device increments for each new transaction. This field is in binary format.

When specifying the AES-DUKPT method, *derivation_data* contains the 20 byte AES-DUKPT derivation data. See Table 684 on page 1719 for the layout of the AES-DUKPT derivation data. The algorithm indicator must be set to X'0000' (2- key TDES). The key usage indicator must be set to X'1000' (PIN Encryption).

clear_PAN_length

Direction	Type
Input/Output	Integer

Specifies the number of PAN digits in the *clear_PAN* parameter. This value must be 0 or between 15 and 19, inclusive on output.

clear_PAN

Direction	Type
Output	String

The field where the deciphered primary account number is returned. The full account number, including check digit, is recovered. The data for this parameter is returned in binary format. It is the binary representation of 4-bit hex (keyword PAN4BITX) or ASCII (keyword PAN8BITA) as indicated by the supplied rule array keyword. The clear PAN is left justified in this field.

clear_CH_name_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *clear_CH_name* parameter. This output value is 0 or 2 through 32 on output. The variable can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

clear_CH_name

Direction	Type
Output	String

The field where the deciphered cardholder full name is returned. The output data for this parameter is in binary format. It is the binary representation of ASCII as indicated by the supplied rule array keyword.

clear_dtrack1_data_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *clear_dtrack1_data* parameter. The output value is 0 or 1 through 56. The value can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

clear_dtrack1_data

Direction	Type
Output	String

The field where the deciphered discretionary track 1 data is returned.

clear_dtrack2_data_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *clear_dtrack2_data* parameter. The output value is 0 or 1 through 19. The value can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

clear_dtrack2_data

Direction	Type
Output	String

The field where the deciphered discretionary track 2 data is returned.

DUKPT_PIN_key_identifier_length

Direction	Type
Input/Output	Integer

The length in bytes of the *DUKPT_PIN_key_identifier* parameter. When the NOPINKEY rule-array keyword is specified, the value is 0.

When PINKEY is specified:

- For CCA key tokens, the value is 64.
- For X9.143 key blocks, the maximum value is 9992. The value should be large enough to hold the derived key block.

On output, the variable is updated with the length of the data returned in the *DUKPT_PIN_key_identifier* variable.

DUKPT_PIN_key_identifier

Direction	Type
Input/Output	String

The derived PIN-encrypting key.

For CCA key tokens

On input, must contain a DES OPINENC or IPINENC skeleton token.

On output, *DUKPT_PIN_key_identifier* contains the DES token with the derived DES OPINENC or IPINENC key.

For X9.143 (TR-31) key blocks

On input, must contain a TDES PIN-encrypting key skeleton key block (key usage P0, algorithm T, mode of use B, D, or E)

On output, contains the key block with the derived PIN-encrypting key.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **FPE Decipher** access control point in the domain role controls the function of this service.

Specifying the DUKPT keyword requires that the **DUKPT - PIN Verify, PIN Translate** access control point be enabled in the domain role.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 341. FPE Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The AES-DUKPT algorithm requires the October 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	The AES-DUKPT algorithm requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

FPE Encipher (CSNBFPEE and CSNEFPEE)

The FPE Encipher callable service is used to encrypt payment card data for the Visa Data Secure Platform (Visa DSP) processing. This service supports two options:

- The standard encryption option, which uses the CBC mode TDES.
- The Visa Format Preserving Encryption (VFPE) option.

If the standard encryption option is selected, data is formatted into blocks and then encrypted with triple-DES encryption with either static TDES keys or with DUKPT keys. If the data is encrypted with the VFPE option, the data is encrypted without changing the data type or length of the field, and DUKPT key management is used.

This service can be use to encrypt one or all of the following fields: the primary account number (PAN), the cardholder name, the track 1 discretionary data, or the track 2 discretionary data. The additional data field (expiration date and service code) are not encrypted with this process.

There are three encryption options:

1. Standard option with CBC mode TDES and DUKPT keys.
2. VFPE option with DUKPT keys.
3. Standard option with CBC mode TDES and static TDES keys.

To use this service, you must specify the following:

- The processing method, which is limited to Visa Data Secure Platform (Visa DSP).
- The key management method, either STATIC or DUKPT.
- The algorithm, which is limited to TDES.
- The mode, either CBC or Visa Format Preserving Encryption (VFPE).
- The plaintext to be encrypted.
- The character set of each field to be encrypted using rule-array keywords.
- The base derivation key and either the key serial number for DES-DUKPT or the AES-DUKPT derivation data for AES-DUKPT, or a double-length TDES key if STATIC key management is used.
- A compliance or non-compliance indicator for the check digit of the PAN to be processed.

The service returns the encrypted fields and optionally, the DUKPT PIN key, if the DUKPT key management is selected and the PINKEY rule is specified.

The callable service name for AMODE(64) invocation is CSNEFPEE.

Format

```
CALL CSNBFPEE(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             clear_PAN_length,
             clear_PAN,
             clear_CH_name_length,
             clear_CH_name,
             clear_dtrack1_data_length,
             clear_dtrack1_data,
             clear_dtrack2_data_length,
             clear_dtrack2_data,
             key_identifier_length,
             key_identifier,
             derivation_data_length,
             derivation_data,
             enc_PAN_length,
             enc_PAN,
             enc_CH_name_length,
             enc_CH_name,
             enc_dtrack1_data_length,
             enc_dtrack1_data,
             enc_dtrack2_data_length,
             enc_dtrack2_data,
             DUKPT_PIN_key_identifier_length,
             DUKPT_PIN_key_identifier,
             reserved1_length,
```

```
                reserved1,
                reserved2_length,
                reserved2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 5. The maximum value is 10.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Note: At least one character set keyword is required.

Table 342. Rule array keywords for FPE Encipher	
Keyword	**Meaning**
Processing method (required)	
VMDS	Specifies that the Visa DSP method (formally known as the Visa Merchant Data Secure method) is to be used for processing.
Key management method (one required)	
STATIC	Specifies the use of double length (2-key) triple-DES symmetric keys. This is a non-DUKPT key.
DUKPT	Specifies the use of the transaction unique general purpose Data Encryption Keys generated by the DUKPT process at the point of service for data encryption. This is required if VFPE mode is specified. Otherwise, this is optional. Both DES DUKPT and AES DUKPT key derivation methods are supported. The content of the *derivation_data* parameter will determine which DUKPT method is used.
Algorithm (required)	
TDES	Specifies the use of CBC mode triple-DES encryption.
Mode (one required)	
CBC	Specifies the use of CBC mode. This is the mode for the standard encryption option.
VFPE	Specifies the use of Visa format preserving encryption.
PAN input output character set (one required if the clear_PAN_length variable is greater than 0. Otherwise, it is not allowed.)	
PAN8BITA	Specifies that the PAN data character set is BASE-10 ASCII represented in binary form. Valid ASCII values are '0' through '9' (X'30' through X'39').
PAN4BITX	Specifies that the PAN data character set is BASE-10 4-bit hex. Two digits per byte. Valid 4-bit hexadecimal values are X'0' through X'9'.
Cardholder name input output character set (required if the clear_CH_name_length variable is greater than 0. Otherwise, it is not allowed.)	
CN8BITA	Specifies that the cardholder name character set is ASCII represented in binary format, one character per byte. See Table 256 on page 616 for valid characters.
Track_1 input output character set (required if the clear_dtrack1_data_length variable is greater than 0. Otherwise, it is not valid.)	
TK18BITA	Specifies that the track 1 discretionary data character set is ASCII represented in binary format, one character per byte. See Table 256 on page 616 for valid characters.
Track_2 input output character set (required if the clear_dtrack2_data_length variable is greater than 0. Otherwise, it is not valid.)	
TK28BITA	Specifies that the track 2 discretionary data character set is ASCII represented in binary format, one character per byte. Valid ASCII values are '0' - '9' (X'30' - X'39') and 'A' - 'F' (X'41' - X'46'.

Table 342. Rule array keywords for FPE Encipher (continued)	
Keyword	**Meaning**
PIN encryption key output selection (one, optional)	
NOPINKEY	Do not return a DUKPT PIN encryption key. This is the default.
PINKEY	Return a DUKPT PIN encryption key.
PAN check digit compliance (one required if mode VFPE and the pan input output character set keyword is present. Otherwise, it is not allowed.)	
CMPCKDGT	Last digit of the PAN contains a compliant check digit per ISO/IEC 7812-1.
NONCKDGT	Last digit of the PAN does not contain a compliant check digit per ISO/IEC 7812-1.

clear_PAN_length

Direction	Type
Input	Integer

Specifies the number of digits in the *clear_PAN* parameter. This value must be 0 if *clear_PAN* is not to be enciphered. The value must be 15 through 19 if PAN data is presented for encryption.

clear_PAN

Direction	Type
Input	String

Contains the account number with which the PIN is associated. The full account number, including check digit, should be included. The data for this parameter is in binary format. It is the binary representation of 4-bit hex (keyword PAN4BITX) or ASCII (keyword PAN8BITA). If the PAN contains an odd number of 4-bit digits, the data must be left justified in the PAN variable and the right-most 4 bits are ignored.

If the *clear_PAN_length* parameter is zero, this parameter is ignored.

clear_CH_name_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *clear_CH_name* parameter. This value must be 0 if the cardholder name is not to be enciphered. The value must be 2 through 32 if the cardholder name data is presented for encryption.

clear_CH_name

Direction	Type
Input	String

Contains the card holder's full name. The data for this parameter is in binary format. It is the binary representation of ASCII characters as defined in Table 256 on page 616. Only characters defined in Table 256 on page 616 are valid.

clear_dtrack1_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *clear_dtrack1_data* parameter. This value must be 0 if the track 1 discretionary data is not to be enciphered. The value must be 1 through 56 if the track 1 discretionary data is presented for encryption.

clear_dtrack1_data

Direction	Type
Input	String

Contains the discretionary data that is stored on track 1 of a magnetic stripe card. This data does not include the PAN, cardholder name, expiration date, or service code. The data for this parameter is in binary format. It is the binary representation of ASCII characters as defined in Table 256 on page 616. Only characters defined in Table 256 on page 616 are valid.

clear_dtrack2_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *clear_dtrack2_data* parameter. This value must be 0 if the track 2 discretionary data is not to be enciphered. The value must be 1 through 19 if the track 2 discretionary data is presented for encryption.

clear_dtrack2_data

Direction	Type
Input	String

Contains the discretionary data that is stored on track 2 of a magnetic stripe card. This data does not include the PAN, expiration date, or service code. The data for this parameter is in binary format. It is the binary representation of ASCII characters. The data for this parameter is in BASE-16 binary format. It is the binary representation of ASCII in the range X'30' through X'39' and X'41' through X'46' (ASCII '0' through '9' and 'A' through 'F').

key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *key_identifier* parameter.

If the *key_identifier* parameter contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

key_identifier

Direction	Type
Input/Output	String

The identifier of the key that is used to either encrypt the card data (key management STATIC) or derive the *DUKPT_PIN_key_identifer* (key management DUKPT). The *key identifier* is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA key tokens

For DES-DUKPT, the key type must be KEYGENKY. In addition, it must have a control vector with bit 18 equal to B'1' (UKPT). The base derivation key is the one from which the operational keys are derived using the DUKPT algorithm defined in ANSI X9.24-1 2007.

For AES-DUKPT, this is an AES DKYGENKY with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1. The base derivation key is the one from which the operational keys are derived using the DUKPT algorithm defined in ANSI x9.24-3 2017.

For key management STATIC, (Zone Encryption Key in the Visa DSP specification), the key type must be either CIPHER or ENCIPHER. For production purposes, it is recommended that the key have left and right halves that are not equal.

Note: Data keys are not supported.

For X9.143 (TR-31) key blocks

For DES-DUKPT, the key usage is B0, algorithm is T, and mode of use is X.

For AES-DUKPT, the key usage is B0, algorithm is A, and mode of use is X.

For key management STATIC, the key usage is D0, algorithm is T, and mode of use is B or E.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

derivation_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *derivation_data* parameter. To specify the DES DUKPT method, set the value to 10 if the key management method DUKPT is specified in the rule array. To specify the AES DUKPT method, set the value to 20 if the key management method DUKPT is specified in the rule array. Otherwise, this value must be 0.

derivation_data

Direction	Type
Input	String

When specifying the DES-DUKPT method, *derivation_data* contains the 80 bit (10 byte) derivation data that is used as input to the DUKPT derivation process. The derivation data contains the current key serial number (CKSN), which is composed of the 59 bit initial key serial number value concatenated with the 21 bit value of the current encryption counter, which the device increments for each new transaction. This field is in binary format.

When specifying the AES-DUKPT method, *derivation_data* contains the 20 byte AES-DUKPT derivation data. See Table 684 on page 1719 for the layout of the AES-DUKPT derivation data. The algorithm indicator must be set to X'0000' (2- key TDES). The key usage indicator must be set to X'1000' (PIN Encryption).

enc_PAN_length

Direction	Type
Input/Output	Integer

Specifies the number of PAN digits in the *enc_PAN* parameter if the mode is VFPE. If the mode is CBC, the variable contains the number of bytes in the *enc_PAN* parameter. The output value is 0 or 15 through 19 for VFPE. For CBC mode, the output length is 16 if service completes successfully and PAN data was enciphered. Otherwise, it is 0.

enc_PAN

Direction	Type
Output	String

The field where the enciphered primary account number is returned. For VFPE mode, if the PAN contains an odd number of 4-bit digits, the data is left justified in the PAN variable and the right-most 4 bits can be ignored.

enc_CH_name_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *enc_CH_name* parameter. The output value is 0 or 2 through 32 for VFPE. For CBC mode, the output value is 16 through 40 and a multiple of 8 if the service is successful and cardholder name data is enciphered. The parameter can be larger on input. However, on output, this length is updated to indicate the actual number of bytes returned by the service.

enc_CH_name

Direction	Type
Output	String

The field where the enciphered cardholder full name is returned.

enc_dtrack1_data_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *enc_dtrack1_data* parameter. The output value is 0 or 1 through 56 for mode VFPE. For mode CBC, the output value is 16 through 64 and a multiple of 8 if the service is successful and the track 1 discretionary data is enciphered. The parameter can be larger on input. However, on output, this length is updated to indicate the actual number of bytes returned by the service.

enc_dtrack1_data

Direction	Type
Output	String

The field where the enciphered discretionary track 1 data is returned.

enc_dtrack2_data_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *enc_dtrack2_data* parameter. The output value is 0 or 1 through 19 for mode VFPE. For mode CBC, the output value is 8 or 16 if the service is successful and the data is enciphered. The parameter can be larger on input. However, on output, this length is updated to indicate the actual number of bytes returned by the service.

enc_dtrack2_data

Direction	Type
Output	String

The field where the enciphered discretionary track 2 data is returned.

DUKPT_PIN_key_identifier_length

Direction	Type
Input/Output	Integer

The length in bytes of the *DUKPT_PIN_key_identifier* parameter. When the NOPINKEY rule-array keyword is specified, the value is 0.

When PINKEY is specified:

- For CCA key tokens, the value is 64.
- For X9.143 key blocks, the maximum value is 9992. The value should be large enough to hold the derived key block.

On output, the variable is updated with the length of the data returned in the *DUKPT_PIN_key_identifier* variable.

DUKPT_PIN_key_identifier

Direction	Type
Input/Output	String

The derived PIN-encrypting key.

For CCA key tokens

On input, must contain a DES OPINENC or IPINENC skeleton token.

On output, *DUKPT_PIN_key_identifier* contains the DES token with the derived DES OPINENC or IPINENC key.

For X9.143 (TR-31) key blocks

On input, must contain a TDES PIN-encrypting key skeleton key block (key usage P0, algorithm T, mode of use B, D, or E)

On output, contains the key block with the derived PIN-encrypting key.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **FPE Encipher** access control point in the domain role controls the function of this service.

Specifying the DUKPT keyword requires that the **DUKPT - PIN Verify, PIN Translate** access control point be enabled in the domain role.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 343. FPE Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The AES-DUKPT algorithm requires the October 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 343. FPE Encipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	The AES-DUKPT algorithm requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

FPE Translate (CSNBFPET and CSNEFPET)

The FPE Translate callable service is used to translate payment data from encryption under one key to encryption under another key with a possibly different format. You should avoid having plain text payment data in your environment. Translations can be performed with data that has been encrypted using the standard encryption option or with data that has been encrypted using the VFPE option. However, the target translation uses double length static TDES keys and the standard encryption option.

This service can be used to translate one or all of the following fields: the primary account number (PAN), the cardholder name, the track 1 discretionary data, or the track 2 discretionary data.

The following translation options are supported:

1. Translate standard option with CBC mode TDES and DUKPT keys.
2. Translate VFPE option with VFPE mode TDES andDUKPT keys.
3. Translate standard option with CBC mode TDES and static TDES keys.

To use this service, you must specify the following:

- The processing method, which is limited to Visa Data Secure Platform (Visa DSP).
- The key management method, either STATIC or DUKPT.
- The algorithm, which is limited to TDES.
- The mode, either CBC or Visa Format Preserving Encryption (VFPE) for the inbound data.
- The ciphertext to be translated.
- The character set of each field to be translated using rule-array keywords.
- The base derivation key and either the key serial number for DES-DUKPT or the AES-DUKPT derivation data for AES-DUKPT, or a double-length TDES key if STATIC key management is used.
- The double length static TDES key used to re-encrypt the data.
- Optionally, a check digit compliance indicator if VFPE is specified.

The service returns the translated fields and optionally, the DUKPT PIN encryption key, if the DUKPT key management is selected and the PINKEY rule is specified.

The callable service name for AMODE(64) invocation is CSNEFPET.

Format

```
CALL CSNBFPET(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              input_PAN_length,
              input_PAN,
              input_CH_name_length,
              input_CH_name,
              input_dtrack1_data_length,
              input_dtrack1_data,
              input_dtrack2_data_length,
              input_dtrack2_data,
              input_key_identifier_length,
              input_key_identifier,
              output_key_identifier_length,
              output_key_identifier,
              derivation_data_length,
              derivation_data,
              output_PAN_length,
              output_PAN,
              output_CH_name_length,
              output_CH_name,
              output_dtrack1_data_length,
              output_dtrack1_data,
              output_dtrack2_data_length,
              output_dtrack2_data,
              DUKPT_PIN_key_identifier_length,
              DUKPT_PIN_key_identifier,
              reserved1_length,
              reserved1,
              reserved2_length,
              reserved2)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The minimum value is 5. The maximum value is 10.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Note: At least one character set keyword is required.

Table 344. Rule array keywords for FPE Translate

Keyword	Meaning
Processing method (required)	
VMDS	Specifies that the Visa DSP method (formally known as the Visa Merchant Data Secure method) is to be used for processing.
Key management method (one required)	
STATIC	Specifies the use of double length (2-key) triple-DES symmetric keys. This is a non-DUKPT key.
DUKPT	Specifies the use of the transaction unique general purpose Data Encryption Keys generated by the DUKPT process at the point of service for data encryption. This is required if VFPE mode is specified. Otherwise, this is optional. Both DES DUKPT and AES DUKPT key derivation methods are supported. The content of the *derivation_data* parameter will determine which DUKPT method is used.
Algorithm (required)	
TDES	Specifies the use of CBC mode triple-DES encryption.
Mode (one required)	
CBC	Specifies the use of CBC mode. This is the mode for the standard encryption option.
VFPE	Specifies the use of Visa format preserving encryption.
PAN input output character set (one required if the clear_PAN_length variable is greater than 0. Otherwise, it is not allowed.)	

Table 344. Rule array keywords for FPE Translate (continued)	
Keyword	**Meaning**
PAN8BITA	Specifies that the PAN data character set is ASCII represented in binary form. Valid only for VFPE mode.
PAN4BITX	Specifies that the PAN data character set is 4-bit hex. Two digits per byte. Valid only for VFPE mode.
PAN-EBLK	Specifies that the PAN data is in a CBC encrypted block. Valid only for CBC mode.
Cardholder name input output character set (required if the clear_CH_name_length variable is greater than 0.)	
CN8BITA	Specifies that the cardholder name character set is ASCII represented in binary format, one character per byte. See Table 256 on page 616 for valid characters. Valid only for VFPE mode.
CN-EBLK	Specifies that the cardholder name data is in a CBC-encrypted block.
Track_1 input character set (required if the clear_dtrack1_data_length variable is greater than 0. Otherwise, it is not valid.)	
TK18BITA	Specifies that the track 1 discretionary data character set is ASCII represented in binary format, one character per byte. See Table 256 on page 616 for valid characters.
TK1-EBLK	Specifies that the track 1 discretionary data is in a CBC-encrypted block. Valid only for CBC mode.
Track_2 input output character set (required if the clear_dtrack2_data_length variable is greater than 0. Otherwise, it is not valid.)	
TK28BITA	Specifies that the track 2 discretionary data character set is ASCII represented in binary format. Valid only for VFPE mode.
TK2-EBLK	Specifies that the track 2 discretionary data is in a CBC encrypted block. Valid only for CBC mode.
PIN encryption key output selection (one, optional, if DUKPT is specified. Otherwise, it is not valid.)	
NOPINKEY	Do not return a DUKPT PIN encryption key. This is the default.
PINKEY	Return a DUKPT PIN encryption key.
PAN check digit compliance (one required if mode VFPE and the PAN input character set keyword is present. Otherwise, it is not allowed.)	
CMPCKDGT	Last digit of the PAN contains a compliant check digit per ISO/IEC 7812-1.
NONCKDGT	Last digit of the PAN does not contain a compliant check digit per ISO/IEC 7812-1.

input_PAN_length

Direction	Type
Input	Integer

Specifies the number of bytes of data in the *input_PAN* parameter if the mode is CBC or the number of PAN digits if the mode is VFPE. The value is 0 if PAN data has not been presented for translation. Otherwise, the value is between 15 and 19 inclusive for VFPE. The value is 16 if CBC mode is selected.

input_PAN

Direction	Type
Input	String

The enciphered primary account number (PAN) that is to be translated. For VFPE mode, if the PAN contains an odd number of 4-bit digits, the data is left justified in the PAN variable and the right-most 4 bits are ignored.

input_CH_name_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_CH_name* parameter. This value must be 0 if cardholder name data is not presented for translation. Otherwise, the value is 2 through 32 for VFPE. For CBC mode, the input value is either 16, 24, 32, or 40.

input_CH_name

Direction	Type
Input	String

The enciphered cardholder full name that is to be translated. Only characters in Table 256 on page 616 are valid.

input_dtrack1_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_dtrack1_data* parameter. This value must be 0 if track 1 discretionary data is not presented for translation. Otherwise, the value is 1 through 56 for VFPE. For CBC mode, the input value is either 16, 24, 32, 40, 48, 56, or 64.

input_dtrack1_data

Direction	Type
Input	String

The encrypted track 1 data that is to be translated. Only characters in Table 256 on page 616 are valid.

input_dtrack2_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_dtrack2_data* parameter. This value must be 0 if track 2 discretionary data is not presented for translation. Otherwise, the value is 1 through 19 for VFPE. For CBC mode, the input value is either 8 or 16.

input_dtrack2_data

Direction	Type
Input	String

The encrypted track 2 data that is to be translated.

input_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_key_identifier* parameter.

If the *input_key_identifier* parameter contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

input_key_identifier

Direction	Type
Input/Output	String

The identifier of the key that is used to either decrypt the input card data (key management STATIC) or derive the *DUKPT_PIN_key_identifer* (key management DUKPT). The *key identifier* is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA key tokens

For DES-DUKPT, the key type must be KEYGENKY. In addition, it must have a control vector with bit 18 equal to B'1' (UKPT). The base derivation key is the one from which the operational keys are derived using the DUKPT algorithm defined in ANSI X9.24-1 2007.

For AES-DUKPT, this is an AES DKYGENKY with the A-DUKPT bit set to 1 in the low-order byte of key usage field 1. The base derivation key is the one from which the operational keys are derived using the DUKPT algorithm defined in ANSI x9.24-3 2017.

For key management STATIC, (Zone Encryption Key in the Visa DSP specification), the key type must be either CIPHER or DECIPHER. For production purposes, it is recommended that the key have left and right halves that are not equal.

Note: Data keys are not supported.

For X9.143 (TR-31) key blocks

For DES-DUKPT, the key usage is B0, algorithm is T, and mode of use is X.

For AES-DUKPT, the key usage is B0, algorithm is A, and mode of use is X.

For key management STATIC, the key usage is D0, algorithm is T, and mode of use is B or D.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

output_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *output_key_identifier* parameter.

If the *input_key_identifier* parameter contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

output_key_identifier

Direction	Type
Input/Output	String

The identifier of the key that is used to decrypt the output card data. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA key tokens, the key type must be either CIPHER or ENCIPHER. For production purposes, it is recommended that the key have left and right halves that are not equal.

Note: Data keys are not supported.

For X9.143 (TR-31) key blocks, the key usage is D0, algorithm is T, and mode of use is B or E.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

derivation_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *derivation_data* parameter. To specify the DES DUKPT method, set the value to 10 if the key management method DUKPT is specified in the rule array. To specify the AES DUKPT method, set the value to 20 if the key management method DUKPT is specified in the rule array. Otherwise, this value must be 0.

derivation_data

Direction	Type
Input	String

When specifying the DES-DUKPT method, *derivation_data* contains the 80 bit (10 byte) derivation data that is used as input to the DUKPT derivation process. The derivation data contains the current key serial number (CKSN), which is composed of the 59 bit initial key serial number value concatenated with the 21 bit value of the current encryption counter, which the device increments for each new transaction. This field is in binary format.

When specifying the AES-DUKPT method, *derivation_data* contains the 20 byte AES-DUKPT derivation data. See Table 684 on page 1719 for the layout of the AES-DUKPT derivation data. The algorithm indicator must be set to X'0000' (2- key TDES). The key usage indicator must be set to X'1000' (PIN Encryption).

output_PAN_length

Direction	Type
Input/Output	Integer

Specifies the number of bytes of data in the *output_PAN* parameter. This value is 0 or 16 on output.

output_PAN

Direction	Type
Output	String

The field where the translated primary account number with which the PIN is associated is returned. The full account number, including check digit, is translated. The data for this parameter is returned as TDES-encrypted data in binary format. The 16 byte output is left justified in this field.

output_CH_name_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *output_CH_name* parameter. This output value is either 0 or 16, 24, 32, or 40 bytes on output. The variable can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the card.

output_CH_name

Direction	Type
Output	String

The field where the translated cardholder full name is returned. The data for this parameter is returned as TDES-encrypted data in binary format.

output_dtrack1_data_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *output_dtrack1_data* parameter. The output value is either 0 or 16, 24, 32, 40, 48, 56, or 64 bytes. The value can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

output_dtrack1_data

Direction	Type
Output	String

The field where the translated discretionary track 1 data is returned. This is the discretionary data from track 1 of a magnetic stripe card. The data for this parameter is returned as TDES-encrypted data in binary format.

output_dtrack2_data_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *output_dtrack2_data* parameter. The output value is either 0, 8, or 16. The value can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

output_dtrack2_data

Direction	Type
Output	String

The field where the translated discretionary track 2 data is returned. This is the discretionary data from track 2 of a magnetic stripe card. The data for this parameter is returned as TDES-encrypted data in binary format.

DUKPT_PIN_key_identifier_length

Direction	Type
Input/Output	Integer

The length in bytes of the *DUKPT_PIN_key_identifier* parameter. When the NOPINKEY rule-array keyword is specified, the value is 0.

When PINKEY is specified:

- For CCA key tokens, the value is 64.

- For X9.143 key blocks, the maximum value is 9992. The value should be large enough to hold the derived key block.

On output, the variable is updated with the length of the data returned in the DUKPT_PIN_key_identifier variable.

DUKPT_PIN_key_identifier

Direction	Type
Input/Output	String

The derived PIN-encrypting key.

For CCA key tokens

On input, must contain a DES OPINENC or IPINENC skeleton token.

On output, *DUKPT_PIN_key_identifier* contains the DES token with the derived DES OPINENC or IPINENC key.

For X9.143 (TR-31) key blocks

On input, must contain a TDES PIN-encrypting key skeleton key block (key usage P0, algorithm T, mode of use B, D, or E)

On output, contains the key block with the derived PIN-encrypting key.

Note: The returned DES token will use the wrapping method indicated in the supplied skeleton token. The default wrapping setting is not used.

reserved1_length

Direction	Type
Input	Integer

Length in bytes of the *reserved1* parameter. The value must be 0.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input	String

This field is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **FPE Translate** access control point in the domain role controls the function of this service.

Specifying the DUKPT keyword requires that the **DUKPT - PIN Verify, PIN Translate** access control point be enabled in the domain role.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 345. FPE Translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The AES-DUKPT algorithm requires the October 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The AES-DUKPT algorithm is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	The AES-DUKPT algorithm requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

PIN Change/Unblock (CSNBPCU and CSNEPCU)

The PIN Change/Unblock callable service is used to generate a special PIN block to change the PIN accepted by an integrated circuit card (smartcard). The special PIN block is based on the new PIN and the card-specific diversified key and, optionally, on the current PIN of the smartcard. The new PIN block

is encrypted with a session key. The session key is derived in a two-step process. First, the card-specific diversified key (ICC Master Key) is derived according to the algorithm of the key:

- For TDES keys: Using the TDES-ENC algorithm of the Diversified Key Generate callable service.
- For AES keys: Using the MK-OPTC algorithm of the Diversified Key Generate2 callable service.

The session key is then generated according to the rule array algorithm:

- TDES-XOR - XOR ICC Master Key with the Application Transaction Counter (ATC).
- TDESEMV2 - Use the EMV2000 algorithm with a branch factor of 2.
- TDESEMV4 - Use the EMV2000 algorithm with a branch factor of 4.
- AES-EMV1 - Use the EMV Book 2 Common Session Key Derivation Option, which is also available as the SESS-ENC algorithm of the Diversified Key Generate2 callable service.

The TDES generating DKYGENKY cannot have replicated halves. The *encryption_issuer_master_key_identifier* is a DKYGENKY that permits generation of a SMPIN key. The *authentication_ issuer_master_key_identifier* is also a DKYGENKY that permits generation of a double length MAC key.

The AES generating key must have the type DKYGENKY. The *encryption_issuer_master_key_identifier* must permit generation of a PINPROT key (D-PPROT or D-ALL). For the EMV-PCU1 service, the *authentication_issuer_master_key_identifier* is not used.

The following output PIN block formats are specified by the VISA ICC Card specification: mutually exclusive rule array keywords, AMEXPCU1, AMEXPCU2, VISAPCU1 and VISAPCU2. They refer to whether the current PIN is used in the generation of the new PIN. Starting with CCA 7.2, the ISO-4 PIN block format is allowed for the *new_reference_PIN_block*, the *current_reference_PIN_block*, or both.

- VISAPCU1 would create a new PIN for a card without a PIN in an encrypted PIN-block in the *new_reference_PIN_block* variable. The contents of the five *current_reference_PIN_* variables are ignored.
- VISAPCU2 would provide the existing PIN for a card with a current PIN in an encrypted PIN-block in the *current_reference_PIN_block* variable and supply the new PIN value in an encrypted PIN block in the *new_reference_PIN_block* variable.
- AMEXPCU1 would create the output PIN from the new-reference PIN, the smart card-unique, intermediate key, and the current-reference PIN.
- AMEXPCU2 would create the output PIN from the new-reference PIN and the smart-card-unique, intermediate key.

The EMV-PCU1 output PIN block format specified in EMV v4.3 Book 2 Format 1 and detailed in the EMV Common Payment Application (2011) specification as updated by SB165. The new-reference PIN is passed in to the service as an encrypted PIN block. The contents of the five *current_reference_PIN_* variables are ignored. Note that both TDES-CBC and AES encryption of the output PIN block are supported.

An enhanced PIN security mode is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. To do this, you must enable the Enhanced PIN Security access control point in the domain role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.

The callable service name for AMODE(64) invocation is CSNEPCU.

Format

```
CALL CSNBPCU(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
```

```
            authentication_issuer_master_key_length,
            authentication_issuer_master_key_identifier,
            encryption_issuer_master_key_length,
            encryption_issuer_master_key_identifier,
            key_generation_data_length,
            key_generation_data,
            new_reference_PIN_key_length,
            new_reference_PIN_key_identifier,
            new_reference_PIN_block,
            new_reference_PIN_profile,
            new_reference_PIN_PAN_data,
            current_reference_PIN_key_length,
            current_reference_PIN_key_identifier,
            current_reference_PIN_block,
            current_reference_PIN_profile,
            current_reference_PIN_PAN_data,
            output_PIN_data_length,
            output_PIN_data,
            output_PIN_profile,
            output_PIN_message_length,
            output_PIN_message )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The valid values are 1 and 2.

rule_array

Direction	Type
Input	Character String

Keywords that provides control information to the callable service. The processing method is the algorithm used to create the generated key. The keywords are left-justified and padded on the right with blanks.

Table 346. Rule Array Keywords for PIN Change/Unblock

Keyword	Meaning
Algorithm (optional)	
AES-EMV1	The ICC Master Key is derived according to the CSNBDKG2 verb with option MK-OPTC, the EMV common session key is derived according to the CSNBDKG2 verb with option SESS-ENC. The *encryption_issuer_master_key_identifier* parameter must refer to an AES key. Only valid with the EMV-PCU1 PIN processing method.
TDES-XOR	TDES encipher clear data to generate the intermediate (card-unique) key, followed by XOR of the final 2 bytes of each key with the ATC counter. This is the default.
TDESEMV2	Same processing as in the diversified key generate service.
TDESEMV4	Same processing as in the diversified key generate service.
PIN processing method (required)	
VISAPCU1	Form the new PIN from the new reference PIN and the smart-card-unique, intermediate key.
VISAPCU2	Form the new PIN from the new reference PIN and the smart-card-unique, the intermediate (card-unique) key and the current reference PIN.
AMEXPCU1	Form the new PIN from the new reference PIN, the smart-card-unique, intermediate key, and the current reference PIN.
AMEXPCU2	Form the new PIN from the new reference PIN and the smart-card-unique, intermediate key.
EMV-PCU1	The new PIN is passed in the *new_reference_PIN_* set of parameters. The contents of the five *current_reference_PIN_* variables are ignored. The PIN block is formatted according to EMV v4.3 Book 2 Format 1 and detailed in EMV Common Payment Application (2011) as updated by SB165. TDES-CBC or AES encryption over the PIN block is done according to the issuer master key passed in and EMV v4.3 Book 2. **Note:** The *authentication_issuer_master_key_identifier* parameter is ignored. The EMV PIN-change command creation process calculates a MAC over the full message using a derived session key and recommends MAC chaining across EMV commands which is outside the scope of this service. Valid with the AES-EMV1, TDES-XOR, TDESEMV2, and TDESEMV4 algorithm keywords.

authentication_issuer_master_key_identifier_length

Direction	Type
Input	Integer

The length of the *authentication_issuer_master_key_identifier* parameter.

For PIN processing method EMV-PCU1, the value must be 0.

When *authentication_issuer_master_key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

authentication_issuer_master_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-derivation key to generate the card-unique authentication key. The key identifier is a variable-length operational key token or key block or the key label of an operational token or block in key storage.

For PIN processing method EMV-PCU1, the *authentication_issuer_master_key_identifier* must be a 64-byte NULL token.

For CCA keys, the identifier is a 64-byte DES key token of key type DKYGENKY. The control vector of this key must indicate subtype DKYL0 and permit the generation of a double-length MAC key (DMAC). This key may not have replicated key halves.

For X9.143 keys, the identifier is a variable-length key block of a TDES key-derivation key: key usage B3, algorithm T, and mode of use X. Optional block "DA" is required.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

encryption_issuer_master_key_length

Direction	Type
Input	Integer

The length of the *encryption_issuer_master_key_identifier* parameter.

When *encryption_issuer_master_key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

encryption_issuer_master_key_identifier

Direction	Type
Input/Output	String

The identifier of the key derivation key to generate the card-unique diversified key. The key identifier is a variable-length operational key token or key block or the key label of an operational token or block in key storage.

For CCA keys

For rule array keywords TDES-XOR, TDESEMV2, or TDESEMV4, the identifier is a 64-byte DES key token of key type DKYGENKY. The control vector of this key must indicate a subtype of DKYL0 and permit the generation of a double-length PIN encryption key (DMPIN). This key may not have replicated key halves.

For keyword AES-EMV1, the variable-length symmetric key token must have a token algorithm of AES and a key type of DKYGENKY, sequence level of DKYL0 or DKYL1, and key type to diversify of D-PPROT or D-ALL. For an AES D-PPROT key, the key usage fields must indicate that the derived key can be used for encryption (ENCRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN services control must include PINXLATE or REFORMAT.

For X9.143 (TR-31) keys

For rule array keywords TDES-XOR, TDESEMV2, or TDESEMV4, this a variable-length TDES key block. Key usage B3, algorithm T, and mode of use X. Optional block "DA" is required.

For keyword AES-EMV1, this a variable-length AES key block. Key usage B3, algorithm A, and mode of use X. Optional block "DA" is required.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

key_generation_data_length

Direction	Type
Input	Integer

The length of the *key_generation_data* parameter. For TDES-XOR, TDESEMV2, or TDESEMV4, this value must be 10, 18, 26, or 34 bytes. For AES-EMV1, this value must be 32.

key_generation_data

Direction	Type
Input/Output	String

The data provided to generate the card-unique session key.

TDES key generation:
For TDES-XOR, this consists of 8 or 16 bytes of data to be processed by TDES to generate the card-unique diversified key followed by a 16 bit ATC counter to offset the card-unique diversified key to form the session key. For TDESEMV2 and TDESEMV4, this may be 10, 18, 26 or 34 bytes. See “Diversified Key Generate (CSNBDKG and CSNEDKG)” on page 151 for more information.

AES key generation:
For AES-EMV1, this parameter consists of 2 sections of 16 bytes each, holding first the derivation data for the CSNBDKG2 service MK-OPTC and second the derivation data for the CSNBDKG2 service SESS-ENC. The MK-OPTC derivation data is specified in EMV v4.3 Book 2 section A1.4.3 'Option C' (PAN and PAN sequence number, padded). The SESS-ENC derivation data should be as specified in EMV v4.3 Book 2 section A1.3.1 'Common Session Key Derivation Option' (Application Cryptogram, padded).

```
Derivation Data :: [ 16-byte MK-OPTC data || 16-byte SESS-ENC data ]
```

new_reference_PIN_key_length

Direction	Type
Input	Integer

The length of the *new_reference_PIN_key_identifier* parameter.

When *new_reference_PIN_key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

new_reference_PIN_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to unwrap the *new_reference_PIN_block*. The key identifier is a variable-length operational key token or key block or the key label of an operational token or block in key storage.

When the label name is supplied, the name must be unique in the CKDS. The key may be a DES key (all *new_reference_PIN_profile* PIN block formats except ISO-4) or an AES key (*new_reference_PIN_profile PIN* block format ISO-4).

For CCA keys

For all block formats except ISO-4, the identifier is a 64-byte DES key token of key type IPINENC or OPINENC.

For block format ISO-4, the identifier is a variable-length symmetric AES key-token of key type PINPROT. In addition, the key usage fields must indicate that the key can be used for encryption (ENCRYPT) or decryption (DECRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN services control must include PINXLATE.

For X9.143 (TR-31) keys

For all block formats except ISO-4, the identifier is a variable-length key block of a TDES PIN-encrypting key: key usage P0, algorithm T, and mode of use E or D.

For block format ISO-4, the identifier is a variable-length key block of an AES PIN-encrypting key: key usage P0, algorithm A, and mode of use E or D.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

new_reference_PIN_block

Direction	Type
Input	String

This field contains the enciphered PIN block of the new PIN. When the *new_reference_PIN_profile* specifies ISO-4, the *new_reference_PIN_block* will be 16 bytes long. For all other formats, the *new_reference_PIN_block* will be 8 bytes long.

new_reference_PIN_profile

Direction	Type
Input	String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword, a format control keyword (NONE) and a pad digit as required by certain formats.

new_reference_PIN_PAN_data

Direction	Type
Input	String

A primary account number (PAN) in character format. The service uses this parameter if the PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the *input_PIN_profile* PIN block format. Otherwise, ensure that this parameter is a 12-byte value in application storage. The information in this parameter will be ignored, but the parameter must be specified.

When using the ISO-0, ISO-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 – 19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Table 347. PAN data structure

Offset	Length	Description
0	2	Length of the PAN data field, p.
2	p	10 to 19 bytes of PAN data.
2+p	0-9	Padding.

current_reference_PIN_key_length

Direction	Type
Input	Integer

The length of the *current_reference_PIN_key_identifier* parameter.

When *new_reference_PIN_key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

current_reference_PIN_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to unwrap the *current_reference_PIN_block*. The key identifier is a variable-length operational key token or key block or the key label of an operational token or block in key storage.

When the label name is supplied, the name must be unique on the CKDS. The key may be a DES key (all *current_reference_PIN_profile* PIN block formats except ISO-4) or an AES key (*current_reference_PIN_profile* PIN block format ISO-4).

For CCA keys

For all block formats except ISO-4, the identifier is a 64-byte key token of a DES PIN-encryption key of key type IPINENC or OPINENC.

For block format ISO-4, the identifier is a variable-length AES symmetric key-token of key type PINPROT. In addition, the key usage fields must indicate that the key can be used for encryption (ENCRYPT) or decryption (DECRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN services control must include PINXLATE.

For X9.143 (TR-31) keys

For all block formats except ISO-4, the identifier is a variable-length key block of a TDES PIN-encrypting key. Key usage P0, algorithm T, and mode of use E or D.

For block format ISO-4, the identifier is a variable-length key block of an TDES PIN-encrypting key. Key usage P0, algorithm T, and mode of use E or D.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

current_reference_PIN_block

Direction	Type
Input	String

This field contains the enciphered PIN block of the new PIN. When the *current_reference_PIN_profile* specifies ISO-4, the *current_reference_PIN_block* will be 16 bytes long. For all other formats, the *current_reference_PIN_block* will be 8 bytes long.

current_reference_PIN_profile

Direction	Type
Input	String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword, a format control keyword (NONE) and a pad digit as required by certain formats. If the *rule_array* contains VISAPCU1 or AMEXPCU2, this value is ignored.

current_reference_PIN_PAN_data

Direction	Type
Input	String

A primary account number (PAN) in character format. The service uses this parameter if the PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the *input_PIN_profile* PIN block format. Otherwise, ensure that this parameter is a 12-byte value in application storage. The information in this parameter will be ignored, but the parameter must be specified.

When using the ISO-0, ISO-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 – 19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Table 348. PAN data structure

Offset	Length	Description
0	2	Length of the PAN data field, p.
2	p	10 to 19 bytes of PAN data.
2+p	0-9	Padding.

output_PIN_data_length

Direction	Type
Input	Integer

The value of this parameter should be 0.

output_PIN_data

Direction	Type
Input	String

This field is reserved.

output_PIN_profile

Direction	Type
Input	String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword (VISAPCU1, VISAPCU2, AMEXPCU1, AMEXPCU2, or EMV-PCU1), a format control keyword, NONE, (left aligned and padded on the right with space characters) and 8 space characters.

output_PIN_message_length

Direction	Type
Input/Output	Integer

The length of the *output_PIN_message* field. The value must be at least 16 for EMV-PCU1, VISAPCU1, and VISAPCU2 and at least 8 for AMEXPCU1 and AMEXPCU2.

output_PIN_message

Direction	Type
Output	String

The reformatted PIN block with the new reference PIN enciphered under the SMPIN session key.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Triple-length DES keys requires the CCA release 6.7, 7.4, or later licensed internal code.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 349. Required access control points for PIN Change/Unblock

PIN-block encrypting key-type	Access control point
OPINENC	PIN Change/Unblock - change EMV PIN with OPINENC
IPINENC	PIN Change/Unblock - change EMV PIN with IPINENC

When the *authentication_key_identifier* or *encryption_key_identifier* is specified with control vector bits (19 – 22) of B'1111', the **Diversified Key Generate - DKYGENKY – DALL** access control point must also be enabled.

An enhanced PIN security mode is available for extracting PINs from a 3621 or 3624 encrypted PIN-block and formatting an encrypted PIN block into IBM 3621 or 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** access control.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *new_reference_PIN_profile*, *current_reference_PIN_profile*, and *output_PIN_profile* parameters is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 350. PIN Change/Unblock hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. Triple-length DES keys are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. Triple-length DES keys are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require the July 2019 or later licensed internal code (LIC). PIN block format ISO-4 requires the October 2020 or later licensed internal code (LIC). Triple-length DES keys require the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. Triple-length DES keys are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	PIN block format ISO-4 requires the September 2020 or later licensed internal code (LIC). Triple-length DES keys require the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Triple-length DES keys require the CCA release 7.4 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 350. PIN Change/Unblock hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Recover PIN from Offset (CSNBPFO and CSNEPFO)

Use the Recover PIN from Offset callable service to calculate the encrypted customer-entered PIN from a PIN generating key, account information, and an IBM-PIN0 Offset. The customer-entered PIN will be returned in a PIN block formatted to the specifications of the *PIN_profile* and *PAN_data*, and encrypted with the key supplied in the *PIN_encryption_key_identifier*.

The callable service name for AMODE(64) invocation is CSNEPFO.

Format

```
CALL CSNBPFO(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             PIN_encryption_key_identifier_length,
             PIN_encryption_key_identifier,
             PIN_generation_key_identifier_length,
             PIN_generation_key_identifier,
             PIN_profile,
             PAN_data,
             offset,
             reserved_1,
             data_array,
             encrypted_PIN_block_length,
             encrypted_PIN_block )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

This parameter is ignored by ICSF.

PIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Length of the *PIN_encryption_key_identifier* field in bytes.

When *PIN_encryption_key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

PIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to encrypt the PIN block. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

The key may be a DES key (all PIN block formats except ISO-4) or an AES key (PIN block format ISO-4).

For CCA keys

For all block formats except ISO-4, the identifier is a 64-byte DES key token of a PIN-encryption key. The control vector must specify an OPINENC key type.

For block format ISO-4, the identifier is a variable-length symmetric AES key-token of key type PINPROT. The key usage fields must indicate that the key can be used for encryption (ENCRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN services control must include CPINENC.

For X9.143 (TR-31) keys

For all block formats except ISO-4, the identifier is a variable-length key block of a TDES PIN-encrypting key: key usage P0, algorithm T, and mode of use E.

For block format ISO-4, the identifier is a variable-length key block of an AES PIN-encrypting key: key usage P0, algorithm A, and mode of use E.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

PIN_generation_key_identifier_length

Direction	Type
Input	Integer

Length of the *PIN_generation_key_identifier* field in bytes.

When *PIN_generation_key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

PIN_generation_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-generation key to generate the bank reference PIN. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte DES key token of key type PINGEN with key usage attribute CPINGENA enabled in the control vector.

For X9.143 (TR-31) keys, the identifier is a variable-length key block containing a TDES PIN generation key: key usage V1, algorithm T, and mode of use C or G.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key

PIN_profile

Direction	Type
Input	String

The parameter consists of three 8-byte character elements that contain information necessary to format a PIN block. The pad digit is needed to format an IBM 3624 or 3621 PIN block. The format control constant must be "NONE ". The first element of the PIN_profile (PIN Block Format) determines the format of the output PIN block.

PAN_data

Direction	Type
Input	String

A primary account number (PAN) in character format. The service uses this parameter if the PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 12-byte value in application storage. The information in this parameter will be ignored, but the parameter must be specified.

When using the ISO-0, ISO-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 – 19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Table 351. PAN data structure

Offset	Length	Description
0	2	Length of the PAN data field, p.
2	p	10 to 19 bytes of PAN data.
2+p	0-9	Padding.

offset

Direction	Type
Input	String

A 16 byte area that contains the 4-byte PVV left-justified and padded with blanks. This is the value which was returned by a prior call to the Clear PIN Generate Alternate callable service.

data_array

Direction	Type
Input	String

The *data_array* parameter is a pointer to a string variable containing three 16-byte numeric character strings, which are equivalent to a single 48-byte string. The values in the data array depend on the keyword for the PIN-calculation method. Each element is not always used, but you must always declare a complete data array.

Array Element	Description
decimalization_table	This element contains the decimalization table of 16 characters (0-9) that are used to convert the hexadecimal digits '0'x to 'F'x of the enciphered validation data to the decimal digits '0'x to '9'x
validation_data	This element contains 1 to 16 characters of account data, left justified and padded on the right with spaces.
reserved_field	Must be 16 bytes of blanks

reserved_1

Direction	Type
Input	Integer

The *reserved_1* parameter must be zero.

encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

The length of the *encrypted_PIN_block* parameter in bytes. When the PIN block format is ISO-4, this must be 16. For all other formats, this must be 8.

encrypted_PIN_block

Direction	Type
Output	String

This parameter contains the encrypted customer PIN that was originally used in the Clear PIN Generate Alternate service. When the PIN block format is ISO-4, the PIN block will be 16 bytes long. For all other formats, the PIN block will be 8 bytes long.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control point

The **Recover PIN From Offset** access control point controls the function of this service.

When the **ANSI X9.8 PIN - Enforce PIN block restrictions** control is enabled in the domain role, the PIN block format in the *input_PIN_profile* parameter must be ISO-0 or ISO-3.

If the ANSI X9.8 PIN – Use stored decimalization tables only access control point is enabled in the domain role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

An enhanced PIN security mode is available for extracting PINs from a 3621 or 3624 encrypted PIN-block and formatting an encrypted PIN block into IBM 3621 or 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** access control.

When the **Disallow translation from DES wrapping to weaker DES wrapping** access control point is enabled in the domain role, this service will fail if the *PIN_encryption_key_identifier* is stronger than the *PIN_generation_key_identifier*.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *PIN_profile* parameter is not allowed to be ISO-1.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 352. Recover PIN from Offset required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. X9.143 key blocks are not supported.

Table 352. Recover PIN from Offset required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require the July 2019 or later licensed internal code (LIC). PIN block format ISO-4 requires the October 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	PIN block format ISO-4 requires the September 2020 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Secure Messaging for Keys (CSNBSKY and CSNESKY)

The Secure Messaging for Keys callable service will encrypt a text block including a clear key value decrypted from an internal or external DES token. The text block is normally a "Value" field of a secure message TLV (Tag/Length/Value) element of a secure message. TLV is defined in ISO/IEC 7816-4.

The callable service name for AMODE(64) invocation is CSNESKY.

Format

```
CALL CSNBSKY(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             input_key_identifier,
             key_encrypting_key_identifier,
```

```
                secmsg_key_identifier,
                text_length,
                clear_text,
                initialization_vector,
                key_offset,
                key_offset_field_length,
                enciphered_text,
                output_chaining_vector )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The valid values are 0 and 1.

rule_array

Direction	Type
Input	Character String

Keywords that provides control information to the callable service. The processing method is the encryption mode used to encrypt the message.

Table 353. Rule Array Keywords for Secure Messaging for Keys	
Keyword	**Meaning**
Enciphering mode (optional)	
TDES-CBC	Use CBC mode to encipher the message (default).
TDES-ECB	Use ECB mode to encipher the message.

input_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to be recovered in the clear and placed in the text to be encrypted. The key identifier is a variable-length internal or external key token or key block or the label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte key token containing a double-length DES key. The control vector of the key must not prohibit export.

For X9.143 keys, the identifier is a variable-length key block of a TDES key with an exportability of E or S.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

key_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to unwrap the input key. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage. If a key label is specified, the key label must be unique.

When the key supplied in the *input_key_identifier* parameter is an internal token or block, this is a 64-byte null token.

When the key supplied in the *input_key_identifier* parameter is an external token or block:

- For CCA keys, the identifier is a 64-byte DES key token of key type IMPORTER or EXPORTER.
- For X9.143 key, the identifier is a variable-length key block of a TDES key-encrypting key: algorithm T, and mode of use D or E. When the *input_key_identifier* is a CCA key, the key usage must be K0; when it is an X9.143 key, the key usage must be K1.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

secmsg_key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to encrypt the update clear text. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA keys, the identifier is a 64-byte DES key token of key type SECMSG with key usage attribute SMKEY enabled in the control vector

For X9.143 keys, the identifier is a variable-length key block of a DES secure-messaging key: key usage K0, algorithm T, and mode of use E.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

text_length

Direction	Type
Input	Integer

The length of the *clear_text* parameter that follows. Length must be a multiple of eight. Maximum length is 4K.

clear_text

Direction	Type
Input	String

Clear text that contains the recovered DES key at the offset specified and is then encrypted. Any padding or formatting of the message must be done by the caller on input.

initialization_vector

Direction	Type
Input	String

The 8-byte supplied string for the TDES-CBC mode of encryption. The *initialization_vector* is XORed with the first 8 bytes of *clear_text* prior to encryption. This field is ignored for TDES-ECB mode.

key_offset

Direction	Type
Input	Integer

The offset within the *clear_text* parameter at *key_offset* where the recovered clear *input_key_identifier* value is to be placed. The first byte of the *clear_text* field is offset 0.

key_offset_field_length

Direction	Type
Input	Integer

The length of the field within *clear_text* parameter at *key_offset* where the recovered clear *input_key_identifier* value is to be placed. Length must be a multiple of eight and is equal to the key length of the recovered key. The key must fit entirely within the *clear_text*.

enciphered_text

Direction	Type
Output	String

The field where the enciphered text is returned. The length of this field must be at least as long as the *clear_text* field.

output_chaining_vector

Direction	Type
Output	String

This field contains the last 8 bytes of enciphered text and is used as the *initialization_vector* for the next encryption call if data needs to be chained for TDES-CBC mode. No data is returned for TDES-ECB.

Restrictions

Triple-length DES keys are not supported.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the Secure Messaging for Keys service and any key labels specified as input.

Keys only appear in the clear within the secure boundary of the cryptographic coprocessor and never in host storage.

Access control point

The **Secure Messaging for Keys** access control point controls the function of this service.

When the **Disallow translation from DES wrapping to weaker DES wrapping** access control point is enabled in the domain role, this service will fail if the *input_key_identifier* is encrypted under the *key_encrypting_key_identifier* and the *key_encrypting_key_identifier* is weaker than the *secmsg_key_identifier*.

When *secmsg_key_identifier* is a TR-31 key, access control point **SKY – Allow K0 for secmsg key identifier** is required.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 354. Secure Messaging for Keys required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	X9.143 key blocks are not supported.

Table 354. Secure Messaging for Keys required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Secure Messaging for PINs (CSNBSPN and CSNESPN)

The Secure Messaging for PINs callable service will encrypt a text block including a clear PIN block recovered from an encrypted PIN block. The input PIN block will be reformatted if the block format in the *input_PIN_profile* is different than the block format n the *output_PIN_profile*. The clear PIN block will only be self encrypted if the SELFENC keyword is specified in the *rule_array*. The text block is normally a 'Value' field of a secure message TLV (Tag/Length/Value) element of a secure message. TLV is defined in ISO/IEC 7816-4.

An enhanced PIN security mode on the CEX3C and later is available to implement restrictions required by the ANSI X9.8 PIN standard. To enforce these restrictions, you must enable the following control points in the domain role.

- ANSI X9.8 PIN - Enforce PIN block restrictions
- ANSI X9.8 PIN - Allow modification of PAN
- ANSI X9.8 PIN - Allow only ANSI PIN blocks

The callable service name for AMODE(64) invocation is CSNESPN.

Format

```
CALL CSNBSPN(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              input_PIN_block,
              PIN_encrypting_key_identifier,
              input_PIN_profile,
              input_PAN_data,
              secmsg_key_identifier,
              output_PIN_profile,
              output_PAN_data,
              text_length,
              clear_text,
              initialization_vector,
              PIN_offset,
              PIN_offset_field_length,
              enciphered_text,
              output_chaining_vector )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The valid values are 0, 1, or 2.

rule_array

Direction	Type
Input	Character String

Keywords that provide control information to the callable service. The processing method is the algorithm used to create the generated key. The keywords are left justified and padded on the right with blanks.

Table 355. Rule Array Keywords for Secure Messaging for PINs

Keyword	Meaning
Token type (one required).	
AES-CBC	Use CBC mode to encipher the message. The *secmsg_key_identifier* parameter must refer to an AES key.

Table 355. Rule Array Keywords for Secure Messaging for PINs (continued)	
Keyword	**Meaning**
AES-ECB	Use ECB mode to encipher the message. The *secmsg_key_identifier* parameter must refer to an AES key.
TDES-CBC	Use CBC mode to encipher the message (default).
TDES-ECB	Use ECB mode to encipher the message.
PIN encryption (optional).	
CLEARPIN	Recovered clear input PIN block (may be reformatted) is placed in the clear in the message for encryption with the secure message key (default).
SELFENC	Recovered clear input PIN block (may be reformatted) is self-encrypted and then placed in the message for encryption with the secure message key.

input_PIN_block

Direction	Type
Input	String

The input PIN block that is to be recovered in the clear and perhaps reformatted, and then placed in the *clear_text* to be encrypted.

When the *input_PIN_profile* specifies ISO-4, the *input_PIN_block* will be 16 bytes long. For all other formats, the *input_PIN_block* will be 8 bytes long.

PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN-encrypting key to unwrap the input PIN block. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage.

The key may be a DES key (all input_PIN_profile PIN block formats except ISO-4) or an AES key (input_PIN_profile PIN block format ISO-4).

For CCA keys

For all block formats except ISO-4, the identifier is a 64-byte key token of a DES PIN-encrypting key. The control vector must specify an IPINENC key type.

For block format ISO-4, the identifier is a variable-length symmetric key-token of an AES PIN-encrypting key of key type PINPROT. The key usage fields must indicate that the key can be used for encryption (ENCRYPT) or decryption (DECRYPT), the encryption mode must be Cipher Block Chaining (CBC), common usage control must be NOFLDFMT, PIN block format usage must be ISO-4, and PIN services control must include REFORMAT.

For X9.143 (TR-31) keys

For all block formats except ISO-4, the identifier is a variable-length key block of a DES PIN-encrypting key: key usage P0, algorithm T, and mode of use D.

For block format ISO-4, the identifier is a variable-length key block of an AES PIN-encrypting key: key usage P0, algorithm A, and mode of use D.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

input_PIN_profile

Direction	Type
Input	Character String

The three 8-byte character elements that contain information necessary to extract the PIN from a formatted PIN block. The valid input PIN formats are ISO-0, ISO-1, ISO-2, ISO-3, and ISO-4. See "The PIN profile" on page 609 for additional information.

input_PAN_data

Direction	Type
Input	String

A primary account number (PAN) in character format. The service uses this parameter if the PIN profile specifies the ISO-0, ISO-3, ISO-4, or VISA-4 keyword for the *input_PIN_profile* PIN block format. Otherwise, ensure that this parameter is a 12-byte value in application storage. The information in this parameter will be ignored, but the parameter must be specified.

When using the ISO-0, ISO-3, or VISA-4 keyword, the value is 12 bytes long. Use the 12 rightmost digits of the PAN data, excluding the check digit.

When using the ISO-4 keyword, the value is 21 bytes long. The PAN data is 10 – 19 bytes long. The length of the PAN data and the PAN data are contained in the structure below padded to 21 bytes with characters that will be ignored.

Table 356. PAN data structure

Offset	Length	Description
0	2	Length of the PAN data field, p.
2	p	10 to 19 bytes of PAN data.
2+p	0-9	Padding.

secmsg_key_identifier

Direction	Type
Input/Output	String

The identifier of the data-encrypting key to encrypt the update clear text. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage. If a key label is specified, the key label must be unique.

The key may be a DES key or an AES key.

For CCA keys

- The identifier a 64-byte DES key token of key type SECMSG with key usage attribute SMPIN enabled in the control vector.
- The identifier is a variable-length AES symmetric key-token of key type SECMSG with key usage attributes secure message encryption enablement must be SMPIN and verb restriction control must be ANY-USE.

For X9.143 (TR-31) keys

- The identifier is a variable-length key block of a DES PINENC key: key usage P0, algorithm T, and mode of use E.

- The identifier is a variable-length key block of an AES PINENC key: key usage P0, algorithm A, and mode of use E.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

output_PIN_profile

Direction	Type
Input	String

The three 8-byte character elements that contain information necessary to create a formatted PIN block. If reformatting is not required, the *input_PIN_profile* and the *output_PIN_profile* must specify the same PIN block format. Output PIN block formats supported are ISO-0, ISO-1, ISO-2 and ISO-3.

output_PAN_data

Direction	Type
Input	String

The 12-digit personal account number (PAN) if the output PIN format is ISO-0 only. Otherwise, this parameter is ignored.

text_length

Direction	Type
Input	Integer

The length of the *clear_text* parameter that follows. Length must be a multiple of eight. Maximum length is 4K.

clear_text

Direction	Type
Input	String

Clear text that contains the recovered and/or reformatted/encrypted PIN at offset specified and then encrypted. Any padding or formatting of the message must be done by the caller on input.

initialization_vector

Direction	Type
Input	String

The supplied string for the TDES-CBC or AES-CBC mode of encryption. For TDES-CBC mode, the *initialization_vector* must be 8 bytes. For AES-CBC mode, the *initialization_vector* must be 16 bytes.

The *initialization_vector* is XORed with the first block of *clear_text* prior to encryption. This field is ignored for TDES-ECB and AES-ECB mode.

PIN_offset

Direction	Type
Input	Integer

The offset within the *clear_text* parameter where the reformatted PIN block is to be placed. The first byte of the *clear_text* field is offset 0.

PIN_offset_field_length

Direction	Type
Input	Integer

The length of the field within *clear_text* parameter at *PIN_offset* where the recovered clear *input_PIN_block* value is to be placed. The PIN block may be self-encrypted if requested by the rule array. Length must be eight. The PIN block must fit entirely within the *clear_text*.

enciphered_text

Direction	Type
Output	String

The field where the enciphered text is returned. The length of this field must be at least as long as the *clear_text* field.

output_chaining_vector

Direction	Type
Output	String

This field contains the last 8 or 16 bytes of enciphered text and is used as the *initialization_vector* for the next encryption call if data needs to be chained.

For TDES-CBC mode, 8 bytes are returned. For AES-CBC mode, 16 bytes are returned. No data is returned for TDES-ECB or AES-ECB.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the Secure Messaging for PINs service and any key labels specified as input.

Keys only appear in the clear within the secure boundary of the cryptographic coprocessors and never in host storage.

Triple-length DES keys requires the CCA release 6.7, 7.4, or later licensed internal code.

Access control point

The **Secure Messaging for PINs** access control point controls the function of this service.

Three additional access controls should be considered: **ANSI X9.8 PIN - Enforce PIN block restrictions, ANSI X9.8 PIN - Allow modification of PAN,** and **ANSI X9.8 PIN - Allow only ANSI PIN blocks**. These three access controls affect how PIN processing is performed as described below. The access controls will effect this and other PIN processing services if enabled.

1. Enable the **ANSI X9.8 PIN - Enforce PIN block restrictions** access control to apply additional restrictions to PIN processing as follows:
   - Constrain use of ISO-2 PIN blocks to offline PIN verification and PIN change operations in integrated circuit card environments only. Specifically, do not allow ISO-2 input or output PIN blocks.
   - Do not translate or reformat a PIN-block format that includes a PAN into a PIN-block format that does not include a PAN. Specifically, do not allow an ISO-1 PIN-block format in the *output_PIN_profile* variable when the PIN-block format in the *input_PIN_profile* variable is ISO-0, ISO-3, or ISO-4.

- Do not allow a change of PAN data. Specifically, when performing translations between PIN block formats that both include PAN data, do not allow the *input_PAN_data* and *output_PAN_data* variables to be different from the PAN data enciphered in the input PIN-block.

2. Enable the **ANSI X9.8 PIN - Allow modification of PAN** access control to override the restriction to not allow a change of PAN data. This override is applicable only when either the **ANSI X9.8 PIN - Enforce PIN block restrictions** control the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control, or both are enabled. This override is to support account number changes in issuing environments. The **ANSI X9.8 PIN - Allow modification of PAN** control has no affect if neither the **ANSI X9.8 PIN - Enforce PIN block restrictions** control nor the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control is enabled. This rule does not apply for CSNBPTRE, and PAN changes are not allowed.
3. Enable the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control to apply a more restrictive variation of the **ANSI X9.8 PIN - Enforce PIN block restrictions** control. In addition to the previously described restrictions of the **ANSI X9.8 PIN - Enforce PIN block restrictions** control, this control also restricts the *input_PIN_profile* and the *output_PIN_profile* to contain only ISO-0, ISO-1, ISO-3, and ISO-4 PIN block formats. Specifically, the IBM 3624 PIN-block format is not allowed with this command. The **ANSI X9.8 PIN - Allow only ANSI PIN blocks** control overrides the **ANSI X9.8 PIN - Enforce PIN block restrictions** control.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format in the *input_PIN_profile* and *output_PIN_profile* parameters is not allowed to be ISO-1.

When *secmsg_key_identifier* is a TR-31 key, access control point **SPN - Allow K0 for secmsg key identifier** is required.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 357. Secure Messaging for PINs required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. Triple-length DES keys are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. Triple-length DES keys are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require the July 2019 or later licensed internal code (LIC). PIN block format ISO-4 requires the October 2020 or later licensed internal code (LIC). Triple-length DES keys require the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 357. Secure Messaging for PINs required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. PIN block format ISO-4 is not supported. Triple-length DES keys are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	PIN block format ISO-4 requires the September 2020 or later licensed internal code (LIC). Triple-length DES keys require the CCA release 6.7 or later licensed internal code (LIC). X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Triple-length DES keys require the CCA release 7.4 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

SET Block Compose (CSNDSBC and CSNFSBC)

The SET Block Compose callable service performs encryption of the Optimal Asymmetric Encryption Padding (OAEP) Block by using one of the following:

- DES-encryption.
- RSA-encryption with optional OAEP-formatting by using a series of SHA-1 hashing operations.

The callable service name for AMODE(64) invocation is CSNFSBC.

Format

```
CALL CSNDSBC(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             block_contents_identifier,
             XData_string_length,
             XData_string,
             data_to_encrypt_length,
             data_to_encrypt,
             data_to_hash_length,
             data_to_hash,
             initialization_vector,
             RSA_public_key_identifier_length,
             RSA_public_key_identifier,
             DES_key_block_length,
             DES_key_block,
             RSA_OAEP_block_length,
             RSA_OAEP_block,
```

```
                chaining_vector,
                DES_encrypted_data_block )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 1 or 2.

rule_array

Direction	Type
Input	Character String

Keywords that provides control information to the callable service. The keyword must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks.

Table 358. Keywords for SET Block Compose Control Information

Keyword	Meaning
Block Type (required)	

Table 358. Keywords for SET Block Compose Control Information (continued)	
Keyword	**Meaning**
SET1.00	The structure of the RSA-OAEP encrypted block is defined by SET protocol.
Formatting Information (optional)	
DES-ONLY	DES encryption only is to be performed; no RSA-OAEP formatting will be performed. (See Usage Notes.)

block_contents_identifier

Direction	Type
Input	String

A one-byte string, containing a binary value that will be copied into the Block Contents (BC) field of the SET DB data block (indicates what data is carried in the Actual Data Block, ADB, and the format of any extra data (*XData_string*)). This parameter is ignored if DES-ONLY is specified in the rule-array.

XData_string_length

Direction	Type
Input	Integer

The length in bytes of the data contained within *XData_string*. The maximum length is 94 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

XData_string

Direction	Type
Input	String

Extra-encrypted data contained within the OAEP-processed and RSA-encrypted block. The format is indicated by *block_contents_identifier*. For a *XData_string_length* value of zero, *XData_string* must still be specified, but will be ignored by ICSF. The string is treated as a string of hexadecimal digits. This parameter is ignored if DES-ONLY is specified in the rule-array.

data_to_encrypt_length

Direction	Type
Input/Output	Integer

The length in bytes of data that is to be DES-encrypted. The length has a maximum value of 32 MB minus 8 bytes to allow for up to 8 bytes of padding. The data is identified in the *data_to_encrypt* parameter. On output, this value is updated with the length of the encrypted data in the *DES_encrypted_data_block*.

data_to_encrypt

Direction	Type
Input	String

The data that is to be DES-encrypted (with a 64-bit DES key generated by this service). The data will be padded by this service according to the PKCS #5 padding rules.

data_to_hash_length

Direction	Type
Input	Integer

The length in bytes of the data to be hashed. The hash is an optional part of the OAEP block. If the *data_to_hash_length* is 0, no hash will be included in the OAEP block. This parameter is ignored if DES-ONLY is specified in the *rule_array* parameter.

data_to_hash

Direction	Type
Input	String

The data that is to be hashed and included in the OAEP block. No hash is computed or inserted in the OAEP block if the *data_to_hash_length* is 0. This parameter is ignored if DES-ONLY is specified in the *rule_array* parameter.

initialization_vector

Direction	Type
Input	String

An 8-byte string containing the initialization vector to be used for the cipher block chaining for the DES encryption of the data in the *data_to_encrypt* parameter. The same initialization vector must be used to perform the DES decryption of the data.

RSA_public_key_identifier_length

Direction	Type
Input	Integer

The length of the *RSA_public_key_identifier* field. The maximum size is 2500 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

RSA_public_key_identifier

Direction	Type
Input	String

A string containing either the key label of the RSA public key or the RSA public key token to be used to perform the RSA encryption of the OAEP block. The modulus bit length of the key must be 1024 bits. This parameter is ignored if DES-ONLY is specified in the rule-array.

DES_key_block_length

Direction	Type
Input/Output	Integer

The length of the *DES_key_block*. The current length of this field is defined to be exactly 64 bytes.

DES_key_block

Direction	Type
Input/Output	String

The DES key information returned from a previous SET Block Compose service. The contents of the *DES_key_block* is the 64-byte DES internal key token (containing the DES key enciphered under the host master key). Your application program must not change the data in this string.

RSA_OAEP_block_length

Direction	Type
Input/Output	Integer

The length of a block of storage to hold the *RSA-OAEP_block*. The length must be at least 128 bytes on input. The length value will be updated on exit with the actual length of the *RSA-OAEP_block*, which is exactly 128 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

RSA_OAEP_block

Direction	Type
Output	String

The OAEP-formatted data block, encrypted under the RSA public key passed as *RSA_public_key_identifier*. When the OAEP-formatted data block is returned, it is left justified within the *RSA-OAEP_block* field if the input field length (*RSA-OAEP_block_length*) was greater than 128 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

chaining_vector

Direction	Type
Input/Output	String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. This field is ignored by this service, but must be specified.

DES_encrypted_data_block

Direction	Type
Output	String

The DES-encrypted data block (data passed in as *data_to_encrypt*). The length of the encrypted data is returned in *data_to_encrypt_length*. The *DES_encrypted_data_block* may be 8 bytes longer than the length of the *data_to_encrypt* because of padding added by this service.

Restrictions

Not all CCA implementations support a key label as input in the *RSA_public_key_identifier* parameter. Some implementations may only support a key token.

The *data_to_encrypt* and the *DES_encrypted_data_block* cannot overlap.

The maximum data block that can be supplied for DES encryption is the limit as expressed by the Encipher (CSNBENC) callable service.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The first time the SET Block Compose service is invoked to form an RSA-OAEP block and DES-encrypt data for communication between a specific source and destination (for example, between the merchant and payment gateway), do not specify the DES-ONLY keyword. A DES key will be generated by the service and returned in the key token contained in the *DES_key_block*. On subsequent calls to the Compose SET Block service for communication between the same source and destination, the DES key can be re-used.

The caller of the service must supply the *DES_key_block*, the *DES_key_block_length*, the *data_to_encrypt*, the *data_to_encrypt_length*, and the rule-array keywords SET1.00 and DES-ONLY. You do not need to supply the block contents identifier, XDATA string and length, RSA-OAEP block and length, and RSA public key information, although you must still specify the parameters. For this invocation, the RSA-OAEP formatting is bypassed and only DES encryption is performed, using the supplied DES key.

Access control point

The **SET Block Compose** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 359. SET Block Compose required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

SET Block Decompose (CSNDSBD and CSNFSBD)

Decomposes the RSA-OAEP block and the DES-encrypted data block of the SET protocol to provide unencrypted data back to the caller.

The callable service name for AMODE(64) invocation is CSNFSBD.

Format

```
CALL CSNDSBD(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
```

```
                RSA_OAEP_block_length,
                RSA_OAEP_block,
                DES_encrypted_data_block_length,
                DES_encrypted_data_block,
                initialization_vector,
                RSA_private_key_identifier_length,
                RSA_private_key_identifier,
                DES_key_block_length,
                DES_key_block,
                block_contents_identifier,
                XData_string_length,
                XData_string,
                chaining_vector,
                data_block,
                hash_block_length,
                hash_block)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 1 or 2.

rule_array

Direction	Type
Input	String

One keyword that provides control information to the callable service. The keyword indicates the block type. The keyword must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks.

Table 360. Keywords for SET Block Compose Control Information

Keyword	Meaning
Block Type (required)	
SET1.00	The structure of the RSA-OAEP encrypted block is defined by SET protocol.
Formatting Information (optional)	
DES-ONLY	DES decryption only is to be performed; no RSA-OAEP block decryption will be performed. (See Usage Notes.)
PINBLOCK	Specifies that the OAEP block will contain PIN information in the XDATA field, including an ISO-0 format PIN block. The *DES_key_block* must be 128 bytes in length and contain a IPINENC or OPINENC key. The PIN block will be encrypted under the PIN encrypting key. The PIN information and the encrypted PIN block are returned in the *XDATA_string* parameter.

RSA_OAEP_block_length

Direction	Type
Input	Integer

The length of *RSA-OAEP_block* must be 128 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

RSA_OAEP_block

Direction	Type
Input	String

The RSA-encrypted OAEP-formatted data block. This parameter is ignored if DES-ONLY is specified in the rule-array.

DES_encrypted_data_block_length

Direction	Type
Input/Output	Integer

The length in bytes of the DES-encrypted data block. The input length must be a multiple of 8 bytes. Updated on return to the length of the decrypted data returned in *data_block*. The maximum value of *DES_encrypted_data_block_length* is 32MB bytes.

DES_encrypted_data_block

Direction	Type
Input	String

The DES-encrypted data block. The data will be decrypted and passed back as *data_block*.

initialization_vector

Direction	Type
Input	String

An 8-byte string containing the initialization vector to be used for the cipher block chaining for the DES decryption of the data in the *DES_encrypted_data_block* parameter. You must use the same initialization vector that was used to perform the DES encryption of the data.

RSA_private_key_identifier_length

Direction	Type
Input	Integer

The length of the *RSA_private_key_identifier* field. The maximum size is 2500 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

RSA_private_key_identifier

Direction	Type
Input	String

A key label of the RSA private key or an internal token of the RSA private key to be used to decipher the RSA-OAEP block passed in *RSA-OAEP_block*. The modulus bit length of the key must be 1024. This parameter is ignored if DES-ONLY is specified in the rule-array.

DES_key_block_length

Direction	Type
Input/Output	Integer

The length of the *DES_key_block*. The current length of this field may be 64 or 128 bytes. If rule array keyword PINBLOCK is specified, the length must be 128 bytes.

DES_key_block

Direction	Type
Input/Output	String

The *DES_key_block* contains either one or two DES internal key tokens. If only one token is specified on input, it contains either a null DES token (or binary zeros) or (if DES-ONLY is specified) the DES key information returned from a previous SET Block Decompose service invocation. This is the 64-byte DES internal key token formed with the DES key which was retrieved from the RSA-OAEP block and enciphered under the host master key. Your application must not change this DES key information.
If two tokens are specified in the *DES_key_block*, the first 64 bytes contain the DES token described previously. The second 64 bytes, used when PINBLOCK is specified in the rule array, contains the DES internal token or the CKDS key label of the IPINENC or OPINENC key used to encrypt the PIN block returned to the caller in the XDATA_string parameter. If a key label is specified, it must be left-justified and padded on the right with blanks.

block_contents_identifier

Direction	Type
Output	String

A one-byte string, containing the binary value from the block contents (BC) field of the SET data block (DB). It indicates what data is carried in the actual data block (ADB) and the format of any extra data (*XData_string*). This parameter is ignored if DES-ONLY is specified in the rule-array.

XData_string_length

Direction	Type
Input/Output	Integer

The length of a string where the data contained within *XData_string* will be returned. The string must be at least 94 bytes in length. The value will be updated upon exit with the actual length of the returned *XData_string*. This parameter is ignored if DES-ONLY is specified in the rule-array.

XData_string

Direction	Type
Output	String

Extra-encrypted data contained within the OAEP-processed and RSA-encrypted block. The format is indicated by *block_contents_identifier*. The string is treated by ICSF as a string of hexadecimal digits. The service will always return the data from the beginning of the XDataString to the end of the SET DB block, a maximum of 94 bytes of data. The caller must examine the value returned in *block_contents_identifier* to determine the actual length of the XDataString. This parameter is ignored if DES-ONLY is specified in the rule-array.

chaining_vector

Direction	Type
Input/Output	String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. This field is ignored by this service, but must be specified.

data_block

Direction	Type
Output	String

The data that was decrypted (passed in as *DES_encrypted_data_block*). Any padding characters are removed.

hash_block_length

Direction	Type
Input/Output	Integer

The length in bytes of the SHA-1 hash returned in *hash_block*. On input, this parameter must be set to the length of the *hash_block* field. The length must be at least 20 bytes. On output, this field is updated to reflect the length of the SHA-1 hash returned in the *hash_block* field (exactly 20 bytes). This parameter is ignored if DES-ONLY is specified in the *rule_array* parameter.

hash_block

Direction	Type
Output	String

The SHA-1 hash extracted from the RSA-OAEP block. This parameter is ignored if DES-ONLY is specified in the *rule_array* parameter.

Restrictions

Not all CCA implementations support a key label as input in the *RSA_private_key_identifier* parameter. Some implementations may only support a key token.

The *data_block* and the *DES_encrypted_data_block* cannot overlap.

Usage notes

Triple-length DES keys requires the CCA release 6.7, 7.4, or later licensed internal code.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

When the SET Block Decompose service is invoked without the DES-ONLY keyword, the DES key is retrieved from the RSA-OAEP block and returned in the key token contained in the *DES_key_block*. On subsequent calls to the SET Block Decompose service, a caller can re-use the DES key. The caller of the service must supply the *DES_key_block*, the *DES_key_block_length*, the *DES_encrypted_data_block*, the *DES_encrypted_data_block_length*, the initialization and chaining vectors, and the *rule_array* keywords SET1.00 and DES-ONLY. The RSA private key information, RSA-OAEP block and length, XData string and length, and hash block and length need not be supplied (although the parameters must still be specified). For this invocation, the decryption of the RSA-OAEP block is bypassed; only DES decryption is performed, using the supplied DES key.

When the SET Block Decompose service is invoked with the PINBLOCK keyword, DES-ONLY may not also be specified. If both of these rule array keywords are specified, the service will fail. If PINBLOCK is specified and the *DES_key_block_length* field is not 128, the service will fail.

Access control points

The **SET Block Decompose** access control point controls the function of this service. If a PIN-block encrypting key is supplied in the *DES_key_block*, the access control point matching the key type of the key must be enabled in the domain role.

Table 361. Required access control points for PIN-block encrypting key

PIN-block encrypting key-type	Access control point
OPINENC	SET Block Decompose - PIN Extension OPINENC
IPINENC	SET Block Decompose - PIN Extension IPINENC

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 362. SET Block Decompose required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the CCA release 6.7 or later licensed internal code (LIC).

Table 362. SET Block Decompose required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Triple-length DES keys are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the CCA release 6.7 or later licensed internal code (LIC).
	Crypto Express7 CCA Coprocessor	Triple-length DES keys require the CCA release 7.4 or later licensed internal code (LIC).
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Transaction Validation (CSNBTRV and CSNETRV)

The Transaction Validation callable service supports the generation and validation of American Express card security codes (CSC). This service generates and verifies transaction values based on information from the transaction and a cryptographic key. You select the algorithm, validation method, and either the generate or verify mode, through rule-array keywords.

For the American Express process, the control vector supplied with the cryptographic key must indicate a MAC or MACVER class key. The key may be single or double length. DATAM and DATAMV keys are not supported. The MAC generate control vector bit must be on (bit 20) if you request CSC generation and MAC verify bit (bit 21) must be on if you request verification.

The callable service name for AMODE(64) invocation is CSNETRV.

Format

```
CALL CSNBTRV(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              transaction_key_identifier_length,
              transaction_key_identifier,
              transaction_info_length,
              transaction_info,
              validation_values_length,
              validation_values )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The valid values are 1, 2, or 3.

rule_array

Direction	Type
Input	Character String

Keywords that provides control information to the callable service. The keywords are left-justified in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. Specify one, two or three of the values in Table 363 on page 835.

Table 363. Rule Array Keywords for Transaction Validation

Keyword	Meaning
American Express card security codes (required)	
CSC-3	3-digit card security code (CSC) located on the signature panel. **VERIFY** implied.
CSC-4	4-digit card security code (CSC) located on the signature panel. **VERIFY** implied.
CSC-5	5-digit card security code (CSC) located on the signature panel. **VERIFY** implied.
CSC-345	Generate 5-byte, 4-byte, 3-byte values when given an account number an an expiration date, **GENERATE** implied.
Operation (optional)	

Table 363. Rule Array Keywords for Transaction Validation (continued)	
Keyword	**Meaning**
VERIFY	Specifies verification of the value presented in the validation values variable.
GENERATE	Specifies generation of the value presented in the validation values variable.
Card Security Code Algorithm (One, optional)	
CSC-V1	Specifies use of CSC version 1.0 algorithm for generating or verifying the validation values.
CSC-V2	Specifies use of CSC version 2.0 algorithm for generating or verifying the validation values.

transaction_key_identifier_length

Direction	Type
Input	Integer

The length of the *transaction_key_identifier* parameter.

When *transaction_key_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

transaction_key_identifier

Direction	Type
Input	String

The identifier of the MAC key to generate or verify the card security code. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For CCA key tokens, the identifier is a 64-byte key token containing a single-length or double-length DES MAC or MACVER key. When the CSC-V2 keyword is specified, the key must be a double-length key.

For X9.143 key blocks, the identifier is a variable-length key containing a DES MAC key: key usage C0, algorithm D or T, and mode of use C, G or V. The mode of use must match the operation rule specified or implied. Rule GENERATE requires mode C or G. Rule VERIFY requires mode C or V. When the CSC-V2 keyword is specified, the key must be a double-length key.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

transaction_info_length

Direction	Type
Input	Integer

The length of the *transaction_info* parameter. For American Express CSC codes, this length must be 19 if the algorithm for CSC v1.0 is specified and it must be 22 if the algorithm for CSC v2.0 is specified.

transaction_info

Direction	Type
Input	String

Account information in character format. For American Express CSC-V1, this is a 19-byte field containing the concatenation of the 4-byte expiration data (in the format YYMM) and the 15-byte American Express account number. For CSC-V2, the string variable will contain the concatenation of the 4-byte expiration date in the format of (YYMM) , the 15-byte American Express account number and the 3-byte service code.

validation_values_length

Direction	Type
Input/Output	Integer

The length of the *validation_values* parameter. Maximum value for this field is 64.

validation_values

Direction	Type
Input	String

This variable contains American Express CSC values. The data is output for **GENERATE** and input for **VERIFY**.

Table 364. Output description for validation values

Operation	Element Description
GENERATE and CSC-345	5555544444333 where: 55555 = CSC 5 value 4444 = CSC 4 value 333 = CSC 3 value
VERIFY and CSC-3	333 = CSC 3 value
VERIFY and CSC-4	4444 = CSC 4 value
VERIFY and CSC-5	55555 = CSC 5 value

Restrictions

Triple-length DES keys are not supported.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 365. Required access control points for Transaction Validation

Operation keyword	Security code keyword	Access control point
GENERATE	CSC-345	Transaction Validation - Generate
VERIFY	CSC-3	Transaction Validation - Verify CSC-3
VERIFY	CSC-4	Transaction Validation - Verify CSC-4
VERIFY	CSC-5	Transaction Validation - Verify CSC-5

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 366. Transaction Validation required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

VISA CVV Service Generate (CSNBCSG and CSNECSG)

Use the VISA CVV Service Generate callable service to generate a:

- VISA Card Verification Value (CVV)
- MasterCard Card Verification Code (CVC)
- Diner's Club Card Verification Value (CVV)

as defined for track 2.

This service generates a CVV that is based upon the information that the *PAN_data*, the *expiration_date*, and the *service_code* parameters provide.

The service uses the Key-A and the Key-B keys to cryptographically process this information. Key-A and Key-B can be single-length DATA or MAC keys or a combined Key-A, Key-B double length MAC key. If the requested CVV is shorter than 5 characters, the CVV is padded on the right by space characters. The CVV is returned in the 5-byte variable that the *CVV_value* parameter identifies. When you verify a CVV, compare the result to the value that the *CVV_value* supplies.

The callable service name for AMODE(64) invocation is CSNECSG.

Format

```
CALL CSNBCSG(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              PAN_data,
              expiration_date,
              service_code,
              CVV_key_A_Identifier,
              CVV_key_B_Identifier,
              CVV_value)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The parameter *rule_array_count* must be 0, 1, or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage.

Table 367. CVV Generate Rule Array Keywords

Keyword	Meaning
PAN data length (optional)	
PAN-13	Specifies that the length of the PAN data is 13 bytes. **PAN-13 is the default value.**
PAN-14	Specifies that the length of the PAN data is 14 bytes.
PAN-15	Specifies that the length of the PAN data is 15 bytes.
PAN-16	Specifies that the length of the PAN data is 16 bytes.
PAN-17	Specifies that the length of the PAN data is 17 bytes.
PAN-18	Specifies that the length of the PAN data is 18 bytes.
PAN-19	Specifies that the length of the PAN data is 19 bytes.
CVV length (optional)	
CVV-1	Specifies that the CVV is to be computed as one byte, followed by 4 blanks. **CVV-1 is the default value.**
CVV-2	Specifies that the CVV is to be computed as 2 bytes, followed by 3 blanks.
CVV-3	Specifies that the CVV is to be computed as 3 bytes, followed by 2 blanks.
CVV-4	Specifies that the CVV is to be computed as 4 bytes, followed by 1 blank.
CVV-5	Specifies that the CVV is to be computed as 5 bytes.

PAN_data

Direction	Type
Input	String

The *PAN_data* parameter specifies an address that points to the place in application data storage that contains personal account number (PAN) information in character form. The PAN is the account number as defined for the track-2 magnetic-stripe standards.

- If the **PAN-13** keyword is specified in the rule array, 13 characters are processed.
- If the **PAN-14** keyword is specified in the rule array, 14 characters are processed.
- If the **PAN-15** keyword is specified in the rule array, 15 characters are processed.
- If the **PAN-16** keyword is specified in the rule array, 16 characters are processed.
- If the **PAN-17** keyword is specified in the rule array, 17 characters are processed.
- If the **PAN-18** keyword is specified in the rule array, 18 characters are processed.
- If the **PAN-19** keyword is specified in the rule array, 19 characters are processed.

Even if you specify the **PAN-13**, **PAN-14** or **PAN-15** keywords, the server might copy 16 bytes to a work area. Therefore ensure that the callable service can address 16 bytes of storage.

expiration_date

Direction	Type
Input	String

The *expiration_date* parameter specifies an address that points to the place in application data storage that contains the card expiration date in numeric character form in a 4-byte field. The application programmer must determine whether the CVV will be calculated with the date form of YYMM or MMYY.

service_code

Direction	Type
Input	String

The *service_code* parameter specifies an address that points to the place in application data storage that contains the service code in numeric character form in a 3-byte field. The service code is the number that the track-2 magnetic-stripe standards define. The service code of '000' is supported.

CVV_key_A_Identifier

Direction	Type
Input/Output	String

The identifier of the MAC key to generate the card verification value. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage.

This key identifier may be a single-length key or a double-length key. When a double-length key is supplied, CCV_key_B_identifer must be a 64-byte null token.

For CCA key tokens

- this is a 64-byte single-length DES DATA or MAC key.
- this is a 64-byte double-length DES MAC key. The control vector bits 0-3 must indicate a CVVKEY-A key (0010).

For X9.143 keys

The identifier is a variable-length DES or TDES MAC key: key usage C0, algorithm D or T, and mode of use C or G.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

CVV_key_B_Identifier

Direction	Type
Input/Output	String

The identifier of the MAC key to generate the card verification value. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage.

For CCA key tokens, this is a 64-byte single-length DES DATA or MAC key.

For X9.143 keys, the identifier is a variable-length DES MAC key: key usage C0, algorithm D, and mode of use C or G.

When *CVV_key_A_identifier* is a double-length key, this parameter must be 64 bytes of binary zero.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

CVV_value

Direction	Type
Output	String

The *CVV_value* parameter specifies an address that points to the place in application data storage that will be used to store the computed 5-byte character output value.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control point

The **VISA CVV Generate** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 368. VISA CVV Service Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

VISA CVV Service Verify (CSNBCSV and CSNECSV)

Use the VISA CVV Service Verify callable service to verify a:

- VISA Card Verification Value (CVV)
- MasterCard Card Verification Code (CVC)
- Diner's Club Card Verification Value (CVV)

as defined for track 2.

This service verifies a CVV that is based upon the information that the *PAN_data*, the *expiration_date*, and the *service_code* parameters provide.

The service uses the Key-A and the Key-B keys to cryptographically process this information. If the requested CVV is shorter than 5 characters, the CVV is padded on the right by space characters. The generated CVV is then compared to the value that the *CVV_value* supplies for verification.

The callable service name for AMODE(64) invocation is CSNECSV.

Format

```
CALL CSNBCSV(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              PAN_data,
              expiration_date,
              service_code,
              CVV_key_A_Identifier,
              CVV_key_B_Identifier,
              CVV_value)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The parameter *rule_array_count* must be 0, 1, or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage.

Table 369. CVV Verify Rule Array Keywords

Keyword	Meaning
PAN data length (optional)	
PAN-13	Specifies that the length of the PAN data is 13 bytes. **PAN-13 is the default value.**
PAN-14	Specifies that the length of the PAN data is 14 bytes.
PAN-15	Specifies that the length of the PAN data is 15 bytes.
PAN-16	Specifies that the length of the PAN data is 16 bytes.
PAN-17	Specifies that the length of the PAN data is 17 bytes.
PAN-18	Specifies that the length of the PAN data is 18 bytes.
PAN-19	Specifies that the length of the PAN data is 19 bytes.
CVV length (optional)	
CVV-1	Specifies that the CVV is to be computed as one byte, followed by 4 blanks. **CVV-1 is the default value.**
CVV-2	Specifies that the CVV is to be computed as 2 bytes, followed by 3 blanks.
CVV-3	Specifies that the CVV is to be computed as 3 bytes, followed by 2 blanks.
CVV-4	Specifies that the CVV is to be computed as 4 bytes, followed by 1 blank.
CVV-5	Specifies that the CVV is to be computed as 5 bytes.

PAN_data

Direction	Type
Input	String

The *PAN_data* parameter specifies an address that points to the place in application data storage that contains personal account number (PAN) information in character form. The PAN is the account number as defined for the track-2 magnetic-stripe standards.

- If the **PAN-13** keyword is specified in the rule array, 13 characters are processed.
- If the **PAN-14** keyword is specified in the rule array, 14 characters are processed.
- If the **PAN-15** keyword is specified in the rule array, 15 characters are processed.
- If the **PAN-16** keyword is specified in the rule array, 16 characters are processed.
- If the **PAN-17** keyword is specified in the rule array, 17 characters are processed.
- If the **PAN-18** keyword is specified in the rule array, 18 characters are processed.
- If the **PAN-19** keyword is specified in the rule array, 19 characters are processed.

Even if you specify the **PAN-13**, **PAN-14** or **PAN-15** keywords, the server might copy 16 bytes to a work area. Therefore ensure that the callable service can address 16 bytes of storage.

expiration_date

Direction	Type
Input	String

The *expiration_date* parameter specifies an address that points to the place in application data storage that contains the card expiration date in numeric character form in a 4-byte field. The application programmer must determine whether the CVV will be calculated with the date form of YYMM or MMYY.

service_code

Direction	Type
Input	String

The *service_code* parameter specifies an address that points to the place in application data storage that contains the service code in numeric character form in a 3-byte field. The service code is the number that the track-2 magnetic-stripe standards define. The service code of '000' is supported.

CVV_key_A_Identifier

Direction	Type
Input/Output	String

The identifier of the MAC key to verify the card verification value. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage.

This key identifier may be a single-length key or a double-length key. When a double-length key is supplied, *CCV_key_B_identifer* must be a 64-byte null token.

For CCA key tokens

- This is a 64-byte single-length DES DATA, MAC, or MACVER key.
- This is a 64-byte double-length DES MAC key. The control vector bits 0-3 must indicate a CVVKEY-A key (0010).

For X9.143 keys
The identifier is a variable-length DES or TDES MAC key: key usage C0, algorithm D or T, and mode of use C or V.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

CVV_key_B_Identifier

Direction	Type
Input/Output	String

The identifier of the MAC key to verify the card verification value. The key identifier is a variable-length operational key token or key block or the 64-byte label of an operational token or block in key storage.

For CCA key tokens, this is a 64-byte single-length DES DATA, MAC, or MACVER key.

For X9.143 keys, the identifier is a variable-length DES MAC key: key usage C0, algorithm D, and mode of use C or V.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

CVV_value

Direction	Type
Input	String

The *CVV_value* parameter specifies an address that contains the CVV value which will be compared to the computed CVV value. This is a 5-byte field.

On an IBM zSeries 900, the user must pad out the CVV_value parameter with blanks if the supplied CVV is less than 5 characters.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

The **VISA CVV Verify** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 370. VISA CVV Service Verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.

Table 370. VISA CVV Service Verify required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Chapter 9. Financial services for DK PIN methods

This section provides information on financial services that are based on the PIN methods of and meet the requirements specified by the German Banking Industry Committee (Deutsche Kreditwirtschaft (DK)). DK is an association of the German banking industry. The intellectual property rights regarding the methods and specification belongs to the German Banking Industry Committee.

Note: All crypto coprocessors must be loaded with the same level of code. There have been several licensed internal code (LIC) released in support of the DK PIN methods. Ensure that all of the coprocessors have the same LIC level to support the function you want to use.

The callable services that support the German Banking Industry Committee (Deutsche Kreditwirtschaft (DK)) PIN methods are:

- "DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG)" on page 850
- "DK Migrate PIN (CSNBDMP and CSNEDMP)" on page 857
- "DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)" on page 863
- "DK PAN Translate (CSNBDPT and CSNEDPT)" on page 871
- "DK PIN Change (CSNBDPC and CSNEDPC)" on page 878
- "DK PIN Verify (CSNBDPV and CSNEDPV)" on page 892
- "DK PRW Card Number Update (CSNBDPNU and CSNEDPNU)" on page 897
- "DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2)" on page 904
- "DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG)" on page 912
- "DK Random PIN Generate (CSNBDRPG and CSNEDRPG)" on page 916
- "DK Random PIN Generate2 (CSNBDRG2 and CSNEDRG2)" on page 922
- "DK Regenerate PRW (CSNBDRP and CSNEDRP)" on page 930

Weak PIN table

The DK PIN methods support the use of a table of weak PINs. Services that generate PINs compare the generated PIN against the table and if the PIN is in the table, the service generates a different PIN. Services that change PINs compare the new PIN against the table and if the new PIN is in the table, the service fails.

Weak PIN tables can be stored in the cryptographic coprocessors for use by callable services. Only tables that have been activated can be used. A TKE Workstation is required to manage the tables in the coprocessors.

Note: ICSF routes work to all active coprocessors based on work load. All coprocessors must have the same set of PINs.

DK PIN methods

The DK PIN methods use a PIN Reference Value (PRW) to verify PINs rather than regenerating the PIN from customer account data. The PRW is generated by concatenating the customer PAN data, the issuer card data, the PIN length, the PIN, and a 4-byte random number and encrypting using a PRW key with the GENONLY key usage. The PRW and random number are the output of the generation. The PIN is verified by generating the PRW using a PRW key with the VERIFY key usage and comparing it against the supplied PRW and random number.

DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG)

Use the DK Deterministic PIN Generate callable service to generate a PIN and PIN reference value (PRW) using an AES PIN calculation key. The PIN reference value is used to verify the PIN in other services.

Note: If the generated PIN appears in the weak PIN table, the generation process is retried by appending to the account information until an acceptable PIN is generated. For additional information, see "Weak PIN table" on page 849.

You can use this service to perform the following tasks:

- Generate an encrypted PIN block in PBF-1 format with a PIN print key to be printed on a PIN mailer.
- Generate a PRW reference value which can be used to verify the PIN.
- Optionally, generate an encrypted PIN block in PBF-1 format to be stored for later use in personalizing replacement cards.

The callable service name for AMODE(64) invocation is CSNEDDPG.

Format

```
CALL CSNBDDPG(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              account_info_ER_length,
              account_info_ER,
              PAN_data_length,
              PAN_data,
              card_p_data_length,
              card_p_data,
              card_t_data_length,
              card_t_data,
              PIN_length,
              PIN_generation_key_identifier_length,
              PIN_generation_key_identifier,
              PRW_key_identifier_length,
              PRW_key_identifier,
              PIN_print_key_identifier_length,
              PIN_print_key_identifier,
              OPIN_encryption_key_identifier_length,
              OPIN_encryption_key_identifier,
              OEPB_MAC_key_identifier_length,
              OEPB_MAC_key_identifier,
              PIN_reference_value_length,
              PIN_reference_value,
              PRW_random_number_length,
              PRW_random_number,
              PIN_print_block_length,
              PIN_print_block,
              encrypted_PIN_block_length,
              encrypted_PIN_block,
              PIN_block_MAC_length,
              PIN_block_MAC)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0 or 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. There are no keywords for this service.

Table 371. Rule array keywords for the DK Deterministic PIN Generate service

Keyword	Meaning
PIN Block output selection keyword (One, optional)	
NOEPB	Do not return an encrypted PIN block (EPB). This is the default value.
EPB	Return an encrypted PIN block and a MAC of the encrypted PIN block.

account_info_ER_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *account_info_ER* parameter. The value must be 16.

account_info_ER

Direction	Type
Input	String

The 16-byte account information used to generate the PIN.

PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PAN_data* parameter. The value must be between 10 and 19, inclusive.

PAN_data

Direction	Type
Input	String

The PAN data which the PIN is associated. The full account number, including check digit, should be included. This parameter is character data.

card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_p_data* parameter. The value must be between 2 and 256, inclusive.

card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_t_data* parameter. The value must be between 2 and 256, inclusive.

card_t_data

Direction	Type
Input	String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the card_p_data, specifies an individual card.

PIN_length

Direction	Type
Input	Integer

Specifies the length of the PIN to be generated. This value must be between 4 and 12, inclusive.

PIN_generation_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_generation_key_identifier* parameter. If the *PIN_generation_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 725.

PIN_generation_key_identifier

Direction	Type
Input/Output	String

The identifier of the PIN generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINCALC, the key usage fields must indicate GENONLY, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the PRW generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

PIN_print_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_print_key_identifier* parameter. If the *PIN_print_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 725.

PIN_print_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the PIN for printing. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOPP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_encryption_key_identifier* parameter. If the rule array indicates that no encrypted PIN block is to be returned, this value must be 0. If the *OPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OEPB_MAC_key_identifier* parameter. If the rule array indicates that no encrypted PIN block MAC is to be returned, this value must be 0. If the *OEPB_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC of the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, GENONLY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

PIN_reference_value_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. The value must be at least 16. On output, it will be set to 16.

PIN_reference_value

Direction	Type
Output	String

The 16-byte calculated PIN reference value.

PRW_random_number_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be at least 4. On output, it will be set to 4.

PRW_random_number

Direction	Type
Output	String

The 4-byte random number associated with the PIN reference value.

PIN_print_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_print_block* parameter. The value must be at least 32. On output, it will be set to 32.

PIN_print_block

Direction	Type
Output	String

The 32-byte encrypted PIN block to be passed to the PIN mailer function.

encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *encrypted_PIN_block* parameter. If the rule array indicates that no encrypted PIN block should be returned, this value must be 0. Otherwise, it should be at least 32.

encrypted_PIN_block

Direction	Type
Output	String

The 32-byte encrypted PIN block in PBF-1 format. This parameter is ignored if no encrypted PIN block is returned.

PIN_block_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_block_MAC* parameter. If the rule_array indicates that no PIN block MAC should be returned, this value must be 0. Otherwise, it must be at least 8.

PIN_block_MAC

Direction	Type
Output	String

The 8-byte CMAC of the encrypted PIN block. This parameter is ignored if no encrypted PIN block is returned.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK Deterministic PIN Generate** access control point in the domain role controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 372. DK Deterministic PIN Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	

Table 372. DK Deterministic PIN Generate required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK Migrate PIN (CSNBDMP and CSNEDMP)

Use the DK Migrate PIN callable service to generate the PIN reference value (PRW) for a specified user account. An ISO-1 formatted PIN block is input to determine the value of the PIN for the account. The PIN is reformatted into a DK-defined PIN block and the PIN reference value is calculated using a PRW random value and other account information. The PIN reference value and associated PRW random value are returned to be used as input by other PIN processes to verify the PIN.

If validation of the PIN is desired to personalize smart cards, specify the EPB PIN block output selection rule-array keyword. This keyword causes an output encrypted PIN block to be returned along with a PIN block MAC. The MAC is calculated over the output PIN block and additional card data using the block cipher-based MAC algorithm called CMAC (NIST SP 800-38B).

Note: Regarding weak PINs, this service does not test for weak PINs.

The callable service name for AMODE(64) invocation is CSNEDMP.

Format

```
CALL CSNBDMP(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              PAN_data_length,
              PAN_data,
              card_p_data_length,
              card_p_data,
              card_t_data_length,
              card_t_data,
              ISO1_PIN_block_length,
              ISO1_PIN_block,
              IPIN_encryption_key_identifier_length,
              IPIN_encryption_key_identifier,
              PRW_key_identifier_length,
              PRW_key_identifier,
              OPIN_encryption_key_identifier_length,
              OPIN_encryption_key_identifier,
              OEPB_MAC_key_identifier_length,
              OEPB_MAC_key_identifier,
              PIN_reference_value_length,
              PIN_reference_value,
              PRW_random_number_length,
              PRW_random_number,
              encrypted_PIN_block_length,
              encrypted_PIN_block,
              PIN_block_MAC_length,
              PIN_block_MAC)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0 or 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. There are no keywords for this service.

Table 373. Rule array keywords for the DK Migrate PIN service

Keyword	Meaning
PIN Block output selection keyword (One, optional)	
NOEPB	Do not return an encrypted PIN block (EPB). This is the default value.

Table 373. Rule array keywords for the DK Migrate PIN service (continued)	
Keyword	**Meaning**
EPB	Return an encrypted PIN block and a MAC of the encrypted PIN block.

PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PAN_data* parameter. The value must be between 10 and 19, inclusive.

PAN_data

Direction	Type
Input	String

The personal account number in character form which the PIN will be associated. The primary account number, including check digit, should be included.

card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_p_data* parameter. The value must be between 2 and 256, inclusive.

card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_t_data* parameter. The value must be between 2 and 256, inclusive.

card_t_data

Direction	Type
Input	String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the card_p_data, specifies an individual card.

ISO1_PIN_block_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *ISO1_PIN_block* parameter. This value must be 8.

ISO1_PIN_block

Direction	Type
Input	String

The 8-byte encrypted PIN block with the current PIN in ISO-1 format with the customer chosen PIN. This PIN is used to generate the PIN reference value.

IPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IPIN_encryption_key_identifier* parameter. If the *IPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

IPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the PIN_block containing the ISO-1 PIN. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be DES and the key type must be IPINENC.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* parameter contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the PRW generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_encryption_key_identifier* parameter. If the rule array indicates that no encrypted PIN block is to be returned, this value must be 0. If the *OPIN_encryption_key_identifier* parameter contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

OEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OEPB_MAC_key_identifier* parameter. If the rule array indicates that no encrypted PIN block MAC is to be returned, this value must be 0. If the *OEPB_MAC_key_identifier* parameter contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC of PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

PIN_reference_value_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. This value must be 16. On output, *PIN_reference_value_length* will be set to 16.

PIN_reference_value

Direction	Type
Output	String

The 16-byte calculated PIN reference value.

PRW_random_number_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be 4. On output, *PRW_random_number_length* will be set to 4.

PRW_random_number

Direction	Type
Output	String

The 4-byte random number associated with the PIN reference value.

encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *encrypted_PIN_block* parameter. If the rule array indicates that no encrypted PIN block should be returned, this value must be 0. Otherwise, it should be at least 32.

encrypted_PIN_block

Direction	Type
Output	String

The 32-byte encrypted PIN block in PBF-1 format. This parameter is ignored if no encrypted PIN block is returned.

PIN_block_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_block_MAC* parameter. If the rule_array indicates that no PIN block MAC should be returned, this value must be 0. Otherwise, it must be at least 8.

PIN_block_MAC

Direction	Type
Output	String

The 8-byte CMAC of the encrypted PIN block. This parameter is ignored if no encrypted PIN block is returned.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK Migrate PIN** access control point in the domain role controls the function of this service.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, this service will fail.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 374. DK Migrate PIN required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)

Use the DK PAN Modify in Transaction callable service to generate a new PIN reference value (PRW) for an existing PIN when a merger has occurred and the account information has changed. The inputs include the current PIN, the account information (PAN and card data) for the current, and the new account.

The DK PRW CMAC Generate service is called prior to this service to generate the MAC of the changed account information. If the MAC associated with the account information does not verify, the service fails.

The callable service name for AMODE(64) invocation is CSNEDPMT.

Format

```
CALL CSNBDPMT(
             return_code,
```

```
                reason_code,
                exit_data_length,
                exit_data,
                rule_array_count,
                rule_array,
                current_PAN_data_length,
                current_PAN_data,
                new_PAN_data_length,
                new_PAN_data,
                current_card_p_data_length,
                current_card_p_data,
                current_card_t_data_length,
                current_card_t_data,
                new_card_p_data_length,
                new_card_p_data,
                new_card_t_data_length,
                new_card_t_data,
                CMAC_FUS_length,
                CMAC_FUS,
                ISO_encrypted_PIN_block_length,
                ISO_encrypted_PIN_block,
                current_PIN_reference_value_length,
                current_PIN_reference_value,
                current_PRW_random_number_length,
                current_PRW_random_number,
                CMAC_FUS_key_identifier_length,
                CMAC_FUS_key_identifier,
                IPIN_encryption_key_identifier_length,
                IPIN_encryption_key_identifier,
                PRW_key_identifier_length,
                PRW_key_identifier,
                new_PRW_key_identifier_length,
                new_PRW_key_identifier,
                new_PIN_reference_value_length,
                new_PIN_reference_value,
                new_PRW_random_number_length,
                new_PRW_random_number)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0 or 1.

rule_array

Direction	Type
Input	Character

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 375. Keywords for the DK PIN Verify Service

Keyword	Meaning
PIN Block format (One, optional)	
ISO-1	Specifies that the encrypted PIN block in the *ISO_encrypted_PIN_block* parameter is in the ISO-1 format. This is the default.
ISO-4	Specifies that the encrypted PIN block in the *ISO_encrypted_PIN_block* parameter is in the ISO-4 format.

current_PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *current_PAN_data* parameter. The value must be between 10 and 19, inclusive.

current_PAN_data

Direction	Type
Input	Character

The current PAN data associated with the PIN. The full account number, including check digit, should be included. This parameter is character data.

new_PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_PAN_data* parameter. The value must be between 10 and 19, inclusive.

new_PAN_data

Direction	Type
Input	Character

The new PAN data to be associated with the PIN. The full account number, including check digit, should be included. This parameter is character data.

current_card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *current_card_p_data* parameter. The value must be between 2 and 256, inclusive.

current_card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp) of the current account, determined by the card issuer.

current_card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *current_card_t_data* parameter. The value must be between 2 and 256, inclusive.

current_card_t_data

Direction	Type
Input	String

The time-invariant card data (CDp) of the current account, determined by the card issuer.

new_card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_card_p_data* parameter. The value must be between 2 and 256, inclusive.

new_card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp) of the current account, determined by the card issuer.

new_card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_card_t_data* parameter. The value must be between 2 and 256, inclusive.

new_card_t_data

Direction	Type
Input	String

The time-invariant card data (CDp) of the current account, determined by the card issuer.

CMAC_FUS_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *CMAC_FUS* parameter. The value must be between 8 and 16, inclusive.

CMAC_FUS

Direction	Type
Input	String

The 8-byte to 16-byte MAC that was of the current and new PANs and card data strings and PIN reference values. The MAC is generated using the DK PRW CMAC Generate service.

ISO_encrypted_PIN_block_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *encrypted_PIN_block* parameter. The value must be 8 when ISO-1 is specified and 16 when ISO-4 is specified.

ISO_encrypted_PIN_block

Direction	Type
Input	String

The encrypted PIN block with the PIN in ISO-1 or ISO-4 format.

current_PIN_reference_value_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *current_PIN_reference_value* parameter. The value must be 16.

current_PIN_reference_value

Direction	Type
Input	String

The 16-byte PIN reference value for comparison to the calculated value.

current_PRW_random_number_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *current_PRW_random_number* parameter. The value must be 4.

current_PRW_random_number

Direction	Type
Input	String

The 4-byte random number associated with the PIN reference value.

CMAC_FUS_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *CMAC_FUS_key_identifier* parameter. If the *CMAC_FUS_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

CMAC_FUS_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify the *CMAC_FUS* value. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate VERIFY, CMAC, and DKPINAD2.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

IPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IPIN_encryption_key_identifier* parameter. If the *IPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

IPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the *encrypted_PIN_block*. The key identifier is an operational token or the key label of an operational token in key storage.

When ISO-1 is specified, the key algorithm of this key must be DES and the key type must be IPINENC. The control vector must enable the verification of an encrypted PIN (EPINVER bit 19 = B'1').

When ISO-4 is specified, the key algorithm of this key must be AES and the key type must be PINPROT. The key usage fields must specify DECRYPT, CBC, NOFLDFMT, and EPINVER.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify the input PRW. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

new_PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_PRW_key_identifier* parameter. If the *new_PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

new_PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the new PRW. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

new_PIN_reference_value_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_PIN_reference_value* parameter. The value must be at least 16. On output, it will be set to 16.

new_PIN_reference_value

Direction	Type
Output	String

The 16-byte new PIN reference value.

new_PRW_random_number_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_PRW_random_number* parameter. The value must be at least 4. On output, it will be set to 4.

new_PRW_random_number

Direction	Type
Output	String

The 4-byte random number associated with the new PIN reference value.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK PAN Modify in Transaction** access control point in the domain role controls the function of this service.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format *rule_array* keyword ISO-1 is not allowed.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 376. DK PAN Modify in Transaction required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords ISO-1 and ISO-4 and triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Keywords ISO-1 and ISO-4 and triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Keywords ISO-1 and ISO-4 and triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).

Table 376. DK PAN Modify in Transaction required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK PAN Translate (CSNBDPT and CSNEDPT)

Use the DK PAN Translate callable service to create an encrypted PIN block with the same PIN and a different PAN. The account data may change, but changing the PIN is to be avoided. This service specifically creates a new encrypted PIN block and MAC on that encrypted PIN block, which will be used to accept the PAN change at an authorization node.

You can use this service to perform the following tasks:

- Generate an encrypted PIN block in PBF-1 format with a changed PAN to be used at the authorization node to create a PIN reference value.
- Generate a CMAC over the encrypted PIN block for validation.

The callable service name for AMODE(64) invocation is CSNEDPT.

Format

```
CALL CSNBDPT(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             card_p_data_length,
             card_p_data,
             card_t_data_length,
             card_t_data,
             new_PAN_data_length,
             new_PAN_data,
             new_card_p_data_length,
             new_card_p_data,
             PIN_reference_value_length,
             PIN_reference_value,
             PRW_random_number_length,
             PRW_random_number,
             current_encrypted_PIN_block_length,
             current_encrypted_PIN_block,
             current_PIN_block_MAC_length,
             current_PIN_block_MAC,
             PRW_key_identifier_length,
             PRW_key_identifier,
             IPIN_encryption_key_identifier_length,
             IPIN_encryption_key_identifier,
             IEPB_MAC_key_identifier_length,
             IEPB_MAC_key_identifier,
```

```
                OPIN_encryption_key_identifier_length,
                OPIN_encryption_key_identifier,
                OEPB_MAC_key_identifier_length,
                OEPB_MAC_key_identifier,
                new_encrypted_PIN_block_length,
                new_encrypted_PIN_block,
                new_PIN_block_MAC_length,
                new_PIN_block_MAC)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. There are no keywords for this service.

card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_p_data* parameter. The value must be between 2 and 256, inclusive.

card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_t_data* parameter. The value must be between 2 and 256, inclusive.

card_t_data

Direction	Type
Input	String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the *card_p_data*, specifies an individual card.

new_PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_PAN_data* parameter. This value must be between 10 and 19, inclusive.

new_PAN_data

Direction	Type
Input	String

The new personal account number (in character form) which the PIN will be associated. The full account number, including check digit, should be included.

new_card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_card_p_data* parameter. The value must be between 2 and 256, inclusive.

new_card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

PIN_reference_value_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. This value must be 16.

PIN_reference_value

Direction	Type
Input	String

The 16-byte PIN reference value for comparison to the calculated value.

PRW_random_number_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be 4.

PRW_random_number

Direction	Type
Input	String

The 4-byte random number associated with the PIN reference value.

current_encrypted_PIN_block_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *current_encrypted_PIN_block* parameter. The value must be 32.

current_encrypted_PIN_block

Direction	Type
Input	String

The 32-byte encrypted PIN block in PBF-1 format of the current PIN.

current_PIN_block_MAC_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *current_PIN_block_MAC* parameter. The value must be 8.

current_PIN_block_MAC

Direction	Type
Input	String

The 8-byte MAC of the current encrypted PIN block and the *card_p_data*.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the PRW verifying key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IPIN_encryption_key_identifier* parameter. If the *IPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

IPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the PIN block containing the current PIN. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate DECRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IEPB_MAC_key_identifier* parameter. If the *IEPB_MAC_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 725.

IEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify MAC of the inbound encrypted PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC. VERIFY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_encryption_key_identifier* parameter. If the *OPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to encrypt the new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINAD1.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_OEPB_MAC_key_identifier* parameter. If the *new_OEPB_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC of the new encrypted PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, GENONLY, and DKPINAD1.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

new_encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_encrypted_PIN_block* parameter. The value must be at least 32. On output, it will be set to 32.

new_encrypted_PIN_block

Direction	Type
Output	String

The 32-byte encrypted new PIN block.

new_PIN_block_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_PIN_block_MAC* parameter. The value must be at least 8.

new_PIN_block_MAC

Direction	Type
Output	String

The 8-byte MAC of the new encrypted PIN block.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK PAN Translate** access control point in the domain role controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 377. DK PAN Translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).

Table 377. DK PAN Translate required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK PIN Change (CSNBDPC and CSNEDPC)

Use the DK PIN Change callable service to allow a customer to change their PIN to a value of their choosing.

The current and new PINs are entered into the ATM, where they are encrypted into ISO-1 or ISO-4 PIN blocks. The PIN and other needed information are used to verify the current PIN. If the PIN does not verify, the process is aborted. If the PIN does verify, the PIN is reformatted into a PBF-O format and the provided information is used to create a new PIN reference value.

Note: Regarding weak PINs, if the new PIN specified appears in the weak PIN table, the PIN change fails with an indication that the selected new PIN was not valid.

The callable service name for AMODE(64) invocation is CSNEDPC.

Format

```
CALL CSNBDPC(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              PAN_data_length,
              PAN_data,
              card_p_data_length,
              card_p_data,
              card_t_data_length,
              card_t_data,
              cur_ISO_PIN_block_length,
              cur_ISO_PIN_block,
              new_ISO_PIN_block_length,
              new_ISO_PIN_block,
              card_script_data_length,
              card_script_data,
              script_offset,
              script_offset_field_length,
              script_initialization_vector_length,
              script_initialization_vector,
              output_PIN_profile,
              PIN_reference_value_length,
              PIN_reference_value,
              PRW_random_number_length,
              PRW_random_number,
              PRW_key_identifier_length,
              PRW_key_identifier,
```

```
                cur_IPIN_encryption_key_identifier_length,
                cur_IPIN_encryption_key_identifier,
                new_IPIN_encryption_key_identifier_length,
                new_IPIN_encryption_key_identifier,
                script_key_identifier_length,
                script_key_identifier,
                script_MAC_key_identifier_length,
                script_MAC_key_identifier,
                new_PRW_key_identifier_length,
                new_PRW_key_identifier,
                OPIN_encryption_key_identifier_length,
                OPIN_encryption_key_identifier,
                OEPB_MAC_key_identifier_length,
                OEPB_MAC_key_identifier,
                script_length,
                script,
                script_MAC_length,
                script_MAC,
                new_PIN_reference_value_length,
                new_PIN_reference_value,
                new_PRW_random_number_length,
                new_PRW_random_number,
                output_encrypted_PIN_block_length,
                output_encrypted_PIN_block,
                PIN_block_MAC_length,
                PIN_block_MAC)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0 - 7.

rule_array

Direction	Type
Input	Character

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 378. Rule array keywords for the DK PIN Change Service

Keyword	Meaning
PIN Block output selection keyword (One, optional)	
NOEPB	Do not return an encrypted PIN block (EPB). This is the default value.
EPB	Return an encrypted PIN block and a MAC to verify the encrypted PIN block.
Script selection algorithm and method keyword (One, optional)	
AES-CBC	Specifies to use CBC mode to AES encrypt the script. If SCR2020 is specified, AES-CBC specifies to AES encrypt the PIN block plus additional data in the script.
NOSCRIPT	Do not return an encrypted SMPIN message with a MAC. This is the default value.
TDES-CBC	Specifies to use CBC mode to TDES encrypt the script. If SCR2020 is specified, TDES-CBC specifies to TDES CBC encrypt the PIN block plus additional data in the script.
TDES-ECB	Specifies to use ECB mode to TDES encrypt the script. If SCR2020 is specified, TDES-ECB specifies to TDES ECB encrypt the PIN block plus additional data in the script.
Script Process (One, optional). May be specified with keyword AES-CBC, TDES-CBC, or TDES-ECB. Otherwise, invalid.	
SCR2013	Specifies to use script processing rules introduced with the service in 2013. This is the default.
SCR2020	Specifies to use the script processing rules introduced with this service in 2020. The new process encrypts only the new PIN block and any additional data in the *card_script_data* field parameter and returns only the encrypted portion of the *card_script_data* field in the output script parameter. This keyword is introduced in APAR OA58880 for ICSF FMID HCR77D1.
Pin encryption keyword (One, optional) Only valid if AES-CBC, TDES-CBC or TDES-ECB is selected above.	

Table 378. Rule array keywords for the DK PIN Change Service (continued)	
Keyword	**Meaning**
CLEARPIN	Do not encrypt the PIN prior to inserting in the script block. This is the default value.
SELF-ENC	Copy the PIN-block self-encrypted to the clear PIN block within the clear output message. Use this rule array keyword to specify that the 8-byte PIN block shall be used as a DES key to encrypt the PIN block. The service copies the self-encrypted PIN block to the clear PIN block in the output message.
MAC Ciphering Method (One required for AES-CBC, one optional for TDES-CBC or TDES-ECB; otherwise, not allowed.)	
CMAC	Specifies to use the cipher-based MAC algorithm block cipher mode of operation for authentication, recommended in NIST SP 800-38B. Required for AES-CBC. Only valid with AES-CBC.
EMVMACD	Specifies the EMV-related message-padding and calculation method. Not valid with AES-CBC.
TDES-MAC	Specifies the ANS X9.9 Option 1 (binary data) procedure and a CBC Triple-DES encryption of the data. Not valid with AES-CBC.
X9.19OPT	Specifies the ANS X9.19 Optional Procedure. A double-length key is required. Not valid with AES-CBC. This is the default value for TDES-CBC and TDES-ECB.
MAC Length and presentation (One, optional) Only valid if AES-CBC, TDES-CBC, or TDES-ECB is selected above.	
MACLEN8	Specifies a 8-byte MAC. This is the default value for TDES-CBC and TDES-ECB.
MACLEN16	Specifies a 16-byte MAC. Only valid with CMAC. This is the default for AES-CBC.
PIN Block format (One, optional)	
ISO-1	Specifies that the encrypted PIN block in the *ISO_encrypted_PIN_block* parameter is in the ISO-1 format. This is the default.
ISO-4	Specifies that the encrypted PIN block in the *ISO_encrypted_PIN_block* parameter is in the ISO-4 format.

PAN_data_length

Direction	**Type**
Input	Integer

Specifies the length in bytes of the *PAN_data* parameter. The value must be between 10 and 19, inclusive.

PAN_data

Direction	**Type**
Input	Character

The PAN data which the PIN is associated. The full account number, including check digit, should be included. This parameter is character data.

card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_p_data* parameter. The value must be between 2 and 256, inclusive.

card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_t_data* parameter. The value must be between 2 and 256, inclusive.

card_t_data

Direction	Type
Input	String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the card_p_data, specifies an individual card.

cur_ISO_PIN_block_length

Direction	Type
Input	Integer

The *cur_ISO_pin_block_length* specifies the length in bytes of the *cur_ISO_PIN_block* parameter. This value must be 8 when ISO-1 is specified and 16 when ISO-4 is specified.

cur_ISO_PIN_block

Direction	Type
Input	String

The encrypted PIN block with the current PIN in ISO-1 or ISO-4 format.

new_ISO_PIN_block_length

Direction	Type
Input	Integer

The *new_ISO_pin_block_length* specifies the length in bytes of the *new_ISO_PIN_block* parameter. This value must be 8 when ISO-1 is specified and 16 when ISO-4 is specified.

new_ISO_PIN_block

Direction	Type
Input	String

The new encrypted PIN block with the customer chosen PIN. The PIN block must be in ISO-1 or ISO-4 format.

card_script_data_length

Direction	Type
Input	Integer

The number of bytes of data in the *card_script_data* parameter. If the script selection of the rule array specifies to not return an encrypted SMPIN message with a PIN block MAC (that is, AES-CBC, TDES-CBC, or TDES- ECB is not specified), the value must be 0.

When the script selection keyword is TDES-CBC or TDES-ECB, the value is a positive value less than or equal to 4096 and a multiple of 8. When the keyword is AES-CBC, the value is a positive value less than or equal to 4096.

card_script_data

Direction	Type
Input	String

The cleartext data to be MAC'd. The *script_offset* value can be considered the PIN block offset. If the SCR2013 keyword or no script process rule is specified, the entire field is encrypted and returned in the script parameter. If the SCR2020 keyword is specified, *script_length* bytes are encrypted starting at the offset indicated by the *script_offset* parameter. The PIN block plus additional data are encrypted and inserted at the offset specified by the *script_offset* parameter the MAC operation is performed. The smaller encrypted result is returned in the *script* parameter.

script_offset

Direction	Type
Input	Integer

The offset in bytes from the beginning of the cleartext data in the *card_script_data* variable to the location for the clear PIN block. The first byte of the cleartext data is offset 0. If the SCR2013 keyword or no script process rule is specified, this offset plus the *script_offset_field_length* must be less than or equal to the *card_script_data_length*. If the SCR2020 keyword is specified, the value of the *script_offset* plus the *script_length* must be less than or equal to the *card_script_data_length*.

script_offset_field_length

Direction	Type
Input	Integer

The number of bytes of data in the PIN block format referenced by the output PIN profile. Currently, this length must be 8. The PIN block must fit entirely within the *card_script_data* parameter. If NOSCRIPT is specified in the rule array, this parameter is ignored.

script_initialization_vector_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *script_initialization_vector* parameter. For script selection algorithm and method keyword AES-CBC, the value must be 16. For TDES-CBC, the value must be 8. Otherwise, the value must be 0.

script_initialization_vector

Direction	Type
Input	String

The 8-byte or 16 byte initialization data for encrypting the script. The value of this parameter must be a string of hexadecimal zeroes. If the *script_initialization_vector_length* is 0, this parameter is ignored.

output_PIN_profile

Direction	Type
Input	String

A 24-byte string containing the PIN profile, including the PIN block format for the script. See "The PIN profile" on page 609 for additional information. You can use PIN-block formats ISO-0, ISO-1, ISO-2, ISO-3, and ISO-4 with this service. If NOSCRIPT is specified in the rule array, this parameter is ignored.

PIN_reference_value_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. This value must be 16.

PIN_reference_value

Direction	Type
Input	String

The 16-byte PIN reference value of the current PIN for comparison to the calculated value.

PRW_random_number_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be 4.

PRW_random_number

Direction	Type
Input	String

The 4-byte random number associated with the PIN reference value of the current PIN.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify the PRW of the current PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

cur_IPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *cur_IPIN_encryption_key_identifier* parameter. If the *cur_IPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

cur_IPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the PIN_block containing the current PIN. The key identifier is an operational token or the key label of an operational token in key storage.

When ISO-1 is specified, the key algorithm of this key must be DES and the key type must be IPINENC. The control vector must enable the verification of an encrypted PIN (EPINVER bit 19 = B'1').

When ISO-4 is specified, the key algorithm of this key must be AES and the key type must be PINPROT. The key usage fields must specify DECRYPT, CBC, NOFLDFMT, ISO-4, and PINXLATE. To use the same PINPROT key for the new and current inbound IPIN encryption key, the key usage fields must specify DECRYPT, CBC, NOFLDFMT, ISO-4, PINXLATE, and EPINVER.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

new_IPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_IPIN_encryption_key_identifier* parameter. If the *new_IPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

new_IPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the PIN_block containing the new PIN. The key identifier is an operational token or the key label of an operational token in key storage.

When ISO-1 is specified, the key algorithm of this key must be DES and the key type must be IPINENC. The control vector must enable for translation (TRANSLAT bit 22 = B'1'). To use the same

key token for the current and the new inbound PIN encryption key, the control vector must have key usages TRANSLAT and EPINVER enabled.

When ISO-4 is specified, the key algorithm of this key must be AES and the key type must be PINPROT. The key usage fields must specify DECRYPT, CBC, and PINXLATE. To use the same key token for the current and the new inbound PIN encryption key, the key usage EPINVER, and PINXLATE must be enabled.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

script_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *script_key_identifier* parameter. If the rule array indicates that no script is to be processed, this value must be 0. If the *script_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

script_key_identifier

Direction	Type
Input/Output	String

The identifier of the key for encryption of the script. The key identifier is an operational token or the key label of an operational token in key storage. For script selection algorithm and method keyword AES-CBC, the key algorithm of the key must be AES, the key type must be SECMSG, and the key usage fields must indicate SMPIN and allow use by the CSNBDPC service (ANY-USE or DPC-ONLY). For keywords TDES-CBC or TDES-ECB, the key algorithm of this key must be DES, the key type must be SECMSG with the SMPIN usage bit (CV bit 19) set to B'1'.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

script_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *script_MAC_key_identifier* parameter. If the rule array indicates that no script is to be processed, this value must be 0. If the *script_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

script_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC of the script. The key identifier is an operational token or the key label of an operational token in key storage. For script selection algorithm and method keyword AES-CBC, the key algorithm of the key must be AES, the key type must be MAC, and the key usage fields must indicate GENERATE or GENONLY and CMAC. For keywords TDES-CBC or TDES-ECB, the key algorithm of this key must be DES, the key type must be MAC, and the key must be double-length.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

new_PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_PRW_key_identifier* parameter. If the *new_PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

new_PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify the new PRW. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_encryption_key_identifier* parameter. If the rule array indicates that no encrypted PIN block is to be returned, this value must be 0. If the *OPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to encrypt the new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the *OPIN_encryption_key_identifier_length* is 0, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

OEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OEPB_MAC_key_identifier* parameter. If the rule array indicates that no encrypted PIN block MAC is to be returned, this value must be 0. If the *OEPB_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC of new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the *OEPB_MAC_key_identifier_length* is 0, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, GENONLY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

script_length

Direction	Type
Input/Output	Integer

The number of bytes of data in the script variable. The value must be 0 if the script selection algorithm and method of the rule array specifies NOSCRIPT. For scripting, if the SCR2020 keyword is specified, the value must be set to the PIN block length plus the length of the additional customer defined data. Otherwise, the value of the *script_offset* plus the *script_length* must be less than or equal to the *card_script_data_length*.

script

Direction	Type
Output	String

The script returned. If the rule array specifies to return a script, *script_length* bytes of this variable are overwritten. If SCR2020 is specified, the parameter contains the encrypted part of the script. Otherwise, it contains the entire script.

script_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *script_MAC* parameter. If the NOSCRIPT keyword is selected, this value must be 0.

When the MAC length keyword is MACLEN8 or MACLEN16, the value must be at least as large as indicated by the keyword specified. When no MAC length keyword is specified and the script selection keyword is AES-CBC, the value must be between 4 and 16 inclusive. On output, the parameter is updated with the actual length of the *script_MAC* parameter.

script_MAC

Direction	Type
Output	String

The 8 byte or 16 byte MAC of the encrypted script. If the *script_MAC_length* is 0, this parameter is ignored.

new_PIN_reference_value_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_PIN_reference_value* parameter. The value must be at least 16. On output, it will be set to 16.

new_PIN_reference_value

Direction	Type
Output	String

The 16-byte new PIN reference value of the new PIN block.

new_PRW_random_number_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_PRW_random_number* parameter. The value must be at least 4. On output, it will be set to 4.

new_PRW_random_number

Direction	Type
Output	String

The 4-byte random number associated with the new PIN reference value.

output_encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *output_encrypted_PIN_block* parameter. If the rule array indicates that no encrypted PIN block should be returned, this value must be 0. Otherwise, it should be at least 32. On output it will be set to 32.

output_encrypted_PIN_block

Direction	Type
Output	String

The 32-byte encrypted new PIN block. If the *output_encrypted_PIN_block_length* is 0, this parameter is ignored.

PIN_block_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_block_MAC* parameter. If the rule_array indicates that no PIN block MAC should be returned, this value must be 0. Otherwise, it must be at least 8.

PIN_block_MAC

Direction	Type
Output	String

The 8-byte MAC of the new encrypted PIN block. If the *PIN_block_MAC_length* is 0, this parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

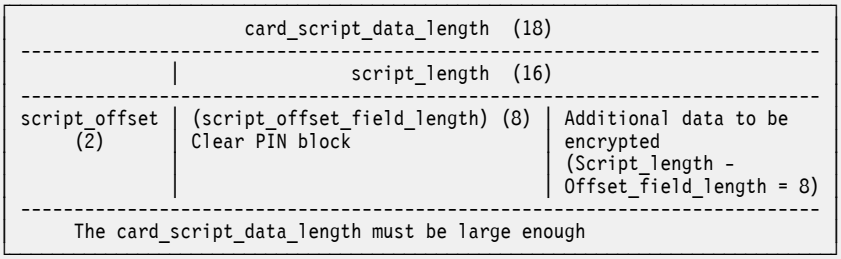
Notes on script processing

When using the **SCR2020** keyword, the script, whose length is represented by *script_length*, is now only a part of the *card_script_data* area, whose length is represented by *card_script_data_length*. This script length will indicate how much of the *card_script_data* area is to be encrypted. Therefore, the *script_offset* + the *script_length* must not be less than the *card_script_data_length*. If it is, a size error is returned.

Note: If the *script_offset* = 0, the *card_script_data_length* and the *script_length* could be equal.

Here is an example showing the correct layout with sample input lengths:

- *card_script_data_length* = 18
- *script_offset* = 2
- *script_offset_field_length* = 8
- *script_length* = 16



Note: It is not possible with SCR2020 to encrypt data **before** the clear PIN block. This is a difference from SCR2013. SCR2020 is mandatory for ISO-4 PIN blocks. Therefore, this restriction applies to all scripts that handle ISO-4 PINs.

Access control points

The **DK PIN Change** access control point in the domain role controls the function of this service.

When the **General ISO PIN Error Mode** access control is enabled, the return code will be a general PIN block error (return code 8 reason code 2514) instead of some of the PIN block errors return code. The use of a general return code can prevent the abuse of PIN processing error messages due to information leakage derived from the return code reason codes returned under various conditions. For more details, see “PIN block error processing mode” on page 609.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format *rule_array* keyword ISO-1 is not allowed.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 379. DK PIN Change required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Rule array keywords AES-CBC, CMAC, and MACLEN16 require the July 2015 or later licensed internal code (LIC). Keywords ISO-1 and ISO-4 and triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The combination of keyword ISO-4 and any keyword form the script selection algorithm and method keyword group requires the June 2020 or later licensed internal code (LIC). Keywords SCR2013 and SCR2020 require the June 2020 or later licensed internal code (LIC).
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Rule array keywords AES-CBC, CMAC, and MACLEN16 require the July 2015 or later licensed internal code (LIC). Keywords ISO-1 and ISO-4 and triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. The combination of keyword ISO-4 and any keyword form the script selection algorithm and method keyword group requires the June 2020 or later licensed internal code (LIC). Keywords SCR2013 and SCR2020 require the June 2020 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	Keywords ISO-1 and ISO-4 and triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The combination of keyword ISO-4 and any keyword form the script selection algorithm and method keyword group requires the June 2020 or later licensed internal code (LIC). Keywords SCR2013 and SCR2020 require the June 2020 or later licensed internal code (LIC).

Table 379. DK PIN Change required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. The combination of keyword ISO-4 and any keyword form the script selection algorithm and method keyword group requires the June 2020 or later licensed internal code (LIC). Keywords SCR2013 and SCR2020 require the June 2020 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	The combination of keyword ISO-4 and any keyword form the script selection algorithm and method keyword group requires the June 2020 or later licensed internal code (LIC). Keywords SCR2013 and SCR2020 require the June 2020 or later licensed internal code (LIC).
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK PIN Verify (CSNBDPV and CSNEDPV)

Use the DK PIN Verify callable service to verify an ISO-1 or ISO-4 format PIN. The input PIN will be converted to PBF-0 format. A test PIN reference value (PRW) is created and that value is bitwise compared to the input PRW.

The callable service name for AMODE(64) invocation is CSNEDPV.

Format

```
CALL CSNBDPV(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             PAN_data_length,
             PAN_data,
             card_data_length,
             card_data,
             PIN_reference_value_length,
             PIN_reference_value,
             PRW_random_number_length,
             PRW_random_number,
             ISO_encrypted_PIN_block_length,
             ISO_encrypted_PIN_block,
             PRW_key_identifier_length,
             PRW_key_identifier,
             IPIN_encryption_key_identifier_length,
             IPIN_encryption_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value can be 0 or 1.

rule_array

Direction	Type
Input	Character

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 380. Keywords for the DK PIN Verify Service

Keyword	Meaning
PIN Block format (One, optional)	
ISO-1	Specifies that the encrypted PIN block in the *ISO_encrypted_PIN_block* parameter is in the ISO-1 format. This is the default.

Table 380. Keywords for the DK PIN Verify Service (continued)	
Keyword	**Meaning**
ISO-4	Specifies that the encrypted PIN block in the *ISO_encrypted_PIN_block* parameter is in the ISO-4 format.

PAN_data_length

Direction	**Type**
Input	Integer

Specifies the length in bytes of the *PAN_data* parameter. The value must be between 10 and 19, inclusive.

PAN_data

Direction	**Type**
Input	Character

The PAN data which the PIN is associated. The full account number, including check digit, should be included. This parameter is character data.

card_data_length

Direction	**Type**
Input	Integer

Specifies the length in bytes of the *card_data* parameter. The value must be between 4 and 512, inclusive.

card_data

Direction	**Type**
Input	String

The time-invariant card data (CDp) and the time-sensitive card data (CDt) which, together with the account number, specifies an individual card.

PIN_reference_value_length

Direction	**Type**
Input	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. This value must be 16.

PIN_reference_value

Direction	**Type**
Input	String

The 16-byte PIN reference value for comparison to the calculated value.

PRW_random_number_length

Direction	**Type**
Input	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be 4.

PRW_random_number

Direction	Type
Input	String

The 4-byte random number associated with the PIN reference value.

ISO_encrypted_PIN_block_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *ISO_encrypted_PIN_block* parameter. This value must be 8 for a ISO-1 block and 16 for a ISO-4 block.

ISO_encrypted_PIN_block

Direction	Type
Input	String

The 8-byte encrypted PIN block in ISO-1 format.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify the PIN reference value. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

IPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IPIN_encryption_key_identifier* parameter. If the *IPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

IPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the PIN_block. The key identifier is an operational token or the key label of an operational token in key storage.

When ISO-1 is specified, the key algorithm of this key must be DES and the key type must be IPINENC. The control vector must enable the verification of an encrypted PIN (EPINVER bit 19 = B'1').

When ISO-4 is specified, the key algorithm of this key must be AES and the key type must be PINPROT. The key usage fields must specify DECRYPT, CBC, NOFLDFMT, and EPINVER.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK PIN Verify** access control point in the domain role controls the function of this service.

When the **General ISO PIN Error Mode** access control is enabled, the return code will be a general PIN block error (return code 8 reason code 2514) instead of some of the PIN block errors return code. The use of a general return code can prevent the abuse of PIN processing error messages due to information leakage derived from the return code reason codes returned under various conditions. For more details, see "PIN block error processing mode" on page 609.

When the **Disallow PIN block format ISO-1** access control is enabled in the domain role, the PIN block format *rule_array* keyword ISO-1 is not allowed.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 381. DK PIN Verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords ISO-1 and ISO-4 and triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Keywords ISO-1 and ISO-4 and triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Keywords ISO-1 and ISO-4 and triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).

Table 381. DK PIN Verify required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK PRW Card Number Update (CSNBDPNU and CSNEDPNU)

Use the DK PRW Card Number Update callable service to generate a PIN reference value (PRW) when a replacement card is being issued. The original PIN and primary account number are used with new time-sensitive card data to generate the new PRW.

You can use this service to perform the following tasks:

- Generate a PRW that can be used to verify the PIN.
- Optionally, generate an encrypted PIN block in PBF-1 format to be stored for later use in personalizing replacement cards.

The callable service name for AMODE(64) invocation is CSNEDPNU.

Note: Beginning with APAR OA57089 and new licensed internal code for IBM z13, IBM z13s, and later servers, the DK PRW Card Number Update (CSNBDPNU and CSNEDPNU) service has been deprecated. New applications should use DK PRW Card Number Update2 instead of DK PRW Card Number Update. See “DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2)” on page 904.

Format

```
CALL CSNBDPNU(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              card_p_data_length,
              card_p_data,
              card_t_data_length,
              card_t_data,
              encrypted_PIN_block_length,
              encrypted_PIN_block,
              PIN_block_MAC_length,
              PIN_block_MAC,
              PRW_key_identifier_length,
              PRW_key_identifier,
              IPIN_encryption_key_identifier_length,
              IPIN_encryption_key_identifier,
              IEPB_MAC_key_identifier_length,
              IEPB_MAC_key_identifier,
              OPIN_encryption_key_identifier_length,
              OPIN_encryption_key_identifier,
              OEPB_MAC_key_identifier_length,
              OEPB_MAC_key_identifier,
```

```
                PIN_reference_value_length,
                PIN_reference_value,
                PRW_random_number_length,
                PRW_random_number,
                new_encrypted_PIN_block_length,
                new_encrypted_PIN_block,
                new_PIN_block_MAC_length,
                new_PIN_block_MAC)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0 or 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 382. Keywords for the DK PRW Card Number Update service	
Keyword	**Meaning**
PIN Block output selection keyword (One, optional)	
NOEPB	Do not return an encrypted PIN block (EPB). This is the default.
EPB	Return an encrypted PIN block.

card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_p_data* parameter. The value must be between 2 and 256, inclusive.

card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_t_data* parameter. The value must be between 2 and 256, inclusive.

card_t_data

Direction	Type
Input	String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the *card_p_data*, specifies an individual card.

encrypted_PIN_block_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *encrypted_PIN_block* parameter. The value must be 32.

encrypted_PIN_block

Direction	Type
Input	String

The 32-byte input encrypted PIN block in PBF-1 format.

PIN_block_MAC_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_block_MAC* parameter. The value must be 8.

PIN_block_MAC

Direction	Type
Input	String

The 8-byte CMAC of the encrypted PIN block.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the PRW generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IPIN_encryption_key_identifier* parameter. If the *IPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

IPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key that encrypts the input PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate DECRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IEPB_MAC_key_identifier* parameter. If the *IEPB_MAC_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 725.

IEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the CMAC verification key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, VERIFY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_encryption_key_identifier* parameter. If no encrypted PIN block is to be returned, this value must be 0. If the *OPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If no encrypted PIN block is to be returned, this value is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OEPB_MAC_key_identifier* parameter. If the rule array indicates that no encrypted PIN block MAC is to be returned, this value must be 0. If the *OEPB_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the CMAC of the new PRW. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block MAC is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

PIN_reference_value_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. This value must be 16. On output, it will be set to 16.

PIN_reference_value

Direction	Type
Output	String

The calculated 16-byte PIN reference value.

PRW_random_number_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be 4. On output, it will be set to 4.

PRW_random_number

Direction	Type
Output	String

The 4-byte random number associated with the PIN reference value.

new_encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_encrypted_PIN_block* parameter. If the rule array indicates that no new encrypted PIN block should be returned, this parameter must be zero. Otherwise, the parameter should be at least 32.

new_encrypted_PIN_block

Direction	Type
Output	String

The new 32-byte encrypted PIN block. If the rule array indicates that no new encrypted PIN block should be returned, this parameter is ignored.

new_PIN_block_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_PIN_block_MAC* parameter. If the *rule_array* indicates that no new_PIN_block_MAC should be returned, this value must be zero. Otherwise, it must be at least 8.

new_PIN_block_MAC

Direction	Type
Output	String

The new 8-byte encrypted MAC of the new PIN block. If the rule array indicates that no new encrypted PIN block should be returned, this parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK PRW Card Number Update** access control point in the domain role controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 383. DK PRW Card Number Update required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	

Table 383. DK PRW Card Number Update required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2)

Use the DK PRW Card Number Update2 callable services to generate PIN reference value (PRW) with new time-sensitive card data, but without changing either the customer PIN or the primary account number.

You can use this service to perform the following tasks:

- Return an updated PIN reference value and associated new PRW random number value to be used as input by other PIN processes to verify the PIN.
- Optionally, generate an encrypted PIN block in DK-defined PBF-1 format to be stored for later use in personalizing replacement cards, along with a verifying CMAC over the encrypted block and additional card data.
- Optionally, return a PIN block in the *new_chip_encrypted_PIN_block* parameter encrypted using an AES PINPROT key.
- Optionally, test the PAN within the clear DK-defined PBF-0 PIN block against the supplied *PAN_data* and determine if they are different.

The callable service name for AMODE(64) invocation is CSNEDCU2.

Format

```
CALL CSNBDCU2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              card_p_data_length,
              card_p_data,
              card_t_data_length,
              card_t_data,
              encrypted_PIN_block_length,
              encrypted_PIN_block,
              PIN_block_MAC_length,
              PIN_block_MAC,
              PRW_key_identifier_length,
              PRW_key_identifier,
              IPIN_encryption_key_identifier_length,
              IPIN_encryption_key_identifier,
              IEPB_MAC_key_identifier_length,
              IEPB_MAC_key_identifier,
              OPIN_encryption_key_identifier_length,
              OPIN_encryption_key_identifier,
              OEPB_MAC_key_identifier_length,
              OEPB_MAC_key_identifier,,
              OPIN_chip_encryption_key_identifier_length,
              OPIN_chip_encryption_key_identifier,
              PAN_data_length,
              PAN_data,
              PIN_reference_value_length,
              PIN_reference_value,
              PRW_random_number_length,
              PRW_random_number,
              new_encrypted_PIN_block_length,
```

```
                new_encrypted_PIN_block,
                new_PIN_block_MAC_length,
                new_PIN_block_MAC,
                new_chip_encrypted_PIN_block_length,
                new_chip_encrypted_PIN_block)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0 - 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 384. Keywords for the DK PRW Card Number Update2 service	
Keyword	**Meaning**
PIN Block output selection keyword (One or two, optional)	
NOEPB	Do not return an encrypted PIN block (EPB). This is the default. Cannot be combined with EBP or CHIP-EPB.
EPB	Return an encrypted PIN block. Cannot be combined with NOEBP.
CHIP-EPB	Return a chip encrypted PIN block. Cannot be combined with NOEPB.
PAN test selection keyword (one, optional)	
NOPANTST	Specifies to not perform a test to determine if the input PAN provided is different from the PAN within the clear PBF-0 PIN block. This is the default.
PANTST	Specifies to perform a test to determine if the input PAN provided is different from the PAN within the clear PBF-0 PIN block.

card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_p_data* parameter. The value must be between 2 and 256, inclusive.

card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_t_data* parameter. The value must be between 2 and 256, inclusive.

card_t_data

Direction	Type
Input	String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the *card_p_data*, specifies an individual card.

encrypted_PIN_block_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *encrypted_PIN_block* parameter. The value must be 32.

encrypted_PIN_block

Direction	Type
Input	String

The 32-byte input encrypted PIN block in PBF-1 format.

PIN_block_MAC_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_block_MAC* parameter. The value must be 8.

PIN_block_MAC

Direction	Type
Input	String

The 8-byte CMAC of the encrypted PIN block.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the PRW generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IPIN_encryption_key_identifier* parameter. If the *IPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

IPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key that encrypts the input PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate DECRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IEPB_MAC_key_identifier* parameter. If the *IEPB_MAC_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 725.

IEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify the MAC of the input PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, VERIFY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_encryption_key_identifier* parameter. When the keyword EPB in the rule array is not specified, this value must be 0. If the *OPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. When the *OPIN_encryption_key_identifier_length* is 0, this value is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OEPB_MAC_key_identifier* parameter. When the keyword EPB in the rule array is not specified, this value must be 0. If the *OEPB_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC of the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. When the *OEPB_MAC_key_identifier_length* is 0, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, GENONLY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OPIN_chip_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_chip_encryption_key_identifier* parameter. When the keyword CHIP-EPB in the rule array is not specified, this value must be 0. If the *OPIN_chip_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_chip_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to encrypt the optional PIN block for the personalization unit. The key identifier is an operational token or the key label of an operational token in key storage. When the *OPIN_chip_encryption_key_identifier_length* is 0, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PAN_data* parameter. The value must be 10 – 19 when the PANTST keyword is specified. Otherwise, the value must be 0.

PAN_data

Direction	Type
Input	String

The primary account number (PAN) data used to generate PIN. Include the full account number, including the check digit. When the *PAN_data_length* value is 0, this parameter is ignored.

PIN_reference_value_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. This value must be 16. On output, it will be set to 16.

PIN_reference_value

Direction	Type
Output	String

The calculated 16-byte PIN reference value.

PRW_random_number_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be 4. On output, it will be set to 4.

PRW_random_number

Direction	Type
Output	String

The 4-byte random number associated with the PIN reference value.

new_encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_encrypted_PIN_block* parameter. If the keyword EPB is not specified in the rule array, this parameter must be zero. Otherwise, the parameter should be at least 32.

new_encrypted_PIN_block

Direction	Type
Output	String

The 32-byte encrypted PIN block in PBF-1 format. If the *new_encrypted_PIN_block_length* value is zero, this parameter is ignored.

new_PIN_block_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_PIN_block_MAC* parameter. If the keyword EPB is not specified in the rule array, this value must be zero. Otherwise, it must be at least 8.

new_PIN_block_MAC

Direction	Type
Output	String

The 8-byte CMAC of the encrypted PIN block. If the *new_PIN_block_MAC_length* value is zero, this parameter is ignored.

new_chip_encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_chip_encrypted_PIN_block* parameter. If the keyword CHIP-EPB is not specified in the rule array, the value to 0. Otherwise, on input the value must be at least 32.

new_chip_encrypted_PIN_block

Direction	Type
Output	String

The 32-byte chip encrypted PIN block. If the *new_chip_encrypted_PIN_block_length* value is zero, this parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK PRW Card Number Update2** access control point in the domain role controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 385. DK PRW Card Number Update2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	This service requires the July 2019 or later licensed internal code. Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	This service requires the July 2019 or later licensed internal code. Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	This service requires the July 2019 or later licensed internal code.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	

Table 385. DK PRW Card Number Update2 required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG)

Use the DK PRW CMAC Generate callable service to generate a message authentication code (MAC) over specific values involved in an account number change transaction. The inputs include the current and new PAN and card data and the PIN reference value.

The output of this service is used as input to the DK PAN Modify in Transaction callable service, which will create the new PIN reference value (PRW) to be used to verify the PIN.

The callable service name for AMODE(64) invocation is CSNEDPCG.

Format

```
CALL CSNBDPCG(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              current_PAN_data_length,
              current_PAN_data,
              new_PAN_data_length,
              new_PAN_data,
              current_card_data_length,
              current_card_data,
              new_card_data_length,
              new_card_data,
              PIN_reference_value_length,
              PIN_reference_value,
              CMAC_FUS_key_identifier_length,
              CMAC_FUS_key_identifier,
              CMAC_FUS_length,
              CMAC_FUS)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. There are no keywords for this service.

current_PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *current_PAN_data* parameter. The value must be between 10 and 19, inclusive.

current_PAN_data

Direction	Type
Input	String

The current PAN data. The full account number, including check digit, should be included.

new_PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_PAN_data* parameter. The value must be between 10 and 19, inclusive.

new_PAN_data

Direction	Type
Input	String

The new PAN data. The full account number, including check digit, should be included.

current_card_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *current_card_data* parameter. The value must be between 4 and 512, inclusive.

current_card_data

Direction	Type
Input	String

The current card data, determined by the card issuer.

new_card_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_card_data* parameter. The value must be between 4 and 512, inclusive.

new_card_data

Direction	Type
Input	String

The new card data, determined by the card issuer.

PIN_reference_value_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. The value must be 16.

PIN_reference_value

Direction	Type
Input	String

The 16-byte PIN reference value of the current PIN.

CMAC_FUS_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *CMAC_FUS_key_identifier* parameter. If the *CMAC_FUS_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

CMAC_FUS_key_identifier

Direction	Type
Input	String

The identifier of the key to generate the MAC. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate GENONLY, CMAC, and DKPINAD2.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

CMAC_FUS_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *CMAC_FUS* parameter. The value must be between 8 and 16, inclusive.

CMAC_FUS

Direction	Type
Output	String

The MAC of the current and new PANs and card data strings.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK PRW CMAC Generate** access control point in the domain role controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 386. DK PRW CMAC Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).

Table 386. DK PRW CMAC Generate required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK Random PIN Generate (CSNBDRPG and CSNEDRPG)

Use the DK Random PIN Generate callable service to generate a PIN and a PIN reference value using the random process. After the PIN is generated, a PIN reference value (PRW) is created. The PIN reference value is used to verify the PIN in other processes.

Note: Regarding weak PINs, if the PIN which is generated appears in the weak PIN table, the generation process is modified and re-tried until a valid PIN is generated.

You can use this service to perform the following tasks:

- Generate an encrypted PIN block in PBF-1 format with a PIN print key to be printed on a PIN mailer.
- Generate a PIN reference value which can be used to verify the PIN.
- Optionally, generate an encrypted PIN block in PBF-1 format to be stored for later use in personalizing replacement cards, along with a verifying CMAC over the encrypted block and additional card data.

The callable service name for AMODE(64) invocation is CSNEDRPG.

Note: Beginning with APAR OA57089 and new licensed internal code for IBM z13, IBM z13s, and later servers, the DK Random PIN Generate (CSNBDRPG and CSNEDRPG) service has been deprecated. New applications should use DK Random PIN Generate2 instead of DK Random PIN Generate. See “DK Random PIN Generate2 (CSNBDRG2 and CSNEDRG2)” on page 922.

Format

```
CALL CSNBDRPG(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              PAN_data_length,
              PAN_data,
              card_p_data_length,
              card_p_data,
              card_t_data_length,
              card_t_data,
              PIN_length,
              PRW_key_identifier_length,
              PRW_key_identifier,
              PIN_print_key_identifier_length,
              PIN_print_key_identifier,
```

```
                OPIN_encryption_key_identifier_length,
                OPIN_encryption_key_identifier,
                OEPB_MAC_key_identifier_length,
                OEPB_MAC_key_identifier,
                PIN_reference_value_length,
                PIN_reference_value,
                PRW_random_number_length,
                PRW_random_number,
                PIN_print_block_length,
                PIN_print_block,
                encrypted_PIN_block_length,
                encrypted_PIN_block,
                PIN_block_MAC_length,
                PIN_block_MAC)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0 or 1.

rule_array

Direction	Type
Input	Character

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 387. Rule array keywords for DK Random PIN Generate with Reference Value Service

Keyword	Meaning
PIN Block output selection keyword (One, optional)	
NOEPB	Do not return an encrypted PIN block (EPB). This is the default value.
EPB	Return an encrypted PIN block.

PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PAN_data* parameter. The value must be between 10 and 19, inclusive.

PAN_data

Direction	Type
Input	Character

The PAN data which the PIN is associated. The full account number, including check digit, should be included. This parameter is character data.

card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_p_data* parameter. The value must be between 2 and 256, inclusive.

card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_t_data* parameter. The value must be between 2 and 256, inclusive.

card_t_data

Direction	Type
Input	String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the card_p_data, specifies an individual card.

PIN_length

Direction	Type
Input	Integer

Specifies the length of the PIN to be generated. This value must be between 4 and 12, inclusive.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to calculate the PRW for the PIN. The key identifier is an operational token or the key label of an operational token. The key algorithm of this key must be AES, the key type must be PINPRW, the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

PIN_print_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_print_key_identifier* parameter. If the *PIN_print_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 725.

PIN_print_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the PIN for printing. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOPP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_encryption_key_identifier* parameter. If the rule array indicates that no encrypted PIN block is to be returned, this value must be 0. If the

OPIN_encryption_key_identifier contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

OEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OEPB_MAC_key_identifier* parameter. If the rule array indicates that no encrypted PIN block MAC is to be returned, this value must be 0. If the *OEPB_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC of the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

PIN_reference_value_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. The value must be at least 16. On output, it will be set to 16.

PIN_reference_value

Direction	Type
Output	String

The 16-byte calculated PIN reference value.

PRW_random_number_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be at least 4. On output, it will be set to 4.

PRW_random_number

Direction	Type
Output	String

The 4-byte random number associated with the PIN reference value.

PIN_print_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_print_block* parameter. It must be at least 32. On output, it will be set to 32.

PIN_print_block

Direction	Type
Output	String

The 32-byte encrypted PIN block to be passed to the PIN mailer function.

encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *encrypted_PIN_block* parameter. If the rule array indicates that no encrypted PIN block should be returned, this value must be 0. Otherwise, it should be at least 32.

encrypted_PIN_block

Direction	Type
Output	String

The 32-byte encrypted PIN block PBF-1 format. This parameter is ignored if no encrypted PIN block is returned.

PIN_block_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_block_MAC* parameter. If the rule_array indicates that no PIN block MAC should be returned, this value must be 0. Otherwise, it must be at least 8.

PIN_block_MAC

Direction	Type
Output	String

The 8-byte CMAC of the encrypted PIN block. This parameter is ignored if no encrypted PIN block is returned.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK Random PIN Generate** access control point in the domain role controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 388. DK Random PIN Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK Random PIN Generate2 (CSNBDRG2 and CSNEDRG2)

Use the DK Random PIN Generate2 service to generate a random PIN of a specified length and calculate a PIN reference value (PRW).

Weak PINs: The random PIN generation process generates a new PIN until the generated PIN is not found in the weak PIN table.

This service performs the following tasks:

- Generates a random PIN of the selected length and creates a PIN block in a DK-defined format that is used to return a PIN print key to be printed on a PIN mailer.
- Generates a PRW random number and calculates a PIN reference value. These values are returned for later use in other PIN processes to verify the PIN.

- Optionally returns a verifying PIN block MAC calculated over the PIN block and additional card data using CMAC, together with an encrypted PIN block. This information can be stored for later use in personalizing replacement cards.
- Optionally returns a PIN block in the *chip_encrypted_PIN_block* parameter encrypted using an AES PINPROT key.

The callable service name for AMODE(64) invocation is CSNEDRG2.

Format

```
CALL CSNBDRG2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              PAN_data_length,
              PAN_data,
              card_p_data_length,
              card_p_data,
              card_t_data_length,
              card_t_data,
              PIN_length,
              PRW_key_identifier_length,
              PRW_key_identifier,
              PIN_print_key_identifier_length,
              PIN_print_key_identifier,
              OPIN_encryption_key_identifier_length,
              OPIN_encryption_key_identifier,
              OEPB_MAC_key_identifier_length,
              OEPB_MAC_key_identifier,
              OPIN_chip_encryption_key_identifier_length,
              OPIN_chip_encryption_key_identifier,
              PIN_reference_value_length,
              PIN_reference_value,
              PRW_random_number_length,
              PRW_random_number,
              PIN_print_block_length,
              PIN_print_block,
              encrypted_PIN_block_length,
              encrypted_PIN_block,
              PIN_block_MAC_length,
              PIN_block_MAC,
              chip_encrypted_PIN_block_length,
              chip_encrypted_PIN_block)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0 - 2.

rule_array

Direction	Type
Input	Character

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 389. Keywords for DK Random PIN Generate2

Keyword	Meaning
PIN Block output selection keyword (One or two, optional)	
NOEPB	Do not return any encrypted PIN block (EPBs). This is the default. Cannot be combined with EBP or CHIP-EPB.
EPB	Return an encrypted PIN block. Cannot be combined with NOEBP.
CHIP-EPB	Return a chip encrypted PIN block. Cannot be combined with NOEPB.

PAN_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PAN_data* parameter. The value must be between 10 and 19, inclusive.

PAN_data

Direction	Type
Input	String

The primary account number (PAN) data used to generate PIN. Include the full account number, including the check digit.

card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_p_data* parameter. The value must be between 2 and 256, inclusive.

card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_t_data* parameter. The value must be between 2 and 256, inclusive.

card_t_data

Direction	Type
Input	String

The time-sensitive card data (Cdt) of the new account, determined by the card issuer, which, together with the account number and the *card_p_data*, specifies an individual card.

PIN_length

Direction	Type
Input	Integer

Specifies the length of the PIN to be generated. This value must be between 4 and 12, inclusive.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the PRW generation key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

PIN_print_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_print_key_identifier* parameter. If the *PIN_print_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 725.

PIN_print_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the PIN for printing. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOPP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_encryption_key_identifier* parameter. If the keyword EPB in the rule array is not specified, this value must be 0. If the *OPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the *OPIN_encryption_key_identifier_length* is 0, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

OEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OEPB_MAC_key_identifier* parameter. If the keyword EPB in the rule array is not specified, this value must be 0. If the *OEPB_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC of the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the *OEPB_MAC_key_identifier_length* is 0, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

OPIN_chip_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_chip_encryption_key_identifier* parameter. When the keyword CHIP-EPB in the rule array is not specified, this value must be 0. If the *OPIN_chip_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_chip_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to encrypt the optional PIN block for the personalization unit. The key identifier is an operational token or the key label of an operational token in key storage. When the *OPIN_chip_encryption_key_identifier_length* is 0, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

PIN_reference_value_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. The value must be at least 16. On output, it will be set to 16.

PIN_reference_value

Direction	Type
Output	String

The 16-byte calculated PIN reference value.

PRW_random_number_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be at least 4. On output, it will be set to 4.

PRW_random_number

Direction	Type
Output	String

The 4-byte random number associated with the PIN reference value.

PIN_print_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_print_block* parameter. It must be at least 32. On output, it will be set to 32.

PIN_print_block

Direction	Type
Output	String

The 32-byte encrypted PIN block to be passed to the PIN mailer function.

encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *encrypted_PIN_block* parameter. If the keyword EPB is not specified in the rule array, this value must be 0. Otherwise, it should be at least 32.

encrypted_PIN_block

Direction	Type
Output	String

The 32-byte encrypted PIN block PBF-1 format. If the *encrypted_PIN_block_length* value is zero, this parameter is ignored.

PIN_block_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_block_MAC* parameter. If the keyword EPB is not specified in the rule array, this value must be 0. Otherwise, it must be at least 8.

PIN_block_MAC

Direction	Type
Output	String

The 8-byte CMAC of the encrypted PIN block. If the *PIN_block_MAC_length* value is zero, this parameter is ignored.

chip_encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *chip_encrypted_PIN_block* parameter. If the keyword CHIP-EPB is not specified in the rule array, the value to 0. Otherwise, on input the value must be at least 32.

chip_encrypted_PIN_block

Direction	Type
Output	String

The 32-byte chip encrypted PIN block. If the *chip_encrypted_PIN_block_length* value is zero, this parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK Random PIN Generate2** access control point in the domain role controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 390. DK Random PIN Generate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	This service requires the July 2019 or later licensed internal code. Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	This service requires the July 2019 or later licensed internal code. Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	This service requires the July 2019 or later licensed internal code.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

DK Regenerate PRW (CSNBDRP and CSNEDRP)

Use the DK Regenerate PRW callable service to generate a new PIN reference value for a changed account number.

You can use this service to perform the following tasks:

- Generate a PIN reference value over the existing PIN and new PAN, which can be used to verify transactions.
- Generate an encrypted PIN block in PBF-1 format to be stored for later use in personalization of smart cards.

The callable service name for AMODE(64) invocation is CSNEDRP.

Format

```
CALL CSNBDRP(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             card_p_data_length,
             card_p_data,
             card_t_data_length,
             card_t_data,
             encrypted_PIN_block_length,
             encrypted_PIN_block,
             PIN_block_MAC_length,
             PIN_block_MAC,
             PRW_key_identifier_length,
             PRW_key_identifier,
             IPIN_encryption_key_identifier_length,
             IPIN_encryption_key_identifier,
             IEPB_MAC_key_identifier_length,
             IEPB_MAC_key_identifier,
             OPIN_encryption_key_identifier_length,
             OPIN_encryption_key_identifier,
             OEPB_MAC_key_identifier_length,
             OEPB_MAC_key_identifier,
             PIN_reference_value_length,
             PIN_reference_value,
             PRW_random_number_length,
             PRW_random_number,
             new_encrypted_PIN_block_length,
             new_encrypted_PIN_block,
             new_PIN_block_MAC_length,
             new_PIN_block_MAC)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. There are no keywords for this service.

card_p_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_p_data* parameter. The value must be between 2 and 256, inclusive.

card_p_data

Direction	Type
Input	String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_t_data* parameter. The value must be between 2 and 256, inclusive.

card_t_data

Direction	Type
Input	String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the *card_p_data*, specifies an individual card.

encrypted_PIN_block_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *encrypted_PIN_block* parameter. The value must be 32.

encrypted_PIN_block

Direction	Type
Input	String

The 32-byte encrypted PIN block in PBF-1 format of the input PIN.

PIN_block_MAC_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PIN_block_MAC* parameter. The value must be 8.

PIN_block_MAC

Direction	Type
Input	String

The 8-byte MAC of the encrypted PIN block.

PRW_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *PRW_key_identifier* parameter. If the *PRW_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

PRW_key_identifier

Direction	Type
Input/Output	String

The identifier of the PRW generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IPIN_encryption_key_identifier* parameter. If the *IPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

IPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to decrypt the PIN_block containing the current PIN. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate DECRYPT, CBC, and DKPINAD1.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *IEPB_MAC_key_identifier* parameter. If the *IEPB_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

IEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to verify MAC of the inbound encrypted PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, VERIFY, and DKPINAD1.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OPIN_encryption_key_identifier* parameter. If the *OPIN_encryption_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OPIN_encryption_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to encrypt the new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OEPB_MAC_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *OEPB_MAC_key_identifier* parameter. If the *OEPB_MAC_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 725.

OEPB_MAC_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to generate the MAC of new encrypted PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, GENONLY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

PIN_reference_value_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PIN_reference_value* parameter. This value must be 16. On output, it will be set to 16.

PIN_reference_value

Direction	Type
Output	String

The 16-byte calculated PIN reference value.

PRW_random_number_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *PRW_random_number* parameter. The value must be 4. On output, it will be set to 4.

PRW_random_number

Direction	Type
Output	String

The 4-byte random number associated with the PIN reference value.

new_encrypted_PIN_block_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_encrypted_PIN_block* parameter. The value should be at least 32.

new_encrypted_PIN_block

Direction	Type
Output	String

The 32-byte encrypted PIN block.

new_PIN_block_MAC_length

Direction	Type
Input/Output	Integer

Specifies the length in bytes of the *new_PIN_block_MAC* parameter. The value must be at least 8.

new_PIN_block_MAC

Direction	Type
Output	String

The 8-byte MAC of the encrypted PIN block.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The **DK Regenerate PRW** access control point in the domain role controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 391. DK Regenerate PRW required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).

Table 391. DK Regenerate PRW required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Chapter 10. X9.143 (TR-31) symmetric-key management

This topic describes the interchange of symmetric keys and other sensitive data between a CCA device and any other device that together share an external X9.143 (TR-31) key block (symmetric key exchange key). The format of this key block is defined in ANSI X9.143 Retail Financial Services Interoperable Secure Key Block Specification.

The key block is a format defined by the ANS Standards Committee to support interchange of symmetric keys in a secure manner and with key attributes included in the exchanged data. At the beginning of every key block is a header consisting of a required portion possibly followed by an optional portion. The format of the required portion along with the values that CCA supports are shown in "X9.143 (TR-31) key block header and optional block data" on page 1538. As can be seen from this table, offset 0 is a one-byte key block version ID field, also called a method, offset 5 is a two-byte key usage field which maps to CCA key type, offset 7 is a one-byte algorithm field for the key being protected, offset 8 is a one-byte mode of use field which maps to CCA key usage, and offset 11 is a one-byte exportability field. These fields are referenced later in this topic.

Prior to X9.143, there were problems with the interchange of symmetric keys. In the banking environment, it is very important that each symmetric key have a specific set of attributes attached to it, specifying such things as the cryptographic operations for which that key can be used. CCA implements these attributes in the form of the control vector (CV), but other vendors implement attributes in their own proprietary ways. Thus, if you are exchanging keys between CCA systems, you can securely pass the attributes using CCA functions and data structures. If, however, that same key was sent to a non-CCA system, there would be no secure way to do that. This is because the two cryptographic architectures have no common key format that could be used to pass both the key and its attributes. As a result, the normal approach has been to strip the attributes and send just the encrypted key, then attach attributes again at the receiving end.

The above scenario has major security problems because it allows an insider to obtain the key without its designated attributes. The insider can then attach other attributes to it, thereby compromising the security of the system. For example, assume the exchanged key is a key-encrypting key (KEK). The attributes of a KEK should restrict its use to key management functions that are designed to prevent exposure of the keys that the KEK is used to encrypt. If that KEK is transmitted without any attributes, an attacker on the inside can turn the key into a type used for data decryption. Such a key can then be used to decipher all the keys that were previously protected using the KEK. It is clearly very desirable to have a way of exchanging keys that prevents this modification of the attributes. X9.143 provides such a method.

The X9.143 key block has a set of defined key attributes. These attributes are securely bound to the key so that they can be transported together between any two systems that both understand the X9.143 format. This is much of the reason for its gain in popularity.

X9.143 uses some key attributes that are different from those in the CCA symmetric key-token. In some cases, there is a one-to-one correspondence between X9.143 attributes and CCA attributes. For these cases, conversion is simple and straightforward. In other cases, the correspondence is one-to-many or many-to-one, and the application program must provide information to help the CCA verbs decide how to perform the translation between CCA and X9.143 attributes. There are also CCA attributes that simply cannot be represented using X9.143. CCA keys with those attributes are not eligible for conversion to X9.143 format.

X9.143 allows two cryptographic methods for protecting the key block.

Key block version ID "B" (X'42'). Method "B" is a DES block cipher and uses an authenticated encryption scheme and cryptographic key derivation methods to produce the encryption and MAC keys. This method can protect DES and TDES keys.

Key block version ID "D" (X'44'). Method "D" is an AES block cipher and uses an authenticated encryption scheme and cryptographic key derivation methods to produce the encryption and MAC keys. This method can protect DES, TDES, AES, and HMAC keys.

The X9.143 key block has these two important features:

1. When protected by a DES key-encrypting key (method "B"), the key is wrapped in such a way that it meets the "key bundling" requirements of various standards. These standards state that the individual 8-byte blocks of a double-length or triple-length TDES key must be bound in such a way that they cannot be individually manipulated. X9.143 accomplishes this mainly by computation of a MAC across the entire structure, excluding the MAC value itself.
2. Key usage attributes, defined to control how the key can be used, are securely bound to the key itself. This makes it possible for a key and its attributes to be securely transferred from one party to another while assuring that the attributes of the key cannot be modified to suit the needs of an attacker.

The callable services that support the X9.143 (TR-31) key distribution protocol are:

- “TR-31 Create (CSNBT31C and CSNET31C)” on page 938
- “TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)” on page 1007
- “TR-31 Import (CSNBT31I and CSNET31I)” on page 959
- “TR-31 Optional Data Build (CSNBT31O and CSNET31O)” on page 998
- “TR-31 Optional Data Read (CSNBT31R and CSNET31R)” on page 1001
- “TR-31 Parse (CSNBT31P and CSNET31P)” on page 1004

TR-31 Create (CSNBT31C and CSNET31C)

The TR-31 Key Create service can be used to randomly generate AES, DES, and HMAC keys in TR-31 blocks or create a skeleton TR-31 key block header, as defined in the ANSI X9.143 standard.

The callable service name for AMODE(64) is CSNET31C.

Format

```
CALL CSNBT31C(
         return_code,
         reason_code,
         exit_data_length,
         exit_data,
         rule_array_count,
         rule_array,
         key_name_1_length,
         key_name_1,
         key_name_2_length,
         key_name_2,
         clear_key_bit_length,
         key_version_number_1,
         key_version_number_2,
         opt_blocks_1_length,
         opt_blocks_1,
         opt_blocks_2_length,
         opt_blocks_2,
         key_field_length_1,
         key_field_length_2,
         KEK_key_identifier_1_length,
         KEK_key_identifier_1,
         KEK_key_identifier_2_length,
         KEK_key_identifier_2,
         generated_key_identifier_1_length,
         generated_key_identifier_1,
         generated_key_identifier_2_length,
         generated_key_identifier_2,
         ob_data_length,
         ob_data,
         reserved2_length,
         reserved2,
         reserved3_length,
         reserved3,
```

```
        reserved4_length,
        reserved4 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 6 to 29, inclusive

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The *rule_array* keywords for this callable service are shown in the following table.

Table 392. Keywords for TR-31 Create control information

Keyword	Meaning
Key Block Algorithm (one required).	
AES	Specifies to build an AES key-block. Only valid with wrapping method 'D'
DES	Specifies to build a DES key-block.
HMAC	Specifies to build an HMAC key-block. Only valid with wrapping method 'D'. Only valid key usage 'KEY-HMAC' (M7).
Key Status (one required).	
NO-KEY	Specifies to build a TR-31 key block without a key value. This creates a key block header (For example, a skeleton key token) that can be populated using CSNBKPI2 or used in key derivation services such as CSNBEDH, CSNBUKD, and others.
RANDOM	Specifies to build a TR-31 key block using a randomly generated AES, DES, or HMAC key fitting the specifications given. This keyword requires a Key Form keyword.
Key Form (one required with RANDOM. Otherwise, not allowed). The first two characters refer to key usage of the single or first of a pair of keys. The next two characters refer to the second of a pair of keys. See "Usage notes" on page 954 and "Access control points" on page 954 for further details. • The "OP" string indicates an operational key be generated or the skeleton of an operational key be built. The key is wrapped by a symmetric master key. • The "EX" string indicates an exportable key be generated or the skeleton of an operational key be built. The key is wrapped by an exporter KEK (TR-31 key usage K0 with mode of use E) or a CCA exporter KEK with key usage EX, EXEX, IMEX, or OPEX. • The "IM" string indicates an importable key be generated or the skeleton of an operational key be built. The key is wrapped by an importer KEK (TR-31 key usage K0 with mode of use D) or a CCA importer KEK with key usage IM, IMEX, IMIM, or OPIM.	
EX	One key that can be sent to another system.
EXEX	A key pair; both keys to be sent elsewhere, possibly for exporting to two different systems. Both keys have the same clear value.
IM	One key that can be locally imported. The key can be imported onto this system to make it operational at another time.
IMEX	A key pair to be imported; one key to be imported locally and one key to be sent elsewhere. Both keys have the same clear value.
IMIM	A key pair to be imported; both keys to be imported locally at another time. Both keys have the same clear value.
OP	One operational key. The key is returned to the caller in operational form to be used locally.
OPEX	A key pair; one key that is operational and one key to be sent elsewhere. Both keys have the same clear value.
OPIM	A key pair; one key that is operational and one key to be imported locally at another time. Both keys have the same clear value.
OPOP	A key pair; either with the same key type with different associated data or complementary key types. Both keys have the same clear value.

Table 392. Keywords for TR-31 Create control information (continued)	
Keyword	**Meaning**
Key Context for *generated_key_identifier_1* (one required if *generated_key_identifier_1* is external. Otherwise, not allowed).	
K1-STRX	Either storage or key exchange context. This allows interoperability with legacy Key Blocks. **Note:** This does not imply that the wrapping key for a Key Block can be used for both storage and key exchange, merely that the storage or exchange of this Key Block is determined by the properties of the wrapping key. Sets the byte at offset 14 of the header to ASCII '0'.
K1-EXCH	Key exchange context only. The Key Block is wrapped by a transport key for exchange between a communicating pair. Sets the byte at offset 14 of the header to ASCII '2'.
Key Context for *generated_key_identifier_2* (one required if *generated_key_identifier_2* is external. Otherwise, not allowed).	
K2-STRX	Either storage or key exchange context. This allows interoperability with legacy Key Blocks. **Note:** This does not imply that the wrapping key for a Key Block can be used for both storage and key exchange, merely that the storage or exchange of this Key Block is determined by the properties of the wrapping key. Sets the byte at offset 14 of the header to ASCII '0'.
K2-EXCH	Key exchange context only. The Key Block is wrapped by a transport key for exchange between a communicating pair. Sets the byte at offset 14 of the header to ASCII '2'.
Wrapping method to be used for *generated_key_identifier_1* (one required).	
K1-BM-A	Use X9.143/TR-31 Binding Method 'A' for *generated_key_identifier_1*. Only valid with EX, EXEX, IM, IMEX, IMIM - cases where the *generated_key_identifier_1* will contain an external key block on output. The KEK key identifier 1 parameter must contain a TDES wrapping key.
K1-BM-B	Use X9.143/TR-31 Binding Method 'B' for *generated_key_identifier_1*. For Key Format keywords OP, OPEX, OPIM, OPOP where *generated_key_identifier_1* will contain an internal key block on output wrapped under the DES master key. For Key Formats EX, EXEX, IM, IMEX, IMIM, the KEK key identifier 1 parameter must contain a TDES wrapping key.
K1-BM-C	Use X9.143/TR-31 Binding Method 'C' for *generated_key_identifier_1*. Only valid with EX, EXEX, IM, IMEX, IMIM - cases where the *generated_key_identifier_1* will contain an external key block on output. The KEK key identifier 1 parameter must contain a TDES wrapping key.
K1-BM-D	Use X9.143/TR-31 Binding Method 'D' for *generated_key_identifier_1*. For Key Format keywords OP, OPEX, OPIM, OPOP where *generated_key_identifier_1* will contain an internal key block on output wrapped under the AES master key. For Key Formats EX, EXEX, IM, IMEX, IMIM, the KEK key identifier 1 parameter must contain an AES wrapping key
Wrapping method to be used for *generated_key_identifier_2* (one required for Key Formats EXEX, IMEX, IMIM, OPEX, OPIM, OPOP. Otherwise, invalid).	

Table 392. Keywords for TR-31 Create control information (continued)

Keyword	Meaning
K2-BM-A	Use X9.143/TR-31 Binding Method 'A' for *generated_key_identifier_2*. Only valid with EXEX, IMEX, IMIM, OPEX, OPIM - cases where the *generated_key_identifier_2* will contain an external key block on output. The KEK key identifier 2 parameter must contain a TDES wrapping key.
K2-BM-B	Use X9.143/TR-31 Binding Method 'B' for *generated_key_identifier_2*. For Key Format keyword OPOP where *generated_key_identifier_2* will contain an internal key block on output wrapped under the DES master key. For Key Formats EXEX, IMEX, IMIM, OPEX, OPIM, the KEK key identifier 2 parameter must contain a TDES wrapping key.
K2-BM-C	Use X9.143/TR-31 Binding Method 'C' for *generated_key_identifier_2*. Only valid with EXEX, IMEX, IMIM, OPEX, OPIM - cases where the *generated_key_identifier_2* will contain an external key block on output. The KEK key identifier 2 parameter must contain a TDES wrapping key.
K2-BM-D	Use X9.143/TR-31 Binding Method 'D' for *generated_key_identifier_2*. For Key Format keyword OPOP where *generated_key_identifier_2* will contain an internal key block on output wrapped under the AES master key. For Key Formats EXEX, IMEX, IMIM, OPEX, OPIM, the KEK key identifier 2 parameter must contain an AES wrapping key.
Key Usage (one required).	
BDK	Specifies to create a BDK base derivation key. Sets the bytes at offset 5 - 6 of the header to ASCII "B0". This key is used to derive the initial PIN encryption key (IPEK) in the derived unique key per transaction (DUKPT) process defined in X9.24-1. An initial key is derived for individual devices such as PIN pads. Only valid with Mode of Use 'X' (K*DERIVE).
CVK	Specifies to create a CVK card verification key. Sets the bytes at offset 5 - 6 of the header to ASCII "C0". Only valid with Modes of Use 'C' (K*GENVER), 'G' (K*-GEN), and 'V' (K*-VER).
DUKPT	Specifies to create an Initial DUKPT key. Sets the bytes at offset 5 - 6 of the header to ASCII "B1". Only valid with Mode of Use 'X' (K*DERIVE). Not valid with wrapping method 'A'.
EMVAC-E	Specifies to create a derivation key for an EMV/chip issuer master key: application cryptograms. Sets the bytes at offset 5 - 6 of the header to ASCII "E0". Only valid with Mode of Use 'X' (K*DERIVE).
EMVAC-F	Specifies to create an EMV/chip card key: application cryptograms. Sets the bytes at offset 5 - 6 of the header to ASCII "F0". Only valid with Mode of Use 'X' (K*DERIVE).
EMVCP-E	Specifies to create a derivation key for an EMV/chip issuer master key: card personalization. Sets the bytes at offset 5 - 6 of the header to ASCII "E5". Only valid with Mode of Use 'X' (K*DERIVE).
EMVDA-E	Specifies to create a derivation key for an EMV/chip issuer master key: data authentication code. Sets the bytes at offset 5 - 6 of the header to ASCII "E3". Only valid with Mode of Use 'X' (K*DERIVE).
EMVDA-F	Specifies to create an EMV/chip card key: data authentication code. Sets the bytes at offset 5 - 6 of the header to ASCII "F3". Only valid with Mode of Use 'X' (K*DERIVE).

Table 392. Keywords for TR-31 Create control information (continued)

Keyword	Meaning
EMVDN-E	Specifies to create a derivation key for an EMV/chip issuer master key: dynamic numbers. Sets the bytes at offset 5 - 6 of the header to ASCII "E4". Only valid with Mode of Use 'X' (K*DERIVE).
EMVDN-F	Specifies to create an EMV/chip card key: dynamic numbers. Sets the bytes at offset 5 - 6 of the header to ASCII "F4". Only valid with Mode of Use 'X' (K*DERIVE).
EMVSC-E	Specifies to create a derivation key for an EMV/chip issuer master key: secure messaging for confidentiality. Sets the bytes at offset 5 - 6 of the header to ASCII "E1". Only valid with Mode of Use 'X' (K*DERIVE).
EMVSC-F	Specifies to create an EMV/chip card key: secure messaging for confidentiality. Sets the bytes at offset 5 - 6 of the header to ASCII "F1". Only valid with Mode of Use 'X' (K*DERIVE).
EMVSI-E	Specifies to create a derivation key for an EMV/chip issuer master key: secure messaging for integrity. Sets the bytes at offset 5 - 6 of the header to ASCII "E2". Only valid with Mode of Use 'X' (K*DERIVE).
EMVSI-F	Specifies to create an EMV/chip card key: secure messaging for integrity. Sets the bytes at offset 5 - 6 of the header to ASCII "F2". Only valid with Mode of Use 'X' (K*DERIVE).
ENC	Specifies to create a symmetric data encryption key. Sets the bytes at offset 5 - 6 of the header to ASCII "D0". Only valid with Modes of Use 'E' (K*-ENC), 'D' (K*-DEC), and 'B' (K*ENCDEC).
ENCSENS	Specifies to create a data encryption key for sensitive data. Sets the bytes at offset 5 - 6 of the header to ASCII "D3". Only valid with Modes of Use 'E' (K*-ENC), 'D' (K*-DEC), and 'B' (K*ENCDEC). Only valid with wrapping method 'B' and 'D'.
ISOMAC0	Specifies to create an ISO 16609 MAC algorithm 1 key (based on ISO 9797-1) using TDEA. Sets the bytes at offset 5 - 6 of the header to ASCII "M0". Only valid with Modes of Use 'C' (K*GENVER), 'G' (K*-GEN), and 'V' (K*-VER).
ISOMAC1	Specifies to create an ISO 9797-1 MAC algorithm 1 key. Sets the bytes at offset 5 - 6 of the header to ASCII "M1". Only valid with Modes of Use 'C' (K*GENVER), 'G' (K*-GEN), and 'V' (K*-VER).
ISOMAC3	Specifies to create an ISO 9797-1 MAC algorithm 3 key. Sets the bytes at offset 5 - 6 of the header to ASCII "M3". Only valid with Modes of Use 'C' (K*GENVER), 'G' (K*-GEN), and 'V' (K*-VER).
ISOMAC6	Specifies to create an ISO 9797-1:2011 MAC algorithm 5/CMAC key. Sets the bytes at offset 5 - 6 of the header to ASCII "M6". Only valid with Modes of Use 'C' (K*GENVER), 'G' (K*-GEN), and 'V' (K*-VER). Valid with wrapping methods ‘B’, ‘C’, and 'D' and algorithms ‘TDES’ and 'AES'. Not valid with wrapping method 'A'.
KDK	Specifies to create a Key Derivation Key. Sets the bytes at offset 5 - 6 of the header to ASCII "B3". Only valid with Mode of Use 'X' (K*DERIVE). Not valid with wrapping method 'A'.
KEK	Specifies to create a key encryption or wrapping key. Sets the bytes at offset 5 - 6 of the header to ASCII "K0". Only valid with Modes of Use 'E' (K*-ENC) and 'D' (K*-DEC).

Table 392. Keywords for TR-31 Create control information (continued)

Keyword	Meaning
KEK-WRAP	Specifies to create a TR-31 key block protection key. Sets the bytes at offset 5 - 6 of the header to ASCII "K1". Only valid with Modes of Use 'E' (K*-ENC) and 'D' (K*-DEC). Not valid with wrapping method 'A'.
KEK-WRK4	Specifies to create an ISO 20038 key block protection key. Sets the bytes at offset 5 - 6 of the header to ASCII "K4". Only valid with Modes of Use 'E' (K*-ENC) and 'D' (K*-DEC). Only valid with wrapping method 'D'.
KEY-HMAC	Specifies to create an HMAC algorithm key. Sets the bytes at offset 5 - 6 of the header to ASCII "M7". Only valid with algorithm "HMAC" set. Only valid with Modes of Use 'C' (K*GENVER), 'G' (K*-GEN), and 'V' (K*-VER).
PINENC	Specifies to create a PIN encryption key. Sets the bytes at offset 5 - 6 of the header to ASCII "P0". Only valid with Modes of Use 'E' (K*-ENC), 'D' (K*-DEC), and 'B' (K*ENCDEC).
PINV3624	Specifies to create a PIN verification, IBM 3624 key. Sets the bytes at offset 5 - 6 of the header to ASCII "V1". Only valid with Modes of Use 'C' (K*GENVER), 'G' (K*-GEN), and 'V' (K*-VER).
PINVO	Specifies to create a PIN verification, KPV, other algorithm key. Sets the bytes at offset 5 - 6 of the header to ASCII "V0". Only valid with Modes of Use 'C' (K*GENVER), 'G' (K*-GEN), and 'V' (K*-VER).
VISAPVV	Specifies to create a PIN verification, Visa PVV key. Sets the bytes at offset 5 - 6 of the header to ASCII "V2". Only valid with Modes of Use 'C' (K*GENVER), 'G' (K*-GEN), and 'V' (K*-VER).
Mode of Use to be used for *generated_key_identifier_1* (one required).	
K1ENCDEC	Specifies that *generated_key_identifier_1* can be used for both encrypting and decrypting of data, or wrapping and unwrapping keys. Sets the byte at offset 8 of the header to ASCII 'B'.
K1GENVER	Specifies that *generated_key_identifier_1* can be used for both generating and verifying of check/PIN value. Sets the byte at offset 8 of the header to ASCII 'C'.
K1-DEC	Specifies that *generated_key_identifier_1* can be used to decrypt data or unwrap keys only. Sets the byte at offset 8 of the header to ASCII 'D'.
K1-ENC	Specifies that *generated_key_identifier_1* can be used to encrypt data or wrap keys only. Sets the byte at offset 8 of the header to ASCII 'E'.
K1-GEN	Specifies that *generated_key_identifier_1* can be used to generate a check/PIN value only. Sets the byte at offset 8 of the header to ASCII 'G'.
K1-VER	Specifies that *generated_key_identifier_1* can be used to verify a check/PIN value only. Sets the byte at offset 8 of the header to ASCII 'V'.
K1DERIVE	Specifies that *generated_key_identifier_1* can be used to derive other keys. Sets the byte at offset 8 of the header to ASCII 'X'.
Mode of Use to be used for *generated_key_identifier_2* (one required for Key Formats EXEX, IMEX, IMIM, OPEX, OPIM, OPOP. Otherwise, invalid.	

Table 392. Keywords for TR-31 Create control information (continued)	
Keyword	**Meaning**
K2ENCDEC	Specifies that *generated_key_identifier_2* can be used for both encrypting and decrypting of data, or wrapping and unwrapping keys. Sets the byte at offset 8 of the header to ASCII 'B'.
K2GENVER	Specifies that *generated_key_identifier_2* can be used for both generating and verifying of check/PIN value. Sets the byte at offset 8 of the header to ASCII 'C'.
K2-DEC	Specifies that *generated_key_identifier_2* can be used to decrypt data or unwrap keys only. Sets the byte at offset 8 of the header to ASCII 'D'. If K1-DEC is set, this keyword is invalid.
K2-ENC	Specifies that *generated_key_identifier_2* can be used to encrypt data or wrap keys only. Sets the byte at offset 8 of the header to ASCII 'E'. If K1-ENC is set, this keyword is invalid.
K2-GEN	Specifies that *generated_key_identifier_2* can be used to generate a check/PIN value only. Sets the byte at offset 8 of the header to ASCII 'G'. If K1-GEN is set, this keyword is invalid.
K2-VER	Specifies that *generated_key_identifier_2* can be used to verify a check/PIN value only. Sets the byte at offset 8 of the header to ASCII 'V'. If K1-VER is set, this keyword is invalid.
K2DERIVE	Specifies that generated_key_identifier_2 can be used to derive other keys. Sets the byte at offset 8 of the header to ASCII 'X'.
Exportability to be used for *generated_key_identifier_1* (one required).	
K1-TRUST	Specifies that *generated_key_identifier_1* will be exportable under a KEK in a form meeting the requirements of ANSI X9.24 Parts 1 or 2. Sets the byte at offset 11 of the header to ASCII 'E'.
K1-NONE	Specifies that *generated_key_identifier_1* will be non-exportable. Sets the byte at offset 11 of the header to ASCII 'N'.
K1-SENS	Specifies that *generated_key_identifier_1* is sensitive, exportable under a key-encrypting key in a form not necessarily meeting the requirements of X9.24 Parts 1 or 2. Sets the byte at offset 11 of the header to ASCII 'S'.
Exportability to be used for *generated_key_identifier_2* (one required for Key Formats EXEX, IMEX, IMIM, OPEX, OPIM, OPOP. Otherwise, invalid).	
K2-TRUST	Specifies that *generated_key_identifier_2* will be exportable under a KEK in a form meeting the requirements of ANSI X9.24 Parts 1 or 2. Sets the byte at offset 11 of the header to ASCII 'E'.
K2-NONE	Specifies that *generated_key_identifier_2* will be non-exportable. Sets the byte at offset 11 of the header to ASCII 'N'.
K2-SENS	Specifies that *generated_key_identifier_2* is sensitive, exportable under a key-encrypting key in a form not necessarily meeting the requirements of X9.24 Parts 1 or 2. Sets the byte at offset 11 of the header to ASCII 'S'.
HMAC Hash algorithm (one required with Algorithm HMAC. Otherwise, invalid).	

Table 392. Keywords for TR-31 Create control information (continued)

Keyword	Meaning
ISOSHA-1	Specifies to use the SHA-1 hash algorithm with the HMAC key as defined by ISO 20038. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII 'H' and does not include an optional block with a block ID of "HM". **Note:** Keyword ISOSHA-1 creates a TR-31 key block with an algorithm of 'H' and no optional block with a block ID of "HM". Under ISO 20038 this key block allows only SHA-1 as the hash algorithm to use with the HMAC key. However, ASC X9 TR 31-2018 also allows a key block with an algorithm of 'H' and no optional block with a block ID of "HM", which is interpreted as an HMAC key with no hash algorithm limits: there is no limit to SHA-1. For this reason, it is recommended to use the ISOSHA-1 keyword only when sending a key to a partner that is known to require and understand the ISO 20038 version of the hash limit, or to have a clear understanding that the partner will receive an HMAC key with no hash algorithm limits under TR 31-2018. When possible, the SHA-1 keyword should be used instead, provided that the partner can receive a key block with the HM optional block that limits hash algorithm.
ISOSHA-2	Specifies to use the SHA-2 hash algorithm with the HMAC key as defined by ISO 20038. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII 'I' and does not include an optional block with a block ID of "HM".
SHA-1	Specifies to use the SHA-1 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII 'H' and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "10".
SHA-224	Specifies to use the SHA-224 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII 'H' and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "20".
SHA-256	Specifies to use the SHA-256 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII 'H' and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "21".
SHA-384	Specifies to use the SHA-384 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII 'H' and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "22".
SHA-512	Specifies to use the SHA-512 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII 'H' and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "23".
Compliance tag (one, optional, only valid when generating an internal or skeleton key block).	

Table 392. Keywords for TR-31 Create control information (continued)

Keyword	Meaning
COMP-TAG	Generate a compliant-tagged key block (internal or skeleton). This may be used to create a skeleton key token with the compliance-tag at any time. However, when used to create an operational key block (whether directly from TR-31 Create or by providing a skeleton key block to another service), the request must be sent to an adapter domain that is in PCI-HSM 2016 compliance mode to be provisioned with key material.
NOCMPTAG	Do not generate a compliant-tagged key. This is the default.
CPACF Control (one, optional, if building an internal token or K*OB-IBM is set).	
XPRTCPAC	Allow export to CPACF protected key format.
NOEXCPAC	Prohibit export to CPACF protected key format. This is the default.
DK PIN Enable (one, optional, if building an internal token or K*OB-IBM is set).	
DKPINOP	Allow the key to be used for DK operations.
Optional blocks to add for key 1 (multiple, optional, only valid when *generated_key_identifier_1* is external). If *generated_key_identifier_1* is internal, these OBs will be automatically added.	
K1OB-IBM	Specifies to add a proprietary IBM optional block to *generated_key_identifier_1*. Currently, this optional block contains a compliance tag bit and KDF indicator.
K1OB-KC	Specifies to add a KC optional block to *generated_key_identifier_1*. This optional block contains a key check value of the key that is in the key block. This keyword is not valid with single length DES keys or HMAC keys.
K1OB-KP	Specifies to add a KP optional block to *generated_key_identifier_1*. This optional block contains a key check value of the key that is used to wrap the key in the key block (for example, the KEK).
K1OB-TC	Specifies to add a TC optional block to *generated_key_identifier_1*. This optional block contains the UTC time when the key block was initially created.
K1OB-TS	Specifies to add a TS optional block to *generated_key_identifier_1*. This optional block contains the UTC time when the current key block was created. This field changes when the key is wrapped under a new KEK or MK, such as in CSNBKTC2.
K1OB-WP	Specifies to add a WP optional block to *generated_key_identifier_1*. This optional block contains the wrapping pedigree of the key. This documents if the key was ever wrapped by a key that is weaker than itself.
Optional blocks to add for key 2 (multiple, optional, only valid when *generated_key_identifier_2* is external). If *generated_key_identifier_2* is internal, these OBs will be automatically added).	
K2OB-IBM	Specifies to add a proprietary IBM optional block to *generated_key_identifier_2*. Currently, this optional block contains a compliance tag bit and KDF indicator.
K2OB-KC	Specifies to add a KC optional block to *generated_key_identifier_2*. This optional block contains a key check value of the key that is in the key block. This keyword is not valid with single length DES keys or HMAC keys.

Table 392. Keywords for TR-31 Create control information (continued)	
Keyword	**Meaning**
K2OB-KP	Specifies to add a KP optional block to *generated_key_identifier_2*. This optional block contains a key check value of the key that is used to wrap the key in the key block (for example, the KEK).
K2OB-TC	Specifies to add a TC optional block to generated_key_identifier_2. This optional block contains the UTC time when the key block was initially created.
K2OB-TS	Specifies to add a TS optional block to *generated_key_identifier_2*. This optional block contains the UTC time when the current key block was created. This field changes when the key is wrapped under a new KEK or MK, such as in CSNBKTC2.
K2OB-WP	Specifies to add a WP optional block to *generated_key_identifier_2*. This optional block contains the wrapping pedigree of the key. This documents if the key was ever wrapped by a key that is weaker than itself.
Usage-specific OBs (1 or 2, optional, only valid when *ob_data_length* is non-zero). These keywords must be specified for internal or external tokens to add the optional block data sent in via *ob_data*.	
K1OB-DA	Specifies to add a DA optional block to *generated_key_identifier_1*. This optional block contains information on the derivations allowed for a derivation key, which are sent in the *ob_data* parameter. Only valid with usage B3.
K2OB-DA	Specifies to add a DA optional block to *generated_key_identifier_2*. This optional block contains information on the derivations allowed for a derivation key, which are sent in the *ob_data* parameter. Only valid with usage B3.

key_name_1_length

Direction	**Type**
Input	Integer

The length of the *key_name_1* parameter for *generated_key_identifier_1*. Valid values are 0 and 64 bytes.

key_name_1

Direction	**Type**
Input	String

A 64-byte key store label to be stored in the associated data of *generated_key_identifier_1*, if *key_name_1_length* is non-zero.

key_name_2_length

Direction	**Type**
Input	Integer

The length of the *key_name_2* parameter for *generated_key_identifier_2*. Valid values are 0 and 64 bytes.

When only one key is being created (Key form rules OP, IM, EX), this parameter is ignored and treated as if zero was specified.

key_name_2

Direction	Type
Input	String

A 64-byte key store label to be stored in the associated data of *generated_key_identifier_2*, if *key_name_2_length* is non-zero.

clear_key_bit_length

Direction	Type
Input	Integer

The size (in bits) of the key to be generated. If a skeleton key is being generated (a Key Status rule of NO-KEY), this parameter will be ignored.

- For the HMAC algorithm, this is a value between 80 and 2048, inclusive.
- For the AES algorithm, this is a value of 128, 192, or 256.
- For the DES algorithm, this is a value of 56, 64, 112, 128, 168, or 192.

key_version_number_1

Direction	Type
Input	String

The two-byte key version number to be inserted in the Key Version Number field of the key block in *generated_key_identifier_1*. If no key version number is needed, the value should be EBCDIC ("00").

key_version_number_2

Direction	Type
Input	String

The two-byte key version number to be inserted in the Key Version Number field of the key block in *generated_key_identifier_2*. If no key version number is needed, the value should be EBCDIC ("00").

When only one key is being created (Key form rules OP, IM, EX), this parameter is ignored.

opt_blocks_1_length

Direction	Type
Input	Integer

The length of parameter *opt_blocks_1* in bytes. If no optional data is to be included in *generated_key_identifier_1*, this parameter must be set to zero.

opt_blocks_1

Direction	Type
Input	String

The optional block data to be included in the key block in *generated_key_identifier_1*. The optional block data is prepared using the TR-31 Optional Data Build callable service and must be in ASCII.

This parameter is ignored if *opt_blks_length_1* is zero.

opt_blocks_2_length

Direction	Type
Input	Integer

The length of parameter *opt_blocks_2* in bytes. If no optional data is to be included in *generated_key_identifier_2*, this parameter must be set to zero.

When only one key is being created (Key form rules OP, IM, EX), this parameter is ignored and treated as if zero was specified.

opt_blocks_2

Direction	Type
Input	String

The optional block data to be included in the key block in *generated_key_identifier_2*. The optional block data is prepared using the TR-31 Optional Data Build callable service and must be in ASCII.

This parameter is ignored if *opt_blks_length_2* is zero.

key_field_length_1

Direction	Type
Input	Integer

The number of bytes of data in the encrypted portion of the key block in *generated_key_identifier_1*. This value does not allow for ASCII encoding of the encrypted data stored in the key field according to the specification. For example, when a value of 32 is specified here, the length of the final ASCII-encoded encrypted data in the key field in the output key block is 64 bytes.

If *generated_key_identifier_1* will contain an internal key, this parameter will be ignored and treated as if set to the maximum value for the specified algorithm (padding will be used if needed):

- DES: 32
- AES: 48
- HMAC: 272

If *generated_key_identifier_1* will contain an external key, this parameter will be used, and you must specify the number of bytes needed that is less than or equal to the maximum for the algorithm specified.

key_field_length_2

Direction	Type
Input	Integer

The number of bytes of data in the encrypted portion of the key block in *generated_key_identifier_2*. This value does not allow for ASCII encoding of the encrypted data stored in the key field according to the specification. For example, when a value of 32 is specified here, the length of the final ASCII-encoded encrypted data in the key field in the output key block is 64 bytes.

If *generated_key_identifier_2* will contain an internal key, this parameter will be ignored and treated as if set to the maximum value for the specified algorithm (padding will be used if needed):

- DES: 32
- AES: 48
- HMAC: 272

If *generated_key_identifier_2* will contain an external key, this parameter will be used, and you must specify the number of bytes needed that is less than or equal to the maximum for the algorithm specified.

When only one key is being created (Key form rules OP, IM, EX), this parameter is ignored and treated as if zero was specified.

KEK_key_identifier_1_length

Direction	Type
Input	Integer

The length in bytes of the *KEK_key_identifier_1* parameter.

For key forms OP, OPOP, OPIM, or OPEX, the value must be zero. Otherwise, if the *KEK_key_identifier_1* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 9992.

KEK_key_identifier_1

Direction	Type
Input/Output	String

When *KEK_key_identifier_1_length* is zero, this parameter is ignored.

The key-encrypting key to wrap the key block in *generated_key_identifier_1* when the key block contains an external key.

If the wrapping method is K1-BM-A, K1-BM-B, or K1-BM-C:

- If using a TR-31 KEK, it must be key usage K1, algorithm T, and mode of use E (key form EX) or D (key form IM).
- If using a CCA KEK, it must be a DES EXPORTER (key form EX) or IMPORTER (key form IM).

If the wrapping method is or K1-BM-D:

- If using a TR-31 KEK, it must be key usage K1, algorithm A, and mode of use E (key form EX) or D (key form IM).
- If using a CCA KEK, it must be a variable-length AES EXPORTER (key form EX) or IMPORTER (key form IM).

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

KEK_key_identifier_2_length

Direction	Type
Input	Integer

The length in bytes of the *KEK_key_identifier_2* parameter.

When only one key is being created (Key form rules OP, IM, EX), this parameter is ignored and treated as if zero was specified.

For key forms OPOP, the value must be zero. Otherwise, if the *KEK_key_identifier_2* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 9992.

KEK_key_identifier_2

Direction	Type
Input/Output	String

When *KEK_key_identifier_2_length* is zero, this parameter is ignored.

The key-encrypting key to wrap the key block in *generated_key_identifier_2* when the key block contains an external key.

If the wrapping method is K2-BM-A, K2-BM-B, or K2-BM-C:

- If using a TR-31 KEK, it must be key usage K1, algorithm T, and mode of use E (key form EX) or D (key form IM).
- If using a CCA KEK, it must be a DES EXPORTER (key form EX) or IMPORTER (key form IM).

If the wrapping method is or K2-BM-D:

- If using a TR-31 KEK, it must be key usage K1, algorithm A, and mode of use E (key form EX) or D (key form IM).
- If using a CCA KEK, it must be a variable-length AES EXPORTER (key form EX) or IMPORTER (key form IM).

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

generated_key_identifier_1_length

Direction	Type
Input/Output	Integer

On input, the length of the buffer for the *generated_key_identifier_1* in bytes. The maximum value is 9992 bytes.

On output, the parameter will hold the actual length of the *generated_key_identifier_1*.

generated_key_identifier_1

Direction	Type
Output	String

The buffer that is to receive the first generated key. If the Key Status is set to RANDOM, the generated key block will be placed in this variable. If the Key Status is set to NO-KEY, this variable will contain the key block header (KBH) (for example, a skeleton key token).

generated_key_identifier_2_length

Direction	Type
Input/Output	Integer

On input, the length of the buffer for the *generated_key_identifier_2* in bytes. The maximum value is 9992 bytes.

On output, the parameter will hold the actual length of the *generated_key_identifier_2*.

When only one key is being created (Key form rules OP, IM, EX), this parameter is ignored and treated as if zero was specified.

generated_key_identifier_2

Direction	Type
Output	String

The buffer that is to receive the second generated key.

When only one key is generated, this parameter is ignored.

Otherwise, it is handled in the same manner as the *generated_key_identifier_1* parameter.

ob_data_length

Direction	Type
Input	String

The length of parameter *ob_data* in bytes. This value must be 0 or a number that is divisible by 5 and less than or equal to 245. This MUST be 0 unless a key block with B3 usage is being built and K*OB-DA is set for at least one of the keys.

ob_data

Direction	Type
Input/Output	String

This parameter contains the optional block data for the DA optional block and is ONLY valid if you are building a key with usage B3. Rules K1OB-DA and/or K2OB-DA MUST be set for each key that should include a DA optional block with this data. For the DA optional block, the data consists of strings of 5 ASCII bytes that provide information for each derivation that is allowed for the key. Each set of 5 byes contains the key usage (2 bytes), algorithm (1 byte), mode of use (1 byte), and exportability (1 byte).

If *ob_data_length* is 0, this parameter will be ignored.

Below are two examples of valid inputs to this parameter:

1. P0TEE
   - PIN Encryption key, algorithm TDEA, Encrypt only, Exportable
2. P0TDEP0TEED0AEN
   a. First Derivation allowed: PIN Encryption key, algorithm TDEA, Decrypt only, Exportable
   b. Second Derivation allowed: PIN Encryption key, algorithm TDEA, Encrypt only, Exportable
   c. Third Derivation allowed: Symmetric Data Encryption key, algorithm AES, Encrypt only, Non-Exportable

reserved2_length

Direction	Type
Input	Integer

Length in bytes of the *reserved2* parameter. The value must be 0.

reserved2

Direction	Type
Input/Output	String

This parameter is ignored.

reserved3_length

Direction	Type
Input	Integer

Length in bytes of the *reserved3* parameter. The value must be 0.

reserved3

Direction	Type
Input/Output	String

This parameter is ignored.

reserved4_length

Direction	Type
Input	Integer

Length in bytes of the *reserved4* parameter. The value must be 0.

reserved4

Direction	Type
Input/Output	String

This parameter is ignored.

Restrictions

Proprietary values for the TR-31 header fields are not supported by this callable service with the exception of the proprietary values used by IBM CCA when carrying a control vector in an optional block in the header.

Usage notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned (see Appendix F, "EBCDIC and ASCII default conversion tables," on page 1657). TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Access control points

The following table shows more access controls that are specific to TR-31 Create, along with the DK specific access controls that are used in TR-31 Create. See Table C4 for more information about the DK-specific access controls.

Table 393. Access controls specific to TR-31 Create and DK-specific access controls used in TR-31 Create

Keyword	Offset	Name	Description
AES	03C1	T31C - Permit TR-31 AES creation	Permits the use of TR-31 key blocks with algorithm AES.
DES	03C2	T31C - Permit TR-31 DES creation	Permits the use of TR-31 key blocks with algorithm DES.
HMAC	03C3	T31C - Permit TR-31 HMAC creation	Permits the use of TR-31 key blocks with algorithm HMAC.

Table 393. Access controls specific to TR-31 Create and DK-specific access controls used in TR-31 Create (continued)

Keyword	Offset	Name	Description
OP, OPOP	03C4	T31C - Permit TR-31 internal key creation	When building a single internal key or an internal key pair in T31C, this command must be enabled. Internal/External is determined by the Key Context (byte 14 of the TR-31 KBH), with 0X31 indicating an internal key. For internal keys, there is no key context rule array keyword.
IM, EX, IMIM, IMEX, EXEX	03C5	T31C - Permit TR-31 external key creation	When building a single external key or an external key pair in T31C, this command must be enabled. Internal/External is determined by the Key Context (byte 14 of the TR-31 KBH), with 0X30 or 0X32 indicating an external key.
OPIM, OPEX	03C6	T31C - Permit TR-31 internal/external key pair creation	When building a key pair with one internal key and one external key, this command must be enabled.
K1-BM-A or K2-BM-A	03C7	T31C - Permit TR-31 KB Version A creation	Permits the use of TR-31 key blocks with wrapping method A.
K1-BM-B or K2-BM-B	03C8	T31C - Permit TR-31 KB Version B creation	Permits the use of TR-31 key blocks with wrapping method B.
K1-BM-C or K2-BM-C	03C9	T31C - Permit TR-31 KB Version C creation	Permits the use of TR-31 key blocks with wrapping method C.
K1-BM-D or K2-BM-D	03CA	T31C - Permit TR-31 KB Version D creation	Permits the use of TR-31 key blocks with wrapping method D.

Valid key usage and key form keywords for a single key can be seen on table C2 below:

Table 394. Key usage keywords for a single key

Key usage	Mode of use	OP	EX	IM
B0	X	X	X	X

Table 394. Key usage keywords for a single key (continued)

Key usage	Mode of use	OP	EX	IM
B1	X	X	X	X
B3	X	X	X	X
C0	C,G,V	X	X	X
D0	B,D,E	X	X	X
D3	B,D,E	X	X	X
E0	X	X	X	X
E1	X	X	X	X
E2	X	X	X	X
E3	X	X	X	X
E4	X	X	X	X
E5	X	X	X	X
F0	X	X	X	X
F1	X	X	X	X
F2	X	X	X	X
F3	X	X	X	X
F4	X	X	X	X
M0	C,G,V	X	X	X
M1	C,G,V	X	X	X
M3	C,G,V	X	X	X
M7	C,G,V	X	X	X
P0	B,D,E	X	X	X
V0	C,G,V	X	X	X
V1	C,G,V	X	X	X
V2	C,G,V	X	X	X

Valid key usage and key form keywords for a key pair can be seen on table C3. Certain keys cannot be built together as a pair. For example, you cannot build two keys together that both have Key Usage "D0" and Mode of Use "D" because you need to have an encryption key to pair with a decryption key. The table below shows the valid rule array values and key pairs that can be created.

Table 395. Key usage and key form keywords

generated_key_identifier_1		***generated_key_identifier_2***		**Key form**				
Key usage	**Mode of use**	**Key usage**	**Mode of use**	**OPOP**	**OPOP, OPIM, IMIM**	**OPEX**	**EXEX**	**IMEX**
B0	X	B0	X		X	X	X	X

Table 395. Key usage and key form keywords (continued)

generated_key_identifier_1		***generated_key_identifier_2***		**Key form**				
Key usage	**Mode of use**	**Key usage**	**Mode of use**	**OPOP**	**OPOP, OPIM, IMIM**	**OPEX**	**EXEX**	**IMEX**
B1	X	B1	X		X	X	X	X
B3	X	B3	X		X	X	X	X
C0	G	C0	G		X	X	X	X
C0	G	C0	V		X	X	X	X
C0	V	C0	G		X	X	X	X
D0	D	D0	E		X	X	X	X
D0	D	D0	E (XLATE)		X	X	X	X
D0	E	D0	D		X	X	X	X
D0	E	D0	D (XLATE)		X	X	X	X
D0	D (XLATE)	D0	E		X	X	X	X
D0	D (XLATE)	D0	E (XLATE)			X	X	X
D0	E (XLATE)	D0	D		X	X	X	X
D0	E (XLATE)	D0	D (XLATE)			X	X	X
D3	B	D3	B			X	X	X
D3	D	D3	E			X	X	X
D3	E	D3	D			X	X	X
K0	E	K0	D			X	X	X
K0	D	K0	E			X	X	X
K1	D	K1	E			X	X	X
K1	E	K1	D			X	X	X
K4	D	K4	E			X	X	X
K4	E	K4	D			X	X	X
M*	G	M*	V		X	X	X	X
M*	G	M*	G		X	X	X	X
M*	V	M*	G		X	X	X	X
P0 ISO4 AES	D	P0 ISO4 AES	E	X				
P0 ISO4 AES	E	P0 ISO4 AES	D	X				
P0 DES	E	P0 DES	D		X	X	X	X
P0 DES	D	P0 DES	E		X	X	X	X

Table 395. Key usage and key form keywords (continued)

generated_key_identifier_1		*generated_key_identifier_2*		Key form				
Key usage	**Mode of use**	**Key usage**	**Mode of use**	**OPOP**	**OPOP, OPIM, IMIM**	**OPEX**	**EXEX**	**IMEX**
V0	G	V0	V			X	X	X
V0	V	V0	G			X	X	X
V1	G	V1	V			X	X	X
V1	V	V1	G			X	X	X
V2	G	V2	V			X	X	X
V2	V	V2	G			X	X	X

The following access controls affect the wrapping of a key with a weaker transport key:

When the wrapping key-encrypting key identifier is a weaker key than the key being generated:

- The service will fail if the **Prohibit weak wrapping - Transport keys** access control is enabled.
- The service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control is enabled.

When the wrapping master key is a weaker key than the key being generated:

- The service will fail if the **Prohibit weak wrapping - Master keys** access control is enabled.
- The service will complete successfully with a warning return code if the **Warn when weak wrap – Master keys** access control is enabled.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 396. TR-31 Create required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 IBM z13s		This service is not supported.
IBM z14 IBM z14 ZR1		This service is not supported.
IBM z15 IBM z15 T02		This service is not supported.

Table 396. TR-31 Create required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 IBM z16 A02	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	This service is not supported.
	Crypto Express8 CCA	This service requires the CCA release 8.1 or later licensed internal code (LIC).

TR-31 Import (CSNBT31I and CSNET31I)

Use the TR-31 Import callable service to convert a TR-31 key block to a DES CCA key token. Since there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the caller may need to specify the attributes to attach to the imported key through the rule array.

Use the TR-31 Import service to convert a TR-31 key block to an AES, DES, or HMAC CCA key token. The key block is wrapped by an AES key-encrypting key using the key wrap algorithm defined in ISO:20038. Proprietary values are used in the key block.

Note: The ANSI X9 TR-31-2018 optional block, which limits the hash algorithm of the HMAC key, is also supported.

The callable service name for AMODE(64) is CSNET31I.

Format

```
CALL CSNBT31I(
          return_code,
          reason_code,
          exit_data_length,
          exit_data,
          rule_array_count,
          rule_array,
          TR31_key_block_length,
          TR31_key_block,
          unwrap_kek_identifier_length,
          unwrap_kek_identifier,
          wrap_kek_identifier_length,
          wrap_kek_identifier,
          output_key_identifier_length,
          output_key_identifier,
          num_opt_blks,
          cv_source,
          protection_method )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 1 - 4.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The *rule_array* keywords for this callable service are shown in the following table. One keyword from one CCA output key usage subgroup shown in the following table is required based on TR-31 input key usage, unless the CV is included in the TR-31 key block as an optional block. If the CV is included in the TR-31 key block as an optional block, the included CV will be used in the output key block as long as it does not conflict with the TR-31 header data.

See Table 409 on page 989 for valid combinations of Usage and Mode.

Table 397. Keywords for TR-31 Import Rule Array Control Information

Keyword	Meaning
Token identifier (One Required)	
INTERNAL	Specifies to return the output key in an internal CCA key-token.

Table 397. Keywords for TR-31 Import Rule Array Control Information (continued)	
Keyword	**Meaning**
EXTERNAL	Specifies to return the output key in an external CCA symmetric key token, wrapped by the transport key identified by the *wrap_kek_identifier* parameter if one is provided or else wrapped by the key identified by the *unwrap_kek_identifier* parameter, or a clear initialization vector ("IO"), see Table 406 on page 980.
Wrap KEK key algorithm (one optional). Only valid for EXTERNAL.	
WKEY-AES	Specifies that the *wrap_key_identifier* parameter identifies an operational variable-length AES key-token or the label of a record in AES key-storage. Only valid with keyword VARDRV-D.
WKEY-DES	Specifies that the *wrap_key_identifier* parameter identifies an operational fixed-length DES key-token or the label of such a record in DES key-storage. Not valid with keyword VARDRV-D. This is the default.
CCA Output Key Usage Subgroups (One keyword from one CCA output key usage subgroup shown in the following table is required based on TR-31 input key usage.) **Note:** None of the following keywords are allowed if the TR-31 key block provided as input has an optional block that contains a CCA control vector. If the TR-31 key block header contains an optional block with a control vector in it, the control vector is used in place of keywords to produce the output CCA key-token. If the key usage and mode of use fields of the key block are not IBM-defined, the control vector must not conflict with any TR-31 header fields.	
C0 Subgroup (One Required for this TR-31 key usage)	
CVK-CVV	Convert TR-31 CVK to a CCA key for use with CVV/CVC. The CCA key will be a MAC key with subtype CVVKEY-A.
CVK-CSC	Convert TR-31 CVK to a CCA key for use with CSC. The CCA key will be a MAC key with subtype AMEX CSC.
K0 Subgroup (One Required for this TR-31 key usage)	
EXPORTER	For TR-31 K0-E or K0-B usage+mode keys. Convert TR-31 KEK to a CCA wrapping key. The key will convert to a CCA EXPORTER key. Note that the K0-B key import has a unique ACP.
OKEYXLAT	For TR-31 K0-E or K0-B usage+mode keys. Convert TR-31 KEK to a CCA wrapping key. The key will convert to a CCA OKEYXLAT key. Note that the K0-B key import has a unique ACP.
IMPORTER	For TR-31 K0-D or K0-B usage+mode keys. Convert TR-31 KEK to a CCA unwrapping key. The key will convert to a CCA IMPORTER key. Note that the K0-B key import has a unique ACP.
IKEYXLAT	For TR-31 K0-D or K0-B usage+mode keys. Convert TR-31 KEK to a CCA unwrapping key. The key will convert to a CCA IKEYXLAT key. Note that the K0-B key import has a unique ACP.
V0/V1/V2 Subgroup (One Required for these TR-31 key usages)	
PINGEN	Convert a TR-31 PIN verification key to a CCA PINGEN key.
PINVER	Convert a TR-31 PIN verification key to a CCA PINVER key.
E0/E2,F0/F2 Subgroup (One Required for these TR-31 key usages)	

Table 397. Keywords for TR-31 Import Rule Array Control Information (continued)	
Keyword	**Meaning**
DMAC	Convert TR-31 EMV master key (chip card or issuer) for Application Cryptograms or Secure Messaging for Integrity to CCA DKYGENKY type DMAC
DMV	Convert TR-31 EMV master key (chip card or issuer) for Application Cryptograms or Secure Messaging for Integrity to CCA DKYGENKY type DMV
E1,F1 Subgroup (One Required for these TR-31 key usages)	
DMPIN	Convert TR-31 EMV master key (chip card or issuer) for Secure Messaging for Confidentiality to CCA DKYGENKY type DMPIN
DDATA	Convert TR-31 EMV master key (chip card or issuer) for Secure Messaging for Confidentiality to CCA DKYGENKY type DDATA
E5 Subgroup (One Required for this TR-31 key usage)	
DMAC	Convert TR-31 EMV master key (issuer) for Card Personalization to CCA DKYGENKY type DMAC.
DMV	Convert TR-31 EMV master key (issuer) for Card Personalization to CCA DKYGENKY type DMV.
DEXP	Convert TR-31 EMV master key (issuer) for Card Personalization to CCA DKYGENKY type DEXP.
P0 Subgroup (One Optional for these TR-31 key usages)	
OPINENC	Convert a TR-31 "PIN encryption" key with mode of use 'B' to a CCA OPINENC key.
IPINENC	Convert a TR-31 "PIN encryption" key with mode of use 'B' to a CCA IPINENC key.
Key Derivation Level (One Required with E0, E1, E2 TR-31 key usages unless the CV is included in the TR-31 key block as an optional block. If the CV is included in the TR-31 key block, the included CV will be used in the output key block as long as it does not conflict with the TR-31 header data.)	
DKYL0	Convert TR-31 EMV master key (chip card or issuer) to CCA DKYGENKY at derivation level DKYL0.
DKYL1	Convert TR-31 EMV master key (chip card or issuer) to CCA DKYGENKY at derivation level DKYL1.
Key Type Modifier (Optional)	
NOOFFSET	Valid only for V0/V1 TR-31 key usage values. Import the PINGEN or PINVER key into a key token that cannot participate in the generation or verification of a PIN when an offset or the Visa PVV process is requested.
Key Wrapping Method (Optional) **Note:** Conflicts between wrapping keywords used and a CV passed in an optional data block of the TR-31 token will result in errors being returned. The main example of this is a CV that indicates 'enhanced-only' in bit 56 when the user or configured default specifies ECB for key wrapping.	
USECONFG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.

Table 397. Keywords for TR-31 Import Rule Array Control Information (continued)	
Keyword	**Meaning**
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This is the default for triple-length keys.
WRAPENH3	Specifies to wrap the key using the enhanced wrapping method and SHA-256 and CMAC authentication code.
WRAP-ECB	Specifies that the original wrapping method is to be used.
Translation Control (One Optional)	
ENH-ONLY	Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the original method. This restricts translation to the original method. This is the default when the wrapping method is WRAPENH2 or WRAPENH3. If the keyword is not specified, translation to the original method will be allowed. This turns on bit 56 in the control vector. This keyword is not valid if processing a zero CV data key. **Note:** 1. If the TR-31 block contains a CV in the optional data block that does not have bit 56 turned on, bit 56 will be turned on in the output token, since with this keyword the user is asking for this behavior. The exception to this is for CVs of all 0x00 bytes, for this case no error will be generated but the CV will remain all 0x00 bytes. 2. Conflicts between wrapping keywords used and a CV passed in an optional data block of the TR-31 token will result in errors being returned. The main example of this is a CV that indicates 'enhanced-only' in bit 56 when the user or configured default specifies ECB for key wrapping. If the default wrapping method is ECB mode, but the enhanced mode and the ENH-ONLY restriction are desired for a particular key token, combine the ENH-ONLY keyword with the WRAP-ENH keyword.
CPACF export (optional)	
XPRTCPAC	Allows export to CPACF protected key format. This keyword is valid only when the output key token is a DES CIPHER key token.

HMAC hash algorithm limit for key usage "M7" and algorithm "H" (one, optional). Not allowed with any other key usage.

Security note: ISO 20038 and ANSI X9 TR-31-2018 represent the HMAC hash algorithm limit in different ways:

- ISO 20038 represents hash limit in the algorithm value at offset 7. An HMAC key limited to SHA-1 uses ASCII 'H' and does not contain an optional block 'HM'.
- ANSI X9 TR-31-2018 always uses 'H' for the algorithm value at offset 7 and represents the hash algorithm limit in the optional block with identifier 'HM'.
- The ISO 'H' algorithm key block has a dual meaning:
  - For an ISO 20038 implementation, the resulting key block is limited to SHA-1 hash MAC.
  - For an ANSI X9 TR-31-2018 implementation, the key does not have any hash algorithm limit because the optional block with identifier 'HM' is not present.

Table 397. Keywords for TR-31 Import Rule Array Control Information (continued)	
Keyword	**Meaning**
HMAC-ISO	This keyword specifies to import an HMAC key block according to the ISO 20038. Interpretation is as follows: • If the key block contains an optional block with identifier 'HM', it will be ignored. • Import of a key block with 'H' for the algorithm value at offset 7 will result in the *output_key_identifier* holding an HMAC key in a Version X'05' variable-length symmetric key-token that allows a hash method of SHA-1 (KUF2 HOB = B'1xxx xxxx'). • Import of a key block with 'I' for the algorithm value at offset 7 will result in the *output_key_identifier* holding an HMAC key in a Version X'05' variable-length symmetric key-token that allows all SHA-2 hash methods (KUF2 HOB = B'x111 1xxx' for SHA-224, SHA-256, SHA-384, and SHA-512).
HMAC-X9	This keyword specifies to import an HMAC key block according to the ANSI X9 TR-31-2018. Interpretation is as follows: • If the key block does not have 'H' for the algorithm value at offset 7, 8/2121 will be returned. If the key block does not contain an optional block with identifier 'HM', new error 8/2182 will be returned. • Import will result in the *output_key_identifier* holding an HMAC key in a Version X'05' variable-length symmetric key-token that allows the hash method in the KUF2 HOB that matches the specification in the optional block with identifier 'HM'.
HMAC-UNK	This keyword specifies for the HSM to inspect the key block to determine the appropriate hash algorithm limit for the HMAC key. Interpretation is as follows: • Import of a key block with 'H' for the algorithm value at offset 7 and with an optional block with identifier 'HM' will result in the *output_key_identifier* holding an HMAC key in a Version X'05' variable-length symmetric key-token that allows the hash method in the KUF2 HOB that matches the specification in the optional block with identifier 'HM'. • Import of a key block with 'H' for the algorithm value at offset 7 and without an optional block with identifier 'HM' will result in the *output_key_identifier* holding an HMAC key in a Version X'05' variable-length symmetric key-token that allows hash methods of SHA-1 and SHA-2 (KUF2 HOB = B'1111 1xxx' for SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512). • Import of a key block with 'I' for the algorithm value at offset 7 will result in the *output_key_identifier* holding an HMAC key in a Version X'05' variable-length symmetric key-token that allows all SHA-2 hash methods (KUF2 HOB = B'x111 1xxx' for SHA-224, SHA-256, SHA-384, and SHA-512). This is the default.

Table 398 on page 965 shows all the valid translations for the import of a TR-31 BDK base derivation key (usage "B0") to either a CCA KEYGENKY key or CCA DKYGENKY key, along with any access controls that must be enabled in the domain role for that key type and control vector attributes. These keys are for translating derived unique key per transaction (DUKPT) base derivation keys.

Table 398. Import translation table for a TR-31 BDK base derivation key (usage "B0")

Key usage	Key block protection method (version ID)	Mode of use	Rule-array keywords	CCA key type and key usage	Access control name	Offset (hex)
"B0"	"A", "B", "C", or "D"	"N"	N/A	KEYGENKY, double length, UKPT (CV bit 18 = B'1')	N/A	N/A
	"B", "C", or "D"	"X"				
	"D"	"X"	N/A	DKYGENKY, A-DUKPT KUF set to 1.	T31I - Permit B0:X to AES DKYGENKY:DUKPT BDK	017E
"B1"	"B", "C", or "D"	"X"	N/A	DES KEYGENKY double length, UKPT (CV bit 18 = B'1')	T31I - Permit B1 to DES KEYGENKY:DUKPT	03E8
"B3"	"B", "C", or "D"	"X"	N/A	DES DKYGENKY	T31I – Permit B3 to DES DKYGENKY	03E9

Notes:

1. These are the base keys from which derived unique key per transaction (DUKPT) initial keys are derived for individual devices such as PIN pads.
2. The following defines the only supported translations for this TR-31 usage. Usage must be the following:

   "B0"
   BDK base derivation key.

   "B1"
   Initial DUKPT derivation key.

   "B3"
   Key derivation key.

Table 399 on page 966 shows all the valid translations for the import of a TR-31 CVK card verification key (usage "C0") to a CCA MAC or DATA key, along with any access controls that must be enabled in the domain role for that key type and control vector attributes. These keys are for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2, and CVV2 algorithm.

Table 399. Import translation table for a TR-31 CVK card verification key (usage "C0")

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"C0"	"A", "B", "C", or "D"	"G" or "C"	CVK-CSC	MAC, single or double length, AMEX-CSC (CV bits 0 - 3 = B'0100')	T31I – Permit C0:G/C/V to DES MAC/MACVER:AMEX-CSC	015B
			CVK-CVV	MAC, double length, CVVKEY-A (CV bits 0 - 3 = B'0010')	T31I - Permit C0:G/C/V to DES MAC/MACVER:CVVKEY-A	015A
		"V"	CVK-CSC	MACVER, single or double length, AMEX-CSC (CV bits 0 - 3 = B'0100')	T31I – Permit C0:G/C/V to DES MAC/MACVER:AMEX-CSC	015B
			CVK-CVV	MACVER, double length, CVVKEY-A (CV bits 0 - 3 = B'0010')	T31I - Permit C0:G/C/V to DES MAC/MACVER:CVVKEY-A	015A

Security considerations:

1. There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "C0" key, provided that one or both applicable MAC generate and MAC verify control vector bits are on. However, a TR-31 "C0" key cannot be imported to the lower-security CCA DATA key, it can only be imported to a CCA key type of MAC or MACVER. This restriction eliminates the ability to export a CCA MAC or MACVER key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to Encipher, Decipher, or both.
2. Since the translation from TR-31 usage "C0" is controlled by rule array keywords when using the TR31 Key Import service, it is possible to convert an exported CCA CVVKEY-A or CVVKEY-B key into an AMEX-CSC key or the other way around. This can be restricted by not enabling access controls X'015A' (T31I - Permit C0:G/C/V to DES MAC/MACVER:CVVKEY-A) and X'015B' (T31I - Permit C0:G/C/V to DES MAC/MACVER:AMEX-CSC) at the same time. However, if both CVVKEY-x and AMEX-CSC translation types are required, then offsets X'015A' and X'015B' must be enabled. In this case, control is up to the development, deployment, and execution of the applications themselves.

Notes:

1. Card verification keys are used for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2, and CVV2 algorithms. In CCA, this corresponds to keys used with two algorithms:
   - Visa CVV and MasterCard CVC codes are generated and verified using the CVV Generate and CVV Verify services. These services require a key type of DATA or MAC/MACVER with a subtype extension (CV bits 0 - 3) of ANY-MAC, single-length CVVKEY-A and single-length CVVKEY-B, and a double-length CVVKEY-A (see CVV Key Combine service). The MAC generate and the MAC verify (CV bits 20 - 21) key usage values must be set appropriately.
   - American Express CSC codes are generated and verified using the Transaction Validation service. This service requires a key type of MAC or MACVER with a subtype extension of ANY-MAC or AMEX-CSC.
2. The translation from TR-31 usage "C0" to a CCA MAC/MACVER key with a subtype extension of ANY-MAC (CV bits 0 - 3 = B'0000') is not allowed.
3. The following defines the only supported translations for this TR-31 usage. Usage must be the following:

   "C0"
   CVK card verification key.

4. CCA does not have an equivalent to the TR-31 "generate only" mode of use, so a translation from TR-31 mode "G" will result in a CCA MAC key with both MAC generate and MAC verify attributes (CV bits 20 - 21 = B'11'). Note that any key that can perform a generate can readily verify a MAC as well.
5. The CCA representation and the TR-31 representation of CVV keys are incompatible. CCA represents the CVVKEY-A and CVVKEY-B keys as two 8-byte (single length) keys, while TR-31 represents these keys as one 16-byte key. The CVV_Generate and CVV_Verify verbs accept one 16-byte CVV key, using left and right key parts as A and B. Current Visa standards require this.
6. Import and export of 8-byte CVVKEY-A and CVVKEY-B MAC/MACVER keys will only be allowed using the IBM proprietary TR-31 usage and mode values ("10" and "1", respectively) to indicate encapsulation of the IBM control vector in an optional block, since the 8-byte CVVKEY-A is meaningless and useless as a TR-31 "C0" usage key of any mode.

Table 400 on page 967 shows all valid translations for import of a TR-31 data encryption key (usage "D0") to a CCA ENCIPHER, DECIPHER, CIPHER, or DATA key, along with any access controls that must be enabled in the domain role for that key type and control vector attributes. These keys are used for the encryption and/or decryption of data.

Table 400. Import translation table for a TR-31 data encryption key (usage "D0")

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
ENC "D0"	"A", "B", "C", or "D"	"E"	N/A	DES ENCIPHER, single, double, or triple length	N/A	N/A
		"D"		DES DECIPHER, single, double, or triple length		
		"B"		DES CIPHER, single, double, or triple length		
	"D"	"E"		AES CIPHER, ENCRYPT	T31I - Permit D0:E/D/B to AES CIPHER:ENC/DEC /ENC+DEC	01E0
		"D"		AES CIPHER, DECRYPT		
		"B"		AES CIPHER, ENCRYPT and DECRYPT		
ENCSENS (“D3”)	“B” or “D”	"E"	N/A	DES CIPHERXO	T31I – Permit D3 to CIPHER:XLATE	03EA
		“D”	N/A	DES CIPHERXI		
		"B"	N/A	DES CIPHERXL		
	“D”	"E"	N/A	AES CIPHER, XLATE, ENCRYPT key usage attributes enabled		
		“D”	N/A	AES CIPHER, XLATE, DECRYPT key usage attributes enabled		
		"B"	N/A	AES CIPHER, XLATE, ENCRYPT, DECRYPT key usage attributes enabled		

Table 400. Import translation table for a TR-31 data encryption key (usage "D0") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)

Security considerations:

- There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "D0" key, provided that one or both applicable Encipher or Decipher control vector bits are on. However, a TR-31 "D0" key cannot be imported to the lower-security CCA DATA key, it **can only be imported** to a CCA key type of ENCIPHER, DECIPHER, or CIPHER. This restriction eliminates the ability to export a CCA DATA key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to MAC generate and MAC verify.

Notes:

1. Data encryption keys are used for the encryption and decryption of data.
2. The following defines the only supported translations for this TR-31 usage. Usage must be the following:

   "D0"
   Data encryption.

   "D3"
   Cipher text translation.
3. There are no specific access-control commands for this translation since it is not ambiguous or in need of interpretation.

Table 401 on page 969 shows all valid translations for import of a TR-31 data encryption key (usage "F0") to a CCA ENCIPHER, DECIPHER, CIPHER, or DATA key, along with any access controls that must be enabled in the domain role for that key type and control vector attributes. These keys are used for the encryption and/or decryption of data.

Table 401. Import translation table for a TR-31 data encryption key (usage "F0")

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"F0"	"A"	"N"	DMAC	DES DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DMAC (CV bits 19- 22 = B'0010')	T31I – Permit F0:N/X to DES DKYGENKY:DKYL0+DMAC	03EB
	"B", "C", or "D"	"X" or "N"				
	"A"	"N"	DMV	DES DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DMV (CV bits 19 -22 = B'0011')	T31I – Permit F0:N/X to DES DKYGENKY:DKYL0+DMV	03EC
	"B", "C", or "D"	"X" or "N"				
	"D"	"X"	N/A	AES DKEYGENKY, D-MAC usage attributes to match keywords.	T31I – Permit F0:X to AES DKYGENKY:DKYL0+D-MAC+GENERATE+CMAC	0504
"F1"	"A"	"N", "E", "D", or "B"	DMPIN	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMPIN (CV bits 19 - 22 = B'1001')	T31I - Permit F1:N/E/D/B/X to DES DKYGENKY:DKYL0+DMPIN	03ED
	"B", "C", or "D"	"X" or "N"				
	"A"	"N", "E", "D", or "B"	DDATA	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')	T31I - Permit F1:N/E/D/B/X to DES DKYGENKY:DKYL0+DDATA	03EE
	"B", "C", or "D"	"X" or "N"				
	"D"	"X"	N/A	AES DKYGENKY, D-SECMSG+ SMPIN + ANY-USE usage attributes to match keywords.	T31I - Permit F1:X to AES DKYGENKY:DKYL0+D-SECMSG+SMPIN+ANY-USE	0505

Table 401. Import translation table for a TR-31 data encryption key (usage "F0") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"F2"	"A"	"N", "E", "D", or "B"	DMAC	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')	T31I - Permit F2:N/X to DES DKYGENKY:DKYL0+DMAC	03EF
	"B", "C", or "D"	"X" or "N"				
	"D"	"X"	N/A	AES DKYGENKY, D-MAC usage attributes to match keywords	T31I - Permit F2:X to AES DKYGENKY:DKYL0+D-MAC+GENERATE+CMAC	0506
"F3"	"A"	"N", "E", "D", "B", or "G"	N/A	DES ENCIPHER	T31I - Permit F3:N/E/D/B/G/X to DES ENCIPHER	0502
	"B", "C", or "D"	"X" or "N"				
	"D"	"X"	N/A	AES DKYGENKY, D-CIPHER usage attributes to match keywords.	T31I - Permit F3:X to AES DKYGENKY:D-CIPHER+ENCRYPT+DECRYPT+CBC	0507
	"D"	"E"		AES CIPHER, ENCRYPT	T31I - Permit F3:E/B to AES CIPHER:ENCRYPT/ ENCRYPT+DECRYPT	0508
		"B"		AES CIPHER, ENCRYPT, DECRYPT		
"F4"	"A"	"N" or "B"	N/A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')	T31I - Permit F4:N/B/X to DES DKYGENKY:DKYL0+DDATA	0503
	"B", "C", or "D"	"X" or "N"				
	"D"	"X"		AES DKYGENKY, D-CIPHER usage attributes to match keywords	T31I - Permit F4:X to AES DKYGENKY:DKYL0+D-CIPHER+ENC+DEC+CBC	0509

Table 401. Import translation table for a TR-31 data encryption key (usage "F0") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)

Notes:

1. EMV/chip issuer master-key keys are used by the chip cards to perform cryptographic operations or, in some cases, to derive keys used to perform operations. In CCA, these are (a) diversified key-generating keys (key type DKYGENKY), allowing derivation of operational keys, or (b) operational keys. Note that in this context, "master key" has a different meaning than for CCA. These master keys, also called KMCs, are described by EMV as DES master keys for personalization session keys. They are used to derive the corresponding chip card master keys, and not typically used directly for cryptographic operations other than key derivation. In CCA, these are usually key generating keys with derivation level DKYL1 (CV bits 12 - 14 = B'001'), used to derive other key generating keys (the chip card master keys). For some cases, or for older EMV key derivation methods, the issuer master keys could be level DKYL0 (CV bits 12 - 14 = B'000').
2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:

   "F0"
   Application cryptograms.

   "F1"
   Secure messaging for confidentiality.

   "F2"
   Secure messaging for integrity.

   "F3"
   Data authentication code.

   "F4"
   Dynamic numbers.
3. EMV support in CCA is quite different than TR-31 support, and CCA key types do not match TR-31 types.
4. DKYGENKY keys are double length only.
5. In CCA, a MAC key that can perform a MAC Generate operation also can perform a MAC Verify. For TR-31 mode "G" (generate only), the translation to a CCA key results in a key that can perform MAC Generate and MAC Verify.

Table 402 on page 972 shows all valid translations for import of a TR-31 key encryption or wrapping, or key block protection key (usages "K0", "K1") to a CCA EXPORT, OKEYXLAT, IMPORTER, or IKEYXLAT key, along with any access controls that must be enabled in the domain role for that key type and control vector attributes. These keys are used only to encrypt or decrypt other keys, or as a key used to derive keys that are used for that purpose.

Table 402. Import translation table for a TR-31 key encryption or wrapping, or key block protection key (usages "K0", "K1")

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"K0"	"A", "B", "C", or "D"	"E"	OKEYXLAT	OKEYXLAT, double length	T31I – Permit K0:E to DES EXPORTER/ OKEYXLAT	015C
			EXPORTER	EXPORTER, double or triple length, EXPORT on (CV bit 21 = B'1')		
		"D"	IKEYXLAT	IKEYXLAT, double length	T31I - Permit K0:D to DES IMPORTER/ IKEYXLAT	015D
			IMPORTER	IMPORTER, double or triple length, IMPORT on (CV bit 21 = B'1')		
		"B"	OKEYXLAT	OKEYXLAT, double length	T31I - Permit K0:B to DES EXPORTER/ OKEYXLAT	015E
			EXPORTER	EXPORTER, double or triple length, EXPORT on (CV bit 21 = B'1')		
			IKEYXLAT	IKEYXLAT, double length	T31I - Permit K0:B to DES IMPORTER/ IKEYXLAT	015F
			IMPORTER	IMPORTER, double or triple length, IMPORT on (CV bit 21 = B'1')		
	"D"	"E"	N/A	AES EXPORTER	T31I - Permit K1/K4:E to AES EXPORTER: EXPTT31D+ VARDRV-D	01E5
		"D"		AES IMPORTER	T31I - Permit AES K1/K4:D to AES IMPORTER: IMPTT31D+ VARDRV-D	01E6
"K1"	"B", "C", or "D"	"E"	OKEYXLAT	OKEYXLAT, double length	T31I - Permit K1/K4:E to DES EXPORTER/ OKEYXLAT	0160
			EXPORTER	EXPORTER, double or triple length, EXPORTER/OKEYXLAT EXPORT on (CV bit 21 = B'1')		
		"D"	IKEYXLAT	IKEYXLAT, double length	T31I - Permit K1/K4:D to DES IMPORTER/ IKEYXLAT	0161
			IMPORTER	IMPORTER, double or triple length, to IMPORTER/IKEYXLAT IMPORT on (CV bit 21 = B'1')		
		"B"	OKEYXLAT	OKEYXLAT, double length	T31I - Permit K1/K4:B to DES EXPORTER/ OKEYXLAT	0162
			EXPORTER	EXPORTER, double or triple length, EXPORTER/OKEYXLAT EXPORT on (CV bit 21 = B'1')		
			IKEYXLAT	IKEYXLAT, double length	T31I - Permit K1/K4:B to DES IMPORTER/ IKEYXLAT	0163
			IMPORTER	IMPORTER, double or triple length, to IMPORTER/IKEYXLAT IMPORT on (CV bit 21 = B'1')		
	"D"	"E"	N/A	AES EXPORTER, EXPTT31D	T31I - Permit K1/K4:E to AES EXPORTER: EXPTT31D+ VARDRV-D	01E5
		"D"		AES IMPORTER. IMPTT31D	T31I - Permit AES K1/K4:D to AES IMPORTER: IMPTT31D+ VARDRV-D	01E6
"K4"	"D"	"E"	N/A	AES EXPORTER, EXPTT31D	T31I - Permit K1/K4:E to AES EXPORTER: EXPTT31D+ VARDRV-D	01E5
		"D"		AES IMPORTER. IMPTT31D	T31I - Permit AES K1/K4:D to AES IMPORTER: IMPTT31D+ VARDRV-D	01E6

Table 402. Import translation table for a TR-31 key encryption or wrapping, or key block protection key (usages "K0", "K1") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)

Security considerations:

1. The CCA OKEYXLAT, EXPORTER, IKEYXLAT, or IMPORTER KEK translation to a TR-31 "K0" key with mode "B" (both wrap and unwrap) is not allowed for security reasons. Even with access-control point control, this capability would give an immediate path to turn a CCA EXPORTER key into a CCA IMPORTER key, and the other way around.
2. When a TR-31 key block does not have an included control vector as an optional block, the default control vector will be used to construct the output key-token. Default CCA EXPORTER or IMPORTER keys have CV bits 18 - 20 on, which are used for key generation.

Notes:

1. Key encryption or wrapping keys are used only to encrypt or decrypt other keys, or as a key used to derive keys that are used for that purpose.
2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:

   "K0"
   Key encryption or wrapping.

   "K1"
   TR-31 key block protection key.
3. Any attempt to import a TR-31 "K0" or "K1" key that has algorithm "D" (DEA) will result in an error because CCA does not support single-length KEKs.
4. CCA mode support is the same for version IDs "B" and "C". That is because the distinction between TR-31 "K0" and "K1" does not exist in CCA keys. CCA does not distinguish between targeted protocols, and so there is no good way to represent the difference. Also note that most wrapping mechanisms now involve derivation or key variation steps.

Table 403 on page 974 shows all valid translations for import of a TR-31 ISO MAC algorithm key (usages "M0", "M1", "M3") to a CCA MAC, MACVER, DATA, DATAM, or DATAMV key, along with any access control commands that must be enabled in the active role for that key type and control vector attributes. These keys are used to compute or verify a code for message authentication.

Table 403. Import translation table for a TR-31 ISO MAC algorithm key (usages "M0", "M1", "M3")

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"M0"	"A", "B", "C", or "D"	"G" or "C"	N/A	MAC, double or triple length, ANY-MAC (CV bits 0 - 3 = B'0000')	T31I - Permit M0/M1/ M3:G/C/V to DES MAC/ MACVER: ANY-MAC	0164
		"V"		MACVER, double or triple length, ANY-MAC (CV bits 0 - 3 = B'0000')		
"M1"		"G" or "C"		MAC, single, double, or triple length, ANY-MAC (CV bits 0 - 3 = B'0000')		
		"V"		MACVER, single, double, or triple length, ANY-MAC (CV bits 0 - 3 = B'0000')		
"M3"		"G" or "C"		MAC, single, double, or triple length, ANY-MAC (CV bits 0 - 3 = B'0000')		
		"V"		MACVER, single, double, or triple length, ANY-MAC (CV bits 0 - 3 = B'0000')		

Table 403. Import translation table for a TR-31 ISO MAC algorithm key (usages "M0", "M1", "M3") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"M6"	“B”, “C”, or “D”	"G" or "C"	N/A	MAC, double, or triple length, ANY-MAC (CV bits 0 - 3 =B'0000')	T31I - Permit M6 to DES MAC	03F0
		"V"		MACVER, double, or triple length, ANY-MAC (CV bits 0 - 3 =B'0000')		
	"D"	"G" or "C"	N/A	AES MAC (CMAC), GENERATE or GENONLY	T31I - Permit M6:G/C/V to AES MAC: CMAC+ GENONLY/ GEN/ VER	01E1
		"V"		AES MAC (CMAC), VERIFY		

Security considerations: There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "M0", "M1", or "M3" key, provided that one or both applicable MAC generate and MAC verify control vector bits are on. However, a TR-31 "M0", "M1", or "M3" key cannot be imported to the lower-security CCA DATA key, it can only be imported to a CCA key type of MAC or MACVER. This restriction eliminates the ability to export a CCA MAC or MACVER key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to Encipher, Decipher, or both.

Notes:

1. MAC keys are used to compute or verify a code for message authentication.
2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:

   "M0" ISO 16609 MAC algorithm 1, TDEA
   The ISO 16609 MAC algorithm 1 is based on ISO 9797. It is identical to "M1", except that it does not support 8-byte DES keys.

   "M1" ISO 9797 MAC algorithm 1
   The ISO 9797 MAC algorithm 1 is identical to "M0", except that it also supports 8-byte DES keys.

   "M3" ISO 9797 MAC algorithm 3
   This is the X9.19 style of Triple-DES MAC.
3. A CCA control vector has no bits defined to limit key usage by algorithm, such as CBC MAC (TR-31 usage "M0" and "M1") or X9.19 (TR-31 usage "M3"). When importing a TR-31 key block, the resulting CCA key token deviates from the restrictions of usages "M0", "M1", and "M3". Importing a TR-31 key block which allows MAC generation ("G" or "C") results in a control vector with the ANY-MAC attribute rather than for the restricted algorithm that is set in the TR-31 key block. The

ANY-MAC attribute provides the same restrictions as what CCA currently uses for generating and verifying MACs.

Table 404 on page 976 shows all the valid translations for import of a TR-31 HMAC algorithm key (usages "M7") to a CCA HMAC key, along with any access control commands that must be enabled in the active role for that key type and control vector attributes. These keys are used to compute or verify a code for message authentication.

Table 404. Import translation table for a TR-31 HMAC algorithm key (usages "M7")

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required key usage attributes	Access control name	Offset (hex)
"M7"	"D"	"G" or "C"	N/A	MAC + HMAC + GENERATE	T31I – Permit M7:G/V/C to HMAC MAC: GENERATE/ VERIFY	017D
		"V"		MAC + HMAC + VERIFY		

Table 405 on page 976 shows all valid translations for import of a TR-31 PIN encryption or PIN verification key (usages "P0", "V0", "V1", "V2") to a CCA OPINENC, IPINENC, PINGEN, or PINVER key, along with any access controls that must be enabled in the domain role for that key type and control vector attributes. These keys are used to protect PIN blocks and to generate or verify a PIN using a particular PIN-calculation method for that key type.

Table 405. Import translation table for a TR-31 PIN encryption or PIN verification key (usages "P0", "V0", "V1", "V2")

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"P0"	"A", "B", "C", or "D"	"E"	N/A	OPINENC, double or triple length	T31I - Permit P0:E to DES OPINENC	0165
			OPINENC	OPINENC, double or triple length		
		"D"	N/A	IPINENC, double or triple length	T31I - Permit P0:D to DES IPINENC	0166
			IPINENC	IPINENC, double or triple length		
		"B"	OPINENC	OPINENC, double or triple length	T31I - Permit P0:E to DES OPINENC	0165
			IPINENC	IPINENC, double or triple length	T31I - Permit P0:D to DES IPINENC	0166
	"D"	"D"	N/A	AES PINPROT, DECRYPT	T31I - Permit P0:E/D to AES PINPROT: ENC/DEC+ CBC+ISO-4	01E2
			IPINENC	AES PINPROT, DECRYPT		
		"E"	N/A	AES PINPROT, ENCRYPT		
			OPINENC	AES PINPROT, ENCRYPT		
		"B"	OPINENC	AES PINPROT, ENCRYPT		
			IPINENC	AES PINPROT, DECRYPT		

Table 405. Import translation table for a TR-31 PIN encryption or PIN verification key (usages "P0", "V0", "V1", "V2") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes		Access control name	Offset (hex)
"V0"	"A", "B", "C", or "D"	"N" (requires both controls)	PINGEN	PINGEN, double or triple length, NO-SPEC (CV bits 0 - 3 = B'0000')	NOOFFSET off (CV bit 37 = B'0')	T31I - Permit V0:N/G/C to DES PINGEN: NO-SPEC NOOFFSET	0167
						T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"G" or "C"			NOOFFSET off (CV bit 37 = B'0')	T31I - Permit V0:N/G/C to DES PINGEN:NO-SPEC NOOFFSET	0167
	"A", "B", "C", or "D"	"N" (requires both controls)	PINGEN, NOOFFSET		NOOFFSET on (CV bit 37 = B'1')	T31I - Permit V0:N/G/C to DES PINGEN:NO-SPEC NOOFFSET	0167
						T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"G" or "C"			NOOFFSET on (CV bit 37 = B'1')	T31I - Permit V0:N/G/C to DES PINGEN:NO-SPEC NOOFFSET	0167
	"A", "B", "C", or "D"	"N" (requires both controls)	PINVER	PINVER, double or triple length, NO-SPEC (CV bits 0 - 3 = B'0000')	NOOFFSET off (CV bit 37 = B'0')	T31I - Permit V0:N/V to DES PINVER:NO-SPEC NOOFFSET	0168
						T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"V" or "C"			NOOFFSET off (CV bit 37 = B'0')	T31I - Permit V0:N/V to DES PINVER:NO-SPEC NOOFFSET	0168
	"A", "B", "C", or "D"	"N" (requires both controls)	PINVER, NOOFFSET		NOOFFSET on (CV bit 37 = B'1')	T31I - Permit V0:N/V to DES PINVER:NO-SPEC NOOFFSET	0168
						T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"V" or "C"			NOOFFSET on (CV bit 37 = B'1')	T31I - Permit V0:N/V to DES PINVER:NO-SPEC NOOFFSET	0168
"V1"	"A", "B", "C", or "D"	"N" (requires both controls)	PINGEN	PINGEN, double or triple length, IBM PIN/IBM-PINO (CV bits 0 - 3 = B'0001')	NOOFFSET off (CV bit 37 = B'0')	T31I - Permit V1:N/G/C to DES PINGEN:IBM-PIN/ IBM-PINO NOOFFSET	0169
						T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"G" or "C"			NOOFFSET off (CV bit 37 = B'0')	T31I - Permit V1:N/G/C to DES PINGEN:IBM-PIN/ IBM-PINO NOOFFSET	0169
	"A", "B", "C", or "D"	"N" (requires both controls)	PINGEN, NOOFFSET		NOOFFSET on (CV bit 37 = B'1')	T31I - Permit V1:N/G/C to DES PINGEN:IBM-PIN/ IBM-PINO NOOFFSET	0169
						T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"G" or "C"			NOOFFSET on (CV bit 37 = B'1')	T31I - Permit V1:N/G/C to DES PINGEN:IBM-PIN/ IBM-PINO NOOFFSET	0169
	"A", "B", "C", or "D"	"N" (requires both controls)	PINVER	PINVER, double or triple length, IBM PIN/IBM-PINO (CV bits 0 - 3 = B'0001')	NOOFFSET off (CV bit 37 = B'0')	T31I - Permit V1:N/V to DES PINVER:IBM-PIN/IBM-PINO NOOFFSET	016A
						T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"V" or "C"			NOOFFSET off (CV bit 37 = B'0')	T31I - Permit V1:N/V to DES PINVER:IBM-PIN/IBM-PINO NOOFFSET	016A
	"A", "B", "C", or "D"	"N" (requires both controls)	PINVER, NOOFFSET		NOOFFSET on (CV bit 37 = B'1')	T31I - Permit V1:N/V to DES PINVER:IBM-PIN/IBM-PINO NOOFFSET	016A
						T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"V" or "C"			NOOFFSET on (CV bit 37 = B'1')	T31I - Permit V1:N/V to DES PINVER:IBM-PIN/IBM-PINO NOOFFSET	016A

Table 405. Import translation table for a TR-31 PIN encryption or PIN verification key (usages "P0", "V0", "V1", "V2") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"V2"	"A", "B", "C", or "D"	"N" (requires both controls)	PINGEN	PINGEN, double or triple length, VISA-PVV (CV bits 0 - 3 = B'0010')	T31I - Permit V2:N/G/C to DES PINGEN:VISA-PVV	016B
					T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"G" or "C"		PINGEN, double or triple length, VISA-PVV (CV bits 0 - 3 = B'0010')	T31I - Permit V2:N/G/C to DES PINGEN:VISA-PVV	016B
	"A", "B", "C", or "D"	"N" (requires both controls)	PINVER	PINVER, double or triple length, VISA-PVV (CV bits 0 - 3 = B'0010')	T31I - Permit V2:N/V to DES PINVER:VISA-PVV	016C
					T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C
	"A", "B", "C", or "D"	"G" or "C"		PINVER, double or triple length, VISA-PVV (CV bits 0 - 3 = B'0010')	T31I - Permit V2:N/V to DES PINVER:VISA-PVV	016C

Security notes: TR-31 key blocks that are protected under legacy version ID "A" (keyword VARXOR-A, using the Key Variant Binding Method 2005 Edition) use the same mode of use "N" (keyword ANY) for PINGEN and PINVER keys. For version ID "A" keys only, for a given PIN key usage, enabling both the PINGEN and PINVER access controls at the same time while enabling offset X'01B0' (for mode "N") is NOT recommended. In other words, for a particular PIN verification usage, you should not simultaneously enable the four commands shown below for that usage:

Failure to comply with this recommendation allows changing PINVER keys into PINGEN and the other way around.

	Key type, mode, or version	Offset	Access control
For usage "V0", a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. Avoid simultaneously enabling these four controls.			
"V0"	Key type PINGEN	X'0167'	T31I - Permit V0:N/G/C to DES PINGEN:NO-SPEC NOOFFSET
	Key type PINVER	X'0168'	T31I - Permit V0:N/V to DES PINVER:NO-SPEC NOOFFSET
	Mode "N"	X'017C'	T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER
	Version "A"	X'0150'	T31I - Permit Version A TR-31 Key Blocks
For usage "V1", a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. Avoid simultaneously enabling these four controls.			
"V1"	Key type PINGEN	X'0169'	T31I - Permit V1:N/G/C to DES PINGEN:IBM-PIN/IBM-PINO NOOFFSET
	Key type PINVER	X'016A'	T31I - Permit V1:N/V to DES PINVER:IBM-PIN/IBM-PINO NOOFFSET
	Mode "N"	X'017C'	T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER
	Version "A"	X'0150'	T31I - Permit Version A TR-31 Key Blocks
For usage "V2", a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. Avoid simultaneously enabling these four controls.			

	Key type, mode, or version	Offset	Access control
"V2"	Key type PINGEN	X'016B'	T31I - Permit V2:N/G/C to DES PINGEN:VISA-PVV
	Key type PINVER	X'016C'	T31I - Permit V2:N/V to DES PINVER:VISA-PVV
	Mode "N"	X'017C'	T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER
	Version "A"	X'0150'	T31I - Permit Version A TR-31 Key Blocks

Notes:

1. PIN encryption keys are used to protect PIN blocks. PIN verification keys are used to generate or verify a PIN using a particular PIN-calculation method for that key type.
2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:

   "P0"
   PIN encryption.

   "V0"
   PIN verification, KPV, other algorithm. Usage "V0" does not have its own PIN-calculation method defined. The mapping to NO-SPEC is sub-optimal. Exporting to "N" mode restricts keys from being imported with the IBM-PIN/IBM-PINO or VISA-PVV attribute, while CCA NO-SPEC allows any method.

   "V1"
   PIN verification, IBM 3624.

   "V2"
   PIN verification, Visa PVV. The NOOFFSET keyword is not allowed for the Visa PVV algorithm because it does not support this attribute.
3. Mode must be one of the following:

   "E"
   Encrypt/wrap only. This restricts PIN encryption keys to encrypting a PIN block. May be used to create or reencipher an encrypted PIN block (for key-to-key translation).

   "D"
   Decrypt/unwrap only. This restricts PIN encryption keys to decrypting a PIN block. Generally used in a PIN translation to decrypt the incoming PIN block.

   "N"
   No special restrictions (other than restrictions implied by the key usage). This is used by several vendors for a PIN generate or PIN verification key when the key block version ID is "A".

   "G"
   Generate only. This is used for a PINGEN key that may not perform a PIN verification. The control vector will not have its EPINVER attribute on (CV bit 22 = B'0').

   "V"
   Verify only. This is used for PIN verification only. If the TR-31 key block does not have a control vector included, the only usage bits set on in the control vector will be the EPINVER bit (CV bits 18 - 22 = B'00001').

   "C"
   Both generate and verify (combined). The control vector will have the default PINGEN bits on (CV bits 18 -22 = B'11111').
4. Importing a TR-31 "P0" key that has mode "B" (both encrypt and decrypt) requires specifying either the OPINENC or IPINENC rule array keywords because CCA does not support PIN encryption keys that can both encrypt and decrypt.

5. If the TR-31 key block contains a control vector, and the control vector has NOOFFSET on, the NOOFFSET keyword is not necessary because the verb will automatically set NOOFFSET on in this case.

Table 406. Import translation table for a initialization vector (usage "I0")

Key usage	Key block protection method keyword (version ID)	Mode of use keyword	Initialization vector algorithm	Rule array keywords	Access control name
"I0"	"A", "B", "C", or "D"	"N"	"D", "T", or "A"	N/A	N/A

Table 407 on page 980 shows all valid translations for import of a TR-31 EMV/chip issuer master-key key (usages "E0", "E1", "E2", "E3", "E4", "E5") to a CCA DKYGENKY, DATA, MAC, CIPHER, or ENCIPHER key, along with any access controls that must be enabled in the domain role for that key type and control vector attributes. These keys are used by the chip cards to perform cryptographic operations or, in some cases, to derive keys used to perform operations.

Table 407. Import translation table for a TR-31 EMV/chip issuer master-key key (usages "E0", "E1", "E2", "E3", "E4", "E5")

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"E0"	"A"	"N"	TDKYL0, DMAC	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')	T31I - Permit E0:N/X to DES DKYGENKY:DKYL0+DMAC	016D
	"B", "C", or "D"	"X"				
		"N"				
	"A"	"N"	DKYL0, DMV	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMV (CV bits 19 - 22 = B'0011')	T31I - Permit E0:N/X to DES DKYGENKY:DKYL0+DMV	016E
	"B", "C", or "D"	"X"				
		"N"				
	"A"	"N"	DKYL1, DMAC	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMAC (CV bits 19 - 22 = B'0010')	T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMAC	016F
	"B", "C", or "D"	"X"				
		"N"				
	"A"	"N"	DKYL1, DMV	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMV (CV bits 19 - 22 = B'0011')	T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMV	0170
	"B", "C", or "D"	"X"				
		"N"				
	"D"	"X"	DKYL0 or DKYL1 or DKYL2	AES DKEYGENKY, D-MAC usage attributes to match keywords	T31I - Permit E0:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	01E7

Table 407. Import translation table for a TR-31 EMV/chip issuer master-key key (usages "E0", "E1", "E2", "E3", "E4", "E5") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"E1"	"A"	"N"	DKYL0, DMPIN	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMPIN (CV bits 19 - 22 = B'1001')	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DMPIN	0171
		"E"				
		"D"				
		"B"				
	"B", "C", or "D"	"X"				
		"N"				
	"A"	"N"	DKYL0, DDATA	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DDATA	0172
		"E"				
		"D"				
		"B"				
	"B", "C", or "D"	"X"				
		"N"				
	"A"	"N"	DKYL1, DMPIN	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMPIN (CV bits 19 - 22 = B'1001')	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DMPIN	0173
		"E"				
		"D"				
		"B"				
	"B", "C", or "D"	"X"				
		"N"				
	"A"	"N"	DKYL1, DDATA	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DDATA (CV bits 19 - 22 = B'0001')	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DDATA	0174
		"E"				
		"D"				
		"B"				
	"B", "C", or "D"	"X"				
		"N"				
	"D"	"X"	DKYL0 or DKYL1 or DKYL2	AES DKYGENKY, D-SECMSG + SMPIN + ANY-USE usage attributes to match keywords	T31I - Permit E1:X to AES DKYGENKY:DKYL0/L1/L2+D-SECMSG+SMPIN	01E8
"E2"	"A"	"N"	DKYL0, DMAC	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')	T31I - Permit E2:N/X to DES DKYGENKY:DKYL0+DMAC	0175
	"B", "C", or "D"	"X"				
		"N"				
	"A"	"N"	DKYL1, DMAC	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMAC (CV bits 19 - 22 = B'0010')	T31I - Permit E2:N/X to DES DKYGENKY:DKYL1+DMAC	0176
	"B", "C", or "D"	"X"				
		"N"				
	"D"	"X"	DKYL0 or DKYL1 or DKYL2	AES DKYGENKY, D-MAC usage attributes to match keywords	T31I - Permit E2:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	01E9

Table 407. Import translation table for a TR-31 EMV/chip issuer master-key key (usages "E0", "E1", "E2", "E3", "E4", "E5") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"E3"	"A"	"N"	N/A	ENCIPHER	T31I - Permit E3:N/E/D/B/G/X to DES ENCIPHER	0177
		"E"				
		"D"				
		"B"				
		"G"				
	"B", "C", or "D"	"X"				
		"N"				
	"D"	"X"	DKYL0 or DKYL1 or DKYL2	AES DKYGENKY, D-CIPHER usage attributes to match keywords	T31I - Permit E3:X to AES DKYGENKY:D-CIPHER+ENC+DEC+CBC	01EA
	"D"	"E"	N/A	AES CIPHER, ENCRYPT	T31I - Permit E3:E/B to AES CIPHER:ENCRYPT/ENC+DEC	01EB
		"B"		AES CIPHER, ENCRYPT, DECRYPT		
"E4"	"A"	"N"	N/A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')	T31I - Permit E4:N/B/X to DES DKYGENKY:DKYL0+DDATA	0178
		"B"				
	"B", "C", or "D"	"X"				
		"N"				
	"D"	"X"	DKYL0 or DKYL1 or DKYL2	AES DKYGENKY, D-CIPHER usage attributes to match keywords	T31I - Permit E4:X to AES DKYGENKY:DKYL0/L1/L2+D-CIPHER+ENC+DEC	01EC

Table 407. Import translation table for a TR-31 EMV/chip issuer master-key key (usages "E0", "E1", "E2", "E3", "E4", "E5") (continued)

Key usage	Key block protection method keyword (version ID)	Mode of use	Rule-array keywords	CCA key type and required control vector attributes	Access control name	Offset (hex)
"E5"	"A"	"G"	DKYL0, DMAC	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')	T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DMAC	0179
		"C"				
		"V"				
		"E"				
		"D"				
		"B"				
		"N"				
	"B", "C", or "D"	"X"				
		"N"				
	"A"	"G"	DKYL0, DDATA	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')	T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DDATA	017A
		"C"				
		"V"				
		"E"				
		"D"				
		"B"				
		"N"				
	"B", "C", or "D"	"X"				
		"N"				
	"A"	"G"	DKYL0, DEXP	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DEXP (CV bits 19 - 22 = B'0101')	T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DEXP	017B
		"C"				
		"V"				
		"E"				
		"D"				
		"B"				
		"N"				
	"B", "C", or "D"	"X"				
		"N"				
	"D"	"X"	DKYL0 or DKYL1 or DKYL2	AES DKYGENKY, D-MAC usage attributes to match keywords	T31I - Permit E5:X to AES DKYGENKY:DKYL0/L1/L2/D-MAC+GEN+CMAC	01ED

Notes:

1. EMV/chip issuer master-key keys are used by the chip cards to perform cryptographic operations or, in some cases, to derive keys used to perform operations. In CCA, these are (a) diversified key-generating keys (key type DKYGENKY), allowing derivation of operational keys, or (b) operational keys. Note that in this context, "master key" has a different meaning than for CCA. These master keys, also called KMCs, are described by EMV as DES master keys for personalization session keys. They are used to derive the corresponding chip card master keys,and not typically used directly for cryptographic operations other than key derivation. In CCA, these are usually key generating keys with derivation level DKYL1 (CV bits 12 - 14 = B'001'), used to derive other key generating keys

(the chip card master keys). For some cases, or for older EMV key derivation methods, the issuer master keys could be level DKYL0 (CV bits 12 - 14 = B'000').

2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:

   "E0"
   Application cryptograms.

   "E1"
   Secure messaging for confidentiality.

   "E2"
   Secure messaging for integrity.

   "E3"
   Data authentication code.

   "E4"
   Dynamic numbers.

   "E5"
   Card personalization.

3. EMV support in CCA is quite different than TR-31 support, and CCA key types do not match TR-31 types.
4. DKYGENKY keys are double length only.
5. In CCA, a MAC key that can perform a MAC Generate operation also can perform a MAC Verify. For TR-31 mode "G" (generate only), the translation to a CCA key results in a key that can perform MAC Generate and MAC Verify.

TR31_key_block_length

Direction	Type
Input	Integer

This parameter specifies the length of the *TR31_key_block* parameter, in bytes. The length field in the TR-31 block is a 4-digit decimal number, so the maximum acceptable length is 9992 bytes.

TR31_key_block

Direction	Type
Input	String

This parameter contains the TR-31 key block that is to be imported. The key block is protected with the key passed in parameter *unwrap_kek_identifier*.

unwrap_kek_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *unwrap_kek_identifier* parameter in bytes.

If the *unwrap_kek_identifier* contains a label, the value must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

unwrap_kek_identifier

Direction	Type
Input/Output	String

The identifier of the key to unwrap the TR-31 key block. The key identifier is an operational key token or key block or the label of an operational token or block in key storage.

When the source key block is wrapped by protection methods VARDRV-A, VARDRV-B, or VARDRV-C, the key is one of the following:

- A CCA DES key-encrypting key of type IMPORTER with the control vector bit IMPORT enabled or IKEYXLAT.
- A TR-31 DES key-encrypting key with key usage K0 or K1, algorithm T, and mode of use D.

When the source key block is wrapped by protection method VARDRV-D, the key is:

- A CCA AES key-encrypting key of type IMPORTER and key usage IMPTT31D with WR-DES, WR-AES, or WR-HMAC capability, matching the wrapped key/initialization vector, and one of the following if necessary:
  - WRDERIVE when importing a derivation key with key usage B0, B1, B3, E0, E1, E2, E3, E4, E5, F0, F1, F2, F3, F4.
  - WR-DATA when importing a data encrypting key with key usage C0, D0, D3, M6, or M7.
  - WR-KEK when importing a key encrypting key with key usage K0, K1, or K4.
  - WR-PIN when importing a pin encrypting key with key usage P0, V0, V1, or V2.
- A TR-31 AES key-encrypting key. Key usage K0 or K1, algorithm A, and mode of use D.

When the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

Notes:

- ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap/unwrap TR-31 version 'B'/'C' key blocks that have, or will have, 'E' exportability. This is because ECB-mode does not comply with ANSI X9.24 Part 1.
- If the *unwrap_kek_identifier* is compliant-tagged, the TR-31 key block is imported as a compliant-tagged key token.

wrap_kek_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *wrap_kek_identifier* parameter in bytes.

The value must be zero if the *wrap_kek_identifier* parameter is not used.

The value must be 64 if the *wrap_kek_identifier* is specified by label.

Otherwise, the value must be between the actual length of the token and 9992.

wrap_kek_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the output CCA key token. The key identifier is an operational key token or the label of an operational token in key storage.

When *wrap_kek_identifier_length* is 0, this parameter is ignored and the *unwrap_kek_identifier* is also to be used to wrap the output CCA token.

When importing a DES key, the wrap KEK must be one of the following:

- A CCA DES key-encrypting key of type IMPORTER with the control vector bit IMPORT enabled or IKEYXLAT.

- A TR-31 DES importer. Key usage K0, algorithm T, and mode of use D.

When importing an AES or HMAC key, the wrap KEK must be one of the following:

- A CCA AES key-encrypting key of type IMPORTER with WR-AES or WR-HMAC capability, matching the wrapped key.
- A TR-31 AES importer. Key usage K0, algorithm A, and mode of use D.

When the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

Notes:

- ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap/unwrap TR-31 version 'B'/'C' key blocks that have/will have 'E' exportability. This is because ECB-mode does not comply with ANSI X9.24 Part 1.
- When the *TR31_key_block* contains a DK proprietary key, the *wrap_kek_identifier* must be specified.
- If the *wrap_kek_identifier* is compliant-tagged, the TR-31 key block is imported as a compliant-tagged key token.

output_key_identifier_length

Direction	Type
Input/Output	Integer

This parameter specifies the length of the *output_key_identifier* parameter, in bytes. On input, it specifies the length of the buffer represented by the *output_key_identifier* parameter and must be at least 64 bytes long. On output, it contains the length of the token returned in the *output_key_identifier* parameter.

output_key_identifier

Direction	Type
Output	String

This parameter contains the key token that is to receive the imported key. The output token will be a CCA internal or external key token containing the key received in the TR-31 key block. When the key usage is "I0", the output is a clear initialization vector.

num_opt_blocks

Direction	Type
Output	Integer

This parameter contains the number of optional blocks that are present in the TR-31 key block.

cv_source

Direction	Type
Output	Integer

This parameter contains information about how the control vector in the output key token was created. It can be one of the following three values:

X'00000000'
No CV was present in an optional block and the output CV was created by the verb based on input parameters and on the attributes in the TR-31 key block header or no CV was present in an optional block and the key usage fields were created by the verb based on input parameters and on the attributes in the TR-31 key block header.

X'00000001'
A CV was obtained from an optional block in the TR-31 key block, and the key usage and mode of use were also specified in the TR-31 header. The callable service verified compatibility of the header values with the CV and then used that CV in the output key token.

X'00000002'
A CV was obtained from an optional block in the TR-31 key block, and the key usage and mode of use in the TR-31 header held the proprietary values indicating that key use and mode should be obtained from the included CV. The CV from the TR-31 token was used as the CV for the output key token.

Any value other than these are reserved for future use and are currently invalid.

protection_method

Direction	Type
Output	Integer

This parameter contains information about what method was used to protect the input TR-31 key block. It can have one of the following values:

X'00000000'
The TR-31 key block was protected using the variant method as identified by a Key Block Version ID value of "A" (0x41).

X'00000001'
The TR-31 key block was protected using the derived key method as identified by a Key Block Version ID value of "B" (0x42).

X'00000002'
The TR-31 key block was protected using the variant method as identified by a Key Block Version ID value of "C" (0x43). Functionally this method is the same as 'A', but to maintain consistency a different value will be returned here for 'C'.

X'00000003'
The TR-31 key block was protected using the AES Key Derivation Binding Method for CBC mode identified by a Key Block Version ID value of "D" (X'44').

Any value other than these are reserved for future use and are invalid.

Restrictions

Proprietary values for the TR-31 header fields are not supported by this callable service with the exception of the proprietary values used by IBM CCA when carrying a control vector in an optional block in the header.

Usage notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned. TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

If the TR-31 key block is marked as a key component, the resulting CCA key will have the Key Part bit (bit 44) in the control vector set to 1.

The exportability attributes of the imported CCA token are set based on attributes in the TR-31 key block as described in the following table.

Table 408. Export attributes of an imported CCA token	
TR-31 Translate attribute value	**CCA action on import**
Non-exportable ("N")	CCA imports the key to an internal CCA key token. CV bit 17 (export) is set to zero to indicate that the key is not exportable. CV bit 57 (TR-31 Translate) is set to one to indicate that the key is not exportable to TR-31.
Exportable under trusted key ("E")	If the TR-31 token is wrapped with a CCA KEK in the old ECB format, the request is rejected because that KEK is not a trusted key. If the CCA KEK is in a newer X9.24 compliant CCA key block, then the TR-31 key is imported to CCA in exactly the same way as described for keys that are exportable under any key.
Exportable under any key ("S")	CCA imports the key to an internal CCA key token. CV bit 17 (export) is set to one to indicate that the key is exportable. CV bit 57 (TR-31 Translate) is set to zero to indicate that the key is also exportable to TR-31.

If necessary, use the Prohibit Export, Prohibit Exported Extended, or Restrict Key Attribute callable service to alter the export attributes of the CCA token after import.

If the TR-31 key block contains an optional block with a CCA CV of '00007D0003000000000000000000000000' for a single length key or '00007D00034100000000000000000000007D00032100000000000000000000' for a double length key, the resulting CCA token will be a zero CV DATA token.

The TR-31 key block can contain a CCA control vector in an optional data field in the header. If the CV is present, the service will check that CV for compatibility with the TR-31 key attributes to ensure the CV is valid for the key and if there are no problems it will use that CV in the CCA key token that is output by the service. If a CV is received, the import operation is not subject to any ACP controlling the importation of specific key types. The CV may be present in the TR-31 key block in two different ways, depending on options used when creating that block.

- If the TR-31 Translate callable service was called with option INCL-CV, the control vector is included in the TR-31 key block and the TR-31 key usage and mode of use fields contain attributes from the set defined in the TR-31 standard. The TR-31 Import callable service checks that those TR-31 attributes are compatible with the CV included in the block. It also verifies that no rule array keywords conflict with the CV contained in the TR-31 block.
- If the TR-31 Translate callable service was called with option ATTR-CV, the control vector is included in the TR-31 key block and the TR-31 key usage and mode of use fields contain proprietary values (ASCII "10" and "1", respectively) to indicate that the usage and mode information is contained in the included control vector. In this case, the TR-31 Import service uses the included CV as the control vector for the CCA key token it produces. It also verifies that the CV does not conflict with rule array keywords passed

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Refer to the PDF version of this book for a list of the valid attribute translations for import of TR-31 key blocks to CCA keys along with the access control points which govern those translations.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified], an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the imported key.

Access control points

The access control points in the domain role that control the general function of this service are:

TR-31 key block version ID	Access control
"A" (X'41')	T31I - Permit version A TR-31 key blocks

TR-31 key block version ID	Access control
"B" (X'42')	T31I - Permit version B TR-31 key blocks
"C" (X'43')	T31I - Permit version C TR-31 key blocks
"D" (X'44')	T31I - Permit version D TR-31 key blocks

When the key wrapping method keyword specifies a wrapping method that is not the default method, the **TR31 Import - Permit override of default wrapping method** access control point must be enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

In addition to the above controls, the service requires these additional controls to be enabled in the domain role depending on the rule array keyword provided:

Table 409. TR-31 to CCA Import required access controls

Rule-array keyword	Access control name	Offset (hex)	Specific key usage, version ID, and mode values
"B0": TR-31 BDK Base Derivation Key			
N/A	T31I - Permit B0:X to AES DKYGENKY:DUKPT BDK	017E	See Table 398 on page 965.
"C0": TR-31 CVK card verification keys			
CVK-CSC	T31I - Permit C0:G/C/V to DES MAC/ MACVER:AMEX-CSC	015B	See Table 399 on page 966.
CVK-CVV	T31I - Permit C0:G/C/V to DES MAC/ MACVER:CVVKEY-A	015A	
"D0": TR-31 data encryption keys			
N/A	T31I - Permit D0:E/D/B to AES CIPHER:ENC/DEC/ENC+DEC	01E0	See Table 400 on page 967.
"K0", "K1" and "K4": TR-31 key encryption or wrapping, or key block protection keys			

Table 409. TR-31 to CCA Import required access controls (continued)

Rule-array keyword	Access control name	Offset (hex)	Specific key usage, version ID, and mode values
OKEYXLAT or EXPORTER	T31I - Permit K0:E to DES EXPORTER/ OKEYXLAT	015C	See Table 402 on page 972.
	T31I - Permit K0:B to DES EXPORTER/ OKEYXLAT	015E	
	T31I - Permit K1/K4:E to DES EXPORTER/ OKEYXLAT	0160	
	T31I - Permit K1/K4:B to DES EXPORTER/ OKEYXLAT	0162	
IKEYXLAT or IMPORTER	T31I - Permit K0:D to DES IMPORTER/ IKEYXLAT	015D	
	T31I - Permit K0:B to DES IMPORTER/ IKEYXLAT	015F	
	T31I - Permit K1/K4:D to DES IMPORTER/ IKEYXLAT	0161	
	T31I - Permit K1/K4:B to DES IMPORTER/ IKEYXLAT	0163	
N/A	T31I - Permit K0:E to AES EXPORTER	01E3	
	T31I - Permit K0:D to AES IMPORTER	01E4	
	T31I - Permit K1/K4:E to AES EXPORTER:EXPTT31D+VARDRV-D	01E5	
	T31I - Permit AES K1/K4:D to AES IMPORTER:IMPTT31D+VARDRV-D	01E6	
"M0", "M1", "M3", and "M6": TR-31 ISO MAC algorithm keys			
N/A	T31I - Permit M0/M1/M3:G/C/V to DES MAC/MACVER:ANY-MAC	0164	See Table 403 on page 974.
	T31I - Permit M6:G/C/V to AES MAC:CMAC+GENONLY/GEN/VER	01E1	
"M7": TR-31 HMAC keys			
N/A	T31I - Permit M7:G/V/C to HMAC MAC: GENERATE/VERIFY	017D	See Table 404 on page 976.
"P0", "V0", "V1": TR-31 PIN encryption or PIN verification keys			

Table 409. TR-31 to CCA Import required access controls (continued)

Rule-array keyword	Access control name	Offset (hex)	Specific key usage, version ID, and mode values
N/A	T31I - Permit P0:E to DES OPINENC	0165	See Table 405 on page 976.
	T31I - Permit P0:D to DES IPINENC	0166	
	T31I - Permit P0:E/D to AES PINPROT:ENC/DEC+CBC+ISO-4	01E2	
PINGEN	T31I - Permit V0:N/G/C to DES PINGEN:NO-SPEC NOOFFSET	0167	
	T31I - Permit V1:N/G/C to DES PINGEN:IBM-PIN/IBM-PINO NOOFFSET	0169	
	T31I - Permit V2:N/G/C to DES PINGEN:VISA-PVV	016B	
	T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C	
PINVER	T31I - Permit V0:N/V to DES PINVER:NO-SPEC NOOFFSET	0168	
	T31I - Permit V1:N/V to DES PINVER:IBM-PIN/IBM-PINO NOOFFSET	016A	
	T31I - Permit V2:N/V to DES PINVER:VISA-PVV	016C	
	T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	017C	
"E0", "E1", "E2", "E3", "E4", and "E5": TR-31 EMC/chip issuer master-key keys			

Table 409. TR-31 to CCA Import required access controls (continued)

Rule-array keyword	Access control name	Offset (hex)	Specific key usage, version ID, and mode values
DKYL0	T31I - Permit E0:N/X to DES DKYGENKY:DKYL0+DMAC	016D	See Table 407 on page 980.
	T31I - Permit E0:N/X to DES DKYGENKY:DKYL0+DMV	016E	
	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DMPIN	0171	
	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DDATA	0172	
	T31I - Permit E2:N/X to DES DKYGENKY:DKYL0+DMAC	0175	
	T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DMAC	0179	
	T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DDATA	017A	
	T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DEXP	017B	
	T31I - Permit E0:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	01E7	
	T31I - Permit E1:X to AES DKYGENKY:DKYL0/L1/L2+D-SECMSG+SMPIN	01E8	
	T31I - Permit E2:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	01E9	
	T31I - Permit E3:X to AES DKYGENKY:D-CIPHER+ENC+DEC+CBC	01EA	
	T31I - Permit E4:X to AES DKYGENKY:DKYL0/L1/L2+D-CIPHER+ENC+DEC	01EC	
	T31I - Permit E5:X to AES DKYGENKY:DKYL0/L1/L2/D-MAC+GEN+CMAC	01ED	

Table 409. TR-31 to CCA Import required access controls (continued)

Rule-array keyword	Access control name	Offset (hex)	Specific key usage, version ID, and mode values
DKYL1	T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMAC	016F	See Table 407 on page 980.
	T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMV	0170	
	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DMPIN	0173	
	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DDATA	0174	
	T31I - Permit E2:N/X to DES DKYGENKY:DKYL1+DMAC	0176	
	T31I - Permit E0:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	01E7	
	T31I - Permit E1:X to AES DKYGENKY:DKYL0/L1/L2+D-SECMSG+SMPIN	01E8	
	T31I - Permit E2:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	01E9	
	T31I - Permit E3:X to AES DKYGENKY:D-CIPHER+ENC+DEC+CBC	01EA	
	T31I - Permit E4:X to AES DKYGENKY:DKYL0/L1/L2+D-CIPHER+ENC+DEC	01EC	
	T31I - Permit E5:X to AES DKYGENKY:DKYL0/L1/L2/D-MAC+GEN+CMAC	01ED	

Table 409. TR-31 to CCA Import required access controls (continued)

Rule-array keyword	Access control name	Offset (hex)	Specific key usage, version ID, and mode values
DKYL2	T31I - Permit E0:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	01E7	See Table 407 on page 980.
	T31I - Permit E1:X to AES DKYGENKY:DKYL0/L1/L2+D-SECMSG+SMPIN	01E8	
	T31I - Permit E2:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	01E9	
	T31I - Permit E3:X to AES DKYGENKY:D-CIPHER+ENC+DEC+CBC	01EA	
	T31I - Permit E4:X to AES DKYGENKY:DKYL0/L1/L2+D-CIPHER+ENC+DEC	01EC	
	T31I - Permit E5:X to AES DKYGENKY:DKYL0/L1/L2/D-MAC+GEN+CMAC	01ED	
DMAC	T31I - Permit E0:N/X to DES DKYGENKY:DKYL0+DMAC	016D	See Table 407 on page 980.
	T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMAC	016F	
	T31I - Permit E2:N/X to DES DKYGENKY:DKYL0+DMAC	0175	
	T31I - Permit E2:N/X to DES DKYGENKY:DKYL1+DMAC	0176	
	T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DMAC	0179	
DMV	T31I - Permit E0:N/X to DES DKYGENKY:DKYL0+DMV	016E	See Table 407 on page 980.
	T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMV	0170	
DMPIN	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DMPIN	0171	See Table 407 on page 980.
	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DMPIN	0173	
DDATA	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DDATA	0172	See Table 407 on page 980.
	T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DDATA	0174	
	T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DDATA	017A	

Table 409. TR-31 to CCA Import required access controls (continued)

Rule-array keyword	Access control name	Offset (hex)	Specific key usage, version ID, and mode values
DEXP	T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DEXP	017B	See Table 407 on page 980.
N/A	T31I - Permit E3:N/E/D/B/G/X to DES ENCIPHER	0177	See Table 407 on page 980.
	T31I - Permit E4:N/B/X to DES DKYGENKY:DKYL0+DDATA	0178	
	T31I - Permit E3:E/B to AES CIPHER:ENCRYPT/ENC+DEC	01EB	

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 410. TR-31 Import required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords WKEY-AES and WKEY-DES and triple-length DES keys require the July 2019 or later licensed internal code (LIC). Keywords IPINENC and OPINENC require the July 2019 or later licensed internal code. The XPRTCPAC keyword is not supported. Compliant-tagged key tokens are not supported. TR-31 key block containing an HMAC key and keywords HMAC-ISO, HMAC-X9, and HMAC-UNK require the June 2020 or later licensed internal code. Import of TR-31 key block containing an AES BDK is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). Import of TR-31 key block Mode of use “N” with B, C, or D key block version IDs is not supported. Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.

Table 410. TR-31 Import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 IBM z14 ZR1	Crypto Express5 CCA Coprocessor	Keywords WKEY-AES and WKEY-DES and triple-length DES keys require the December 2018 or later licensed internal code (LIC). Keywords IPINENC and OPINENC require the July 2019 or later licensed internal code. The XPRTCPAC keyword is not supported. Compliant-tagged key tokens are not supported. TR-31 key block containing an HMAC key and keywords HMAC-ISO, HMAC-X9, and HMAC-UNK require the June 2020 or later licensed internal code. Import of TR-31 key block containing an AES BDK is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). Import of TR-31 key block Mode of use “N” with B, C, or D key block version IDs is not supported. Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.
	Crypto Express6 CCA Coprocessor	Keywords WKEY-AES and WKEY-DES and triple-length DES keys require the December 2018 or later licensed internal code (LIC). Keywords IPINENC and OPINENC require the July 2019 or later licensed internal code. The XPRTCPAC keyword requires a CEX6C with the P41458.002 or later MCL. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). TR-31 key block containing an HMAC key and keywords HMAC-ISO, HMAC-X9, and HMAC-UNK require the June 2020 or later licensed internal code. Import of TR-31 key block containing an AES BDK requires the October 2020 or later licensed internal code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). Import of TR-31 key block Mode of use “N” with B, C, or D key block version IDs is not supported. Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.

Table 410. TR-31 Import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 IBM z15 T02	Crypto Express5 CCA Coprocessor	The XPRTCPAC keyword is not supported. Compliant-tagged key tokens are not supported. TR-31 key block containing an HMAC key and keywords HMAC-ISO, HMAC-X9, and HMAC-UNK require the June 2020 or later licensed internal code. Import of TR-31 key block containing an AES BDK is not supported. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). Import of TR-31 key block Mode of use “N” with B, C, or D key block version IDs require CCA release (5.7, 6.7, 7.4, or 8.0) or later licensed internal code (LIC). Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.
	Crypto Express6 CCA Coprocessor	TR-31 key block containing an HMAC key and keywords HMAC-ISO, HMAC-X9, and HMAC-UNK require the June 2020 or later licensed internal code. Import of TR-31 key block containing an AES BDK requires the September 2020 or later Licensed Internal Code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). Import of TR-31 key block Mode of use “N” with B, C, or D key block version IDs require CCA release (5.7, 6.7, 7.4, or 8.0) or later licensed internal code (LIC). Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.
	Crypto Express7 CCA Coprocessor	TR-31 key block containing an HMAC key and keywords HMAC-ISO, HMAC-X9, and HMAC-UNK require the June 2020 or later licensed internal code. Import of TR-31 key block containing an AES BDK requires the September 2020 or later Licensed Internal Code (LIC). Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). Import of TR-31 key block Mode of use “N” with B, C, or D key block version IDs require CCA release (5.7, 6.7, 7.4, or 8.0) or later licensed internal code (LIC). Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.

Table 410. TR-31 Import required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Import of TR-31 key block Mode of use “N” with B, C, or D key block version IDs require CCA release (5.7, 6.7, 7.4, or 8.0) or later licensed internal code (LIC). Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.
	Crypto Express8 CCA Coprocessor	Support for operational X9.143 key block requires the CCA release 8.1 or later licensed internal code (LIC). Support for external X9.143 key blocks with a key context field of ASCII '1' requires the CCA release 8.1 or later licensed internal code (LIC).

TR-31 Optional Data Build (CSNBT31O and CSNET31O)

A TR-31 key block can hold optional fields which are securely bound to the key block using the integrated MAC. The optional blocks may either contain information defined in the TR-31 standard, or they may contain proprietary data.

Use the TR-31 Optional Data Build callable service to construct the optional block data structure for a TR-31 key block. It builds the structure by adding one optional block with each call, until your entire set of optional blocks have been added.

With each call, the application program provides a single optional block by specifying its ID, its length, and its data in parameters *opt_block_id*, *opt_block_length*, and *opt_block_data* respectively. Each subsequent call appends the current optional block to any preexisting blocks in the *opt_blocks* parameter. On the first call to the callable service, *opt_blocks* is typically empty.

The callable service name for AMODE(64) is CSNET31O.

Format

```
CALL CSNBT31O(
         return_code,
         reason_code,
         exit_data_length,
         exit_data,
         rule_array_count,
         rule_array,
         opt_blocks_bfr_length,
         opt_blocks_length,
         opt_blocks,
         num_opt_blocks,
         opt_block_id,
         opt_block_data_length,
         opt_block_data )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 0 since no keywords are currently defined for this callable service.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. There are no *rule_array* keywords currently defined for this callable service.

opt_blocks_bfr_length

Direction	Type
Input	Integer

This parameter specifies the length of the buffer passed with the opt_blocks parameter. This length is used to determine if it would overflow the buffer size when adding a new optional block to the current contents of the buffer.

opt_blocks_length

Direction	Type
Input/Output	Integer

This parameter specifies the actual length of the set of optional blocks currently contained in the *opt_blocks* buffer. On output, it is updated with the length after the callable service has added the new optional block.

opt_blocks

Direction	Type
Input/Output	String

This parameter specifies a buffer containing the set of optional blocks being built. In the first call, it will generally be empty. The callable service will append one optional block to the buffer with each call. Parameter *opt_blocks_bfr_length* specifies the total length of this buffer, and an error will be returned if this length would be exceeded by adding the optional block in parameter *opt_block_data* to the current contents. This parameter is encoded in ASCII on both input and output.

num_opt_blocks

Direction	Type
Output	Integer

This parameter contains the number of optional blocks contained in the structure returned in parameter *opt_blocks*. This is provided as an output parameter so that it can subsequently be used as an input to the TR-31 Translate callable service.

opt_block_id

Direction	Type
Input	String

This parameter specifies a two-byte value which is the identifier (ID) of the optional block passed in parameter *opt_block_data*.

opt_block_data_length

Direction	Type
Input	Integer

This parameter specifies the length of the data passed in parameter *opt_block_data*. Note that it is valid for this length to be zero; an optional block can have an ID and a length, but no data.

opt_block_data

Direction	Type
Input	String

This parameter specifies a buffer where the application passes the data for the optional block that is to be added to those already in the buffer in parameter opt_blocks. The length of this data is specified in parameter opt_block_data_length.

Restrictions

None.

Usage notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned (see Appendix F, “EBCDIC and ASCII default conversion tables,” on page 1657). TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

Note that the Padding Block, ID "PB" is not allowed to be added by the user. A Padding Block of the appropriate size, if needed, will be added when building the TR-31 key block in TR-31 Translate. If the TR-31 Translate callable service encounters a padding block in the optional block data, an error will occur.

Required hardware

No cryptographic hardware is required by this callable service.

TR-31 Optional Data Read (CSNBT31R and CSNET31R)

A TR-31 key block can hold optional fields which are securely bound to the key block using the integrated MAC. The optional blocks may either contain information defined in the TR-31 standard, or they may contain proprietary data. A separate range of optional block identifiers is reserved for use with proprietary blocks.

Note that some of the parameters are only used with keyword INFO and others are only used with keyword DATA.

The callable service name for AMODE(64) is CSNET31R.

Format

```
CALL CSNBT31R(
         return_code,
         reason_code,
         exit_data_length,
         exit_data,
         rule_array_count,
         rule_array,
         TR31_key_block_length,
         TR31_key_block,
         opt_block_id,
         num_opt_blocks,
         opt_block_ids,
         opt_block_lengths,
         opt_block_data_length,
         opt_block_data )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 1

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The *rule_array* keywords for this callable service are shown in the following table.

Table 411. Keywords for TR-31 Optional Data Read Rule Array Control Information

Keyword	Meaning
Operation – one required	
INFO	Return information about the optional blocks in the TR-31 key block.
DATA	Return the data contained in a specified optional block in the TR-31 key block.

TR31_key_block_length

Direction	Type
Input	Integer

This parameter specifies the length of the *TR31_key_block* parameter, in bytes. The parameter may specify a length that is greater than the size of the key block however it can never be greater than the size of the buffer where the key block resides. This value must be between 16 and 9992, inclusive. If a label is specified, the length must be exactly 64.

TR31_key_block

Direction	Type
Input	String

This parameter contains the TR-31 key block that is to be parsed. The length of the TR-31 block is specified using parameter *TR31_key_block_length*.

opt_block_id

Direction	Type
Input	String

This parameter is only used with option DATA. It is ignored for others. It specifies a 2-byte string which contains the identifier of the block from which the application is requesting data. The callable service will locate this optional block within the TR-31 structure and copy the data from that optional block into the returned *opt_block_data* buffer. If the specified optional block is not found in the TR-31 key block, an error will occur.

num_opt_blocks

Direction	Type
Input	Integer

This parameter specifies the number of optional blocks in the TR-31 key block. The value is compared to the corresponding value in the TR-31 block header and if they do not match the callable service fails with an error. This parameter is only used for option INFO and is not examined for any other options.

opt_block_ids

Direction	Type
Output	String Array

This parameter contains an array of two-byte string values. Each of these values is the identifier (ID) of one of the optional blocks contained in the TR-31 key block. The callable service returns a list containing the ID of each optional block that is in the TR-31 block, and the list is in the order that the optional blocks appear in the TR-31 header. The total length of the returned list will be two times the number of optional blocks, and the caller must supply a buffer with a length at least twice the value it passes in parameter *num_opt_blocks*. This parameter is only used for option INFO and is not examined for any other options.

opt_block_lengths

Direction	Type
Output	Array

This parameter contains an array of 16-bit integer values. Each of these values is the length in bytes of one of the optional blocks contained in the TR-31 key block. The callable service returns a list containing the length of each optional block that is in the TR-31 block, and the list is in the order that the optional blocks appear in the TR-31 header. The total length of the returned list will be two times the number of optional blocks and the application program must supply a buffer with a length at least two times the value it passes in parameter *num_opt_blocks*. This parameter is only used for option INFO and is not examined or altered for any other options.

opt_block_data_length

Direction	Type
Input/Output	Integer

This parameter specifies the length for parameter opt_block_data. On input it must be set to the length of the buffer provided by the application program, and on output it is updated to contain the length of the returned optional block data, in bytes. It is only used for option DATA.

opt_block_data

Direction	Type
Output	String

This parameter contains a buffer where the callable service stores the data it reads from the specified optional block. The buffer must have enough space for the data, as indicated by the input value of parameter opt_block_data_length. If not an error occurs and no changes are made to the contents of

the buffer. If the size of the buffer is sufficient, the data is copied to the buffer and its length is stored in parameter opt_block_data_length. It is only used for option DATA and is not examined or altered for any other options.

Restrictions

None

Usage notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned (see Appendix F, "EBCDIC and ASCII default conversion tables," on page 1657). TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

The TR-31 Optional Data Read callable service (CSNBT31R and CSNET31R) can be used in conjunction with the TR-31 Parse callable service (CSNBT31P and CSNET31P) to obtain both the standard header fields and any optional data blocks from the key block. This is generally a three-step process.

1. Use the TR-31 Parse callable service to determine how many optional blocks are in the TR-31 token. This is returned in the *num_opt_blocks* parameter.
2. Use keyword INFO with the TR-31 Optional Data Read callable service to obtain lists of the optional block identifiers and optional block lengths. Your buffers must be large enough to hold the returned data, but the required size can be determined from the number of blocks obtained in the preceding step.
3. Use keyword DATA with the TR-31 Optional Data Read callable service to obtain the data for a particular optional block, specified by the block identifier.

Required hardware

No cryptographic hardware is required by this callable service.

TR-31 Parse (CSNBT31P and CSNET31P)

Use the TR-31 Parse callable service to retrieve standard header information from a TR-31 key block without importing the key.

The callable service name for AMODE(64) is CSNET31P.

Format

```
CALL CSNBT31P(
         return_code,
         reason_code,
         exit_data_length,
         exit_data,
         rule_array_count,
         rule_array,
         TR31_key_block_length,
         TR31_key_block,
         key_block_version,
         key_block_length,
         key_usage,
         algorithm,
         mode,
         key_version_number,
         exportability,
         num_opt_blocks )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 0 because no keywords are currently defined for this callable service.

rule_array

Direction	Type
Input	String

A *rule_array* contains keywords that provide control information to the callable service. No rule array keywords are currently defined for this callable service.

TR31_key_block_length

Direction	Type
Input	Integer

This parameter specifies the length of the *TR31_key_block* parameter, in bytes. The parameter may specify a length that is greater than the size of the key block (however it can never be greater than the

size of the buffer where the key block resides). This value must be between 16 and 9992, inclusive. If a label is specified, the length must be exactly 64.

TR31_key_block

Direction	Type
Input	String

This parameter contains the TR-31 key block that is to be parsed.

key_block_version

Direction	Type
Output	String

This parameter contains a one-byte character value that indicates the version of the TR-31 key block, parsed from the block itself. CCA only supports versions "A", "B", and "C" key blocks.

key_block_length

Direction	Type
Output	Integer

This parameter contains the length of the key block as obtained from the TR-31 key block header. Note that this may be different from the input value in parameter *TR31_key_block_length*, if the application program specifies a length that is greater than the actual length of the key block.

key_usage

Direction	Type
Output	String

This parameter contains a 2-byte string value indicating the TR-31 key usage value for the key contained in the block. The value is obtained from the TR-31 key block header. The usage defines the type of function this key can be used with, such as data encryption, PIN encryption, or key wrapping.

algorithm

Direction	Type
Output	String

This parameter contains a one-byte string identifying the cryptographic algorithm the wrapped key is to be used with. The value is read from the TR-31 key block header. The service does not treat a proprietary algorithm value as an error.

mode

Direction	Type
Output	String

This parameter contains a one-byte string indicating the TR-31 mode of use for the key contained in the block. The value is obtained from the TR-31 key block header. The mode of use describes what operations the key can perform, within the limitations specified with the key usage value. For example, a key with usage for data encryption can have a mode to indicate it may be used for encryption only, decryption only, or both encryption and decryption.

key_version_number

Direction	Type
Output	String

This parameter contains a two-byte string obtained from the TR-31 key block header which represents versioning information about the key contained in the block.

exportability

Direction	Type
Output	String

This parameter contains a one-byte string indicating the key exportability value from the TR-31 key block header. This value indicates whether the key can be exported from this system, and if so it specifies conditions under which export is permitted.

num_opt_blocks

Direction	Type
Output	Integer

This parameter contains the number of optional blocks that are part of the TR-31 key block.

Restrictions

None

Usage notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned (see Appendix F, “EBCDIC and ASCII default conversion tables,” on page 1657). TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

The TR-31 Optional Data Read callable service (CSNBT31R and CSNET31R) can be used in conjunction with the TR-31 Parse callable service (CSNBT31P and CSNET31P) to obtain both the standard header fields and any optional data blocks from the key block. This is generally a three-step process.

1. Use the TR-31 Parse callable service to determine how many optional blocks are in the TR-31 token. This is returned in the num_opt_blocks parameter.
2. Use keyword INFO with the TR-31 Optional Data Read callable service to obtain lists of the optional block identifiers and optional block lengths. Your buffers must be large enough to hold the returned data, but the required size can be determined from the number of blocks obtained in the preceding step.
3. Use keyword DATA with the TR-31 Optional Data Read callable service to obtain the data for a particular optional block, specified by the block identifier.

Required hardware

No cryptographic hardware is required by this callable service.

TR-31 Translate (CSNBT31X and CSNET31X) (previously called TR-31 Export)

Use the TR-31 Translate callable service to:

- Translate an internal CCA token to an internal TR-31 key block.
- Export an internal CCA token to an external TR-31 key block.
- Translate and convert an external CCA token to an external TR-31 key block.
- Export an internal TR-31 key block to an external TR-31 key block.
- Import an external TR-31 key block to an internal TR-31 key block.
- Translate an external TR-31 key block from one wrapping key to another.
- Change the attributes of an internal TR-31 key block using keywords.
- Compliance check a TR-31 key block.
- Compliance tag an internal TR-31 key block.

In addition, you may change the attributes of a TR-31 key block using rule array keywords, to the extent allowed, during import, export, or translate. Because there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the caller may need to specify the attributes to attach to the exported output key through the rule array.

The callable service name for AMODE(64) is CSNET31X.

Format

```
CALL CSNBT31X(
                    return_code,
                    reason_code,
                    exit_data_length,
                    exit_data,
                    rule_array_count,
                    rule_array,
                    key_version_number,
                    key_field_length,
                    source_key_identifier_length,
                    source_key_identifier,
                    unwrap_kek_identifier_length,
                    unwrap_kek_identifier,
                    wrap_kek_identifier_length,
                    wrap_kek_identifier,
                    opt_blks_length,
                    opt_blocks,
                    tr31_key_block_length,
                    tr31_key_block )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Input/Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter.

The *rule_array_count* parameter must be between 1 and 13 inclusive.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The *rule_array* keywords for this callable service are shown in Table 412 on page 1009. See Table 428 on page 1050 for valid combinations of Usage and Mode

Table 412. Keywords for TR-31 Translate Rule Array Control Information

Keyword	Meaning
Source key algorithm - one optional	
SKEY-AES	Specifies that the *source_key_identifier* parameter identifies an AES key-token or the label of a key record in the CKDS. Only valid with keyword VARDRV-D.
SKEY-DES	Specifies that the *source_key_identifier* parameter identifies a DES key-token or the label of a key record in the CKDS. This is the default.
SKEYHMAC	Specifies that the *source_key_identifier* parameter identifies an HMAC key-token or the label of a key record in the CKDS. Only valid with keyword VARDRV-D.
TR-31 key block protection method or compliance rule - one required	
VARXOR-A	Use the variant method corresponding to a TR-31 Key Block Version ID of "A" (0x41). This is a DES method and uses a DES wrapping key to protect a DES key.
VARDRV-B	Use the key derivation method corresponding to a TR-31 Key Block Version ID of "B" (0x42). This is a DES method and uses a DES wrapping key to protect a DES key.
VARXOR-C	Use the variant method corresponding to a TR-31 Key Block Version ID of "C" (0x43). This is a DES method and uses a DES wrapping key to protect a DES key.

Table 412. Keywords for TR-31 Translate Rule Array Control Information (continued)	
Keyword	**Meaning**
VARDRV-D	Specifies to protect the key block using the AES Key Derivation Binding Method 2018 edition. Corresponds to TR-31 Key Block Version ID of "D" (0x44). Required with DK proprietary export keywords TYPATO11, TYPBTO10, or DMP0TO12. This is a AES method and uses an AES wrapping key to protect a DES, AES, or HMAC key.
COMP-CHK	Check if the key block can have the compliance tag. Requires source key identifier to be an internal TR-31 key block. No other keyword may be specified with this keyword.
COMP-TAG	Convert the input key block into a compliant-tagged key block. The source key identifier must be an internal TR-31 key block. No other keyword may be specified with this keyword.
Control vector transport control - optional. Not valid with protection method VARDRV-D. Otherwise, if no keyword from this group is supplied, the CV in the *source_key_identifier* is still verified to agree with the 'key usage' and 'mode of use' keywords specified from the preceding groups. Not valid with the INITVEC keyword.	
INCL-CV	Include the CCA Control Vector as an optional field in the TR-31 key block header. The TR-31 usage and mode of use fields will indicate the key attributes, and those attributes (derived from the keywords passed from the preceding groups) will be verified by the callable service to be compatible with the ones in the included control vector.
ATTR-CV	Include the CCA Control Vector as an optional field in the TR-31 key block header. The TR-31 usage will be set to the proprietary ASCII value "10" ('3130'x) to indicate usage information is specified in the included CV, and the mode of use will be set to the proprietary ASCII value "1" ('31'x) to indicate that mode is likewise specified in the CV. **Note:** No key usage or mode of use keywords are allowed when this keyword is specified.
Key Context for TR31 key block.	
STOREXCH	Either storage or key exchange context. This allows interoperability with legacy key blocks. **Note:** This does not imply that the wrapping key for a key block can be used for both storage and key exchange, merely that the storage or exchange of this key Bblock is determined by the properties of the wrapping key. Sets the byte at offset 14 of the header to ASCII "0". This is seen as "external" in CCA. This is the default.
INTERNAL	Key storage context only. The key block is internal and can be used as an operational key, but not a transport key. Sets the byte at offset 14 of the header to ASCII "1".
EXCHANGE	Key exchange context only. The key block is wrapped by a transport key for exchange between a communicating pair. Sets the byte at offset 14 of the header to ASCII "2". This is seen as "external" in CCA.

Table 413. Keywords for TR-31 Translate Rule Array Control Information

Keyword	TR-31 key usage	CCA key types	Meaning
***TR-31 key usage values for output key* (one required).** Not valid if ATTR-CV keyword is specified. Only these TR-31 usages are supported.			
BDK	"B0"	DES KEYGENKY or AES DKYGENKY	Specifies to export to a TR-31 base derivation key (BDK). You must select one mode of use keyword from Table 415 on page 1020 with this usage keyword. The table shows all of the supported translations for key usage keyword BDK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
DUKPT	"B1"	DES KEYGENKY or AES DKYGENKY	Specifies to export an Initial DUKPT key. You must select one mode of use keyword from Table 415 on page 1020 with this usage keyword. The table shows all of the supported translations for key usage keyword DUKPT. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
KDK	"B3"	DES DKYGENKY or AES KDKGENKY	Specifies to export a key derivation key. You must select one mode of use keyword from Table 415 on page 1020 with this usage keyword. The table shows all of the supported translations for key usage keyword KDK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
CVK	"C0"	DES MAC or DES DATA	Specifies to export to a TR-31 CVK card verification key. You must select one mode of use keyword from Table 416 on page 1022 with this usage keyword. The table shows all of the supported translations for key usage keyword CVK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
ENC	"D0"	DES ENCIPHER, DES DECIPHER, DES CIPHER, DES DATA, AES CIPHER	Specifies to export to a TR-31 data encryption key You must select one mode of use keyword from Table 417 on page 1024 with this usage keyword. The table shows all of the supported translations for key usage keyword ENC. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.

Table 413. Keywords for TR-31 Translate Rule Array Control Information (continued)

Keyword	TR-31 key usage	CCA key types	Meaning
ENCSENS	"D3"	DES CIPHERXI, DES CIPHERXO, DES CIPHERXL, AES CIPHER:C-XLATE	Specifies to export a data encryption key for sensitive data. You must select one mode of use keyword from Table 415 on page 1020 with this usage keyword. The table shows all of the supported translations for key usage keyword ENC. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
EMVACMK	"E0"	DES DKYGENKY or AES DKYGENKY	Specifies to export to a TR-31 EMV/chip issuer master key: application cryptograms key. You must select one mode of use keyword from Table 423 on page 1040 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVACMK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
EMVSCMK	"E1"	DES DKYGENKY or AES DKYGENKY	Specifies to export to a TR-31 EMV/chip issuer master key: secure messaging for confidentiality key. You must select one mode of use keyword from Table 423 on page 1040 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVSCMK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
EMVSIMK	"E2"	DES DKYGENKY or AES DKYGENKY	Specifies to export to a TR-31 EMV/chip issuer master key: secure messaging for integrity key. You must select one mode of use keyword from Table 423 on page 1040 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVSIMK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
EMVDAMK	"E3"	DES DATA, DES MAC, DES ENCIPHER, DES CIPHER or AES CIPHER, AES DKYGENKY	Specifies to export to a TR-31 EMV/chip issuer master key: data authentication code key. You must select one mode of use keyword from Table 423 on page 1040 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVDAMK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.

Table 413. Keywords for TR-31 Translate Rule Array Control Information (continued)

Keyword	TR-31 key usage	CCA key types	Meaning
EMVDNMK	"E4"	DES DKYGENKY or AES DKYGENKY	Specifies to export to a TR-31 EMV/chip issuer master key: dynamic numbers key. You must select one mode of use keyword from Table 423 on page 1040 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVDNMK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
EMVCPMK	"E5"	DES DKYGENKY or AES DKYGENKY	Specifies to export to a TR-31 EMV/chip issuer master key: card personalization key. You must select one mode of use keyword from Table 423 on page 1040 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVDPMK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
EMVAC-F	"F0"	DES DKYGENKY or AES DKYGENKY	Specifies to export an EMV/chip card key: application cryptograms.
EMVSC-F	"F1"	DES DKYGENKY or AES DKYGENKY	Specifies to export an EMV/chip card key: secure messaging for confidentiality.
EMVSI-F	"F2"	DES DKYGENKY or AES DKYGENKY	Specifies to export an EMV/chip card key: secure messaging for integrity.
EMVDA-F	"F3"	DES DATA, MAC, CIPHER, ENCIPHER-or-AES CIPHER, DKYGENKY	Specifies to export an EMV/chip card key: data authentication code.
EMVDN-F	"F4"	DES DKYGENKY or AES DKYGENKY	Specifies to export an EMV/chip card key: dynamic numbers.
ISOMAC0	"M0"	DES MAC, DES MACVER, DES DATA, DES DATAM, or DES DATAMV	Specifies to export to a TR-31 ISO 16609 MAC algorithm 1 (using TDEA) key. You must select one mode of use keyword from Table 419 on page 1027 with this usage keyword. The table shows all of the supported translations for key usage keyword ISOMAC0. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
ISOMAC1	"M1"	DES MAC, DES MACVER, DES DATA, DES DATAM, or DES DATAMV	Specifies to export to a TR-31 ISO 9797-1 MAC algorithm 1 key. You must select one mode of use keyword from Table 419 on page 1027 with this usage keyword. The table shows all of the supported translations for key usage keyword ISOMAC1. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.

Table 413. Keywords for TR-31 Translate Rule Array Control Information (continued)

Keyword	TR-31 key usage	CCA key types	Meaning
ISOMAC3	"M3"	DES MAC, MACVER, DATA, DATAM, or DATAMV	Specifies to export to a TR-31 ISO 9797-1 MAC algorithm 3 key. You must select one mode of use keyword from Table 419 on page 1027 with this usage keyword. The table shows all of the supported translations for key usage keyword ISOMAC3. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
ISOMAC6	"M6"	DES MAC, DATA, DATAM, MACVER, DATAMV, AES MAC	Specifies to export a DES MAC-capable or AES CMAC key to a TR-31 ISO 9797-1:2011 MAC algorithm 5/ CMAC key block. You must select one mode of use keyword from Table 419 on page 1027 with this usage keyword. The table shows all of the supported translations for key usage keyword ISOMAC6. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
HMAC	"M7"	MAC	Specifies to export an HMAC algorithm key. You must select one mode of use keyword from Table 421 on page 1034 with this usage keyword. The table shows all of the supported translations for key usage keyword HMAC. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
INITVEC	"I0"	N/A	Specifies to export an initialization vector passed to the service to a TR-31 Initialization Vector key block.
KEK	"K0"	DES EXPORTER, DES OKEYXLAT, AES EXPORTER, or SECMSG:SMKEY	Specifies to export to a TR-31 key-encryption or wrapping key. You must select one mode of use keyword from Table 418 on page 1025 with this usage keyword. The table shows all of the supported translations for key usage keyword KEK. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
KEK-WRAP	"K1"	DES IMPORTER, DES IKEYXLAT, or AES IMPORTER	Specifies to export to a TR-31 key block protection key. You must select one mode of use keyword from Table 418 on page 1025 with this usage keyword. The table shows all of the supported translations for key usage keyword KEK-WRAP. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.

Table 413. Keywords for TR-31 Translate Rule Array Control Information (continued)

Keyword	TR-31 key usage	CCA key types	Meaning
KEK-WRK4	"K4"	DES IMPORTER, AES IMPORTER, DES EXPORTER, AES EXPORTER, DES IKEYXLAT, or DES OKEYXLAT	Specifies to export an AES key encrypting key to an ISO 20038 key block protection key. You must select one mode of use keyword from Table 418 on page 1025 with this usage keyword. The table shows all of the supported translations for key usage keyword KEK-WRK4. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
PINENC	"P0"	DES OPINENC, DES IPINENC, AES PINPROT, or SECMSG:SMPIN	Specifies to export to a TR-31 PIN encryption key. You must select one mode of use keyword from Table 422 on page 1035 with this usage keyword. The table shows all of the supported translations for key usage keyword PINENC. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
PINVO	"V0"	DES PINGEN or DES PINVER	Specifies to export to a TR-31 PIN verification key or other algorithm. You must select one mode of use keyword from Table 422 on page 1035 with this usage keyword. The table shows all of the supported translations for key usage keyword PINVO. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
PINV3624	"V1"	DES PINGEN or DES PINVER	Specifies to export to a TR-31 PIN verification, IBM 3624 key. You must select one mode of use keyword from Table 422 on page 1035 with this usage keyword. The table shows all of the supported translations for key usage keyword PINV3624. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.
VISAPVV	"V2"	DES PINGEN or DES PINVER	Specifies to export to a TR-31 PIN verification, VISA PVV key. You must select one mode of use keyword from Table 422 on page 1035 with this usage keyword. The table shows all of the supported translations for key usage keyword VISAPVV. It also shows the access controls that must be enabled in the domain role in order to use the combination of inputs shown.

Table 413. Keywords for TR-31 Translate Rule Array Control Information (continued)

Keyword	TR-31 key usage	CCA key types	Meaning
TYPATO11	"11"	AES KDKGENKY	Specifies to export an AES KDKGENKY key that has usage Entity Type A (KDKTYPEA) as its key diversification key entity type into a TR-31 proprietary external token format, as defined by the German Banking Industry Committee, Die Deutsch Kreditwirtschaft, also known as DK. The keyword requires a mode-of-use keyword from Table 424 on page 1042. The table shows the access controls that must be enabled in the domain role to use the combination of inputs shown.
TYPBTO10	"10"	AES KDKGENKY	Specifies to export an AES KDKGENKY key that has usage Entity Type B (KDKTYPEB) as its key diversification key entity type into a TR-31 proprietary external token format, as defined by DK. The keyword requires a mode-of-use keyword from Table 424 on page 1042. The table shows the access controls that must be enabled in the domain role to use the combination of inputs shown.
DMP0TO12	"12"	DES DKYGENKY	Specifies to export a DES DKYGENKY key that has a usage DKYL0 (CV bits 12 - 14 = B'000') and DMPIN (CV bits 19 - 22 = B'1001') into a TR-31 proprietary external token format, as defined by DK. The keyword requires a mode-of-use keyword from Table 424 on page 1042. The table shows the access controls that must be enabled in the domain role to use the combination of inputs shown.
***TR-31 mode of key use* (one required).** Not valid if ATTR-CV keyword is specified. Only those TR-31 modes shown are supported.			
Keyword	**TR-31 mode**	**Usage keyword**	**Meaning**
ENCDEC	"B"	ENC, KEK, KEK-WRAP, KEK-WRK4, PINENC	Specifies both encrypt and decrypt, and wrap and unwrap.
DEC-ONLY	"D"	ENC, KEK, KEK-WRAP, PINENC	Specifies to decrypt and unwrap only.
ENC-ONLY	"E"	ENC, PINENC	Specifies to encrypt and wrap only.
GENVER	"C"	CVK, HMAC, ISOMAC0, ISOMAC1, ISOMAC3, PINVO, PINV3624, VISAPVV	Specifies to both generate and verify.
GEN-ONLY	"G"	CVK, HMAC, ISOMAC0, ISOMAC1, ISOMAC3, PINVO, PINV3624, VISAPVV	Specifies to generate only.

Table 413. Keywords for TR-31 Translate Rule Array Control Information (continued)

Keyword	TR-31 key usage	CCA key types	Meaning
VER-ONLY	"V"	CVK, HMAC, ISOMAC0, ISOMAC1, ISOMAC3, PINVO, PINV3624, VISAPVV	Specifies to verify only.
DERIVE	"X"	BDK, EMVACMK, EMVSCMK, EMVSIMK, EMVDAMK, EMVDNMK, EMVCPMK	Specifies that key is used to derive other keys.
ANY	"N"	BDK, PINVO, PINV3624, VISAPVV, EMVACMK, EMVSCMK, EMVSIMK, EMVDAMK, EMVDNMK, EMVCPMK	Specifies no special restrictions (other than restrictions implied by the key usage).

***TR-31 exportability* (one, optional).** Use to set exportability field in TR-31 key block. Defines whether the key may be transferred outside the cryptographic domain in which the key is found.

Keyword	TR-31 byte	Meaning
EXP-TRST	"E"	Specifies that the key in the TR-31 key block is exportable under a key-encrypting key in a form that meets the requirements of X9.24 Parts 1 or 2. **Note:** A TR-31 key block with a key block version ID of "B" or "C" and an exportability field value of "E" cannot be wrapped by a key-encrypting key that is wrapped in ECB mode (legacy wrap mode). This is because ECB mode does not comply with ANS X9.24 Part 1.
EXP-NONE	"N"	Specifies that the key in the TR-31 key block is non-exportable.
EXP-ANY	"S"	Specifies that the key in the TR-31 key block is sensitive, exportable under a key-encrypting key in a form not necessarily meeting the requirements of X9.24 parts 1 or 2. This is the default.

***Initialization vector algorithm* (one required with INITVEC).**

Keyword	TR-31 byte	Meaning
IV-DES	"D"	Specifies that algorithm DES is placed in the "I0" key block algorithm field.
IV-TDES	"T"	Specifies that algorithm TDES is placed in the "I0" key block algorithm field.
IV-AES	"A"	Specifies that algorithm AES is placed in the "I0" key block algorithm field.

***HMAC hash algorithm limit* (one, required).** Valid only with HMAC keys "M7".

The keyword specified determines whether the format of the TR-31 key block is based on ASC X9 TR 31-2018 or ISO 20038. ISO 20038 and ANSI X9 TR-31-2018 represent the HMAC hash algorithm limit in different ways:

- ISO 20038 represents hash limit in the algorithm value at offset 7. An HMAC key limited to SHA-1 uses ASCII 'H', for the SHA-2 limit the value is ASCII 'I', and for the SHA-3 limit the value is ASCII 'J'.
- ANSI X9 TR-31-2018 always uses 'H' for the algorithm value at offset 7 and represents the hash algorithm limit in the optional block with identifier 'HM'.

Only valid with SKEYHMAC. Note that the input HMAC key token must allow the hash algorithm selected below in key-usage field 2, high-order byte.

Keyword	Meaning

Table 413. Keywords for TR-31 Translate Rule Array Control Information (continued)

Keyword	TR-31 key usage	CCA key types	Meaning
ISOSHA-1		Specifies to use the SHA-1 hash algorithm with the HMAC key as defined by ISO 20038. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII "H" and does not include an optional block with a block ID of "HM". The *source_key_identifier* parameter must identify an HMAC key in a version X'05' variable-length symmetric key-token that allows a hash method of SHA-1 (KUF2 HOB = B'1xxx xxxx'). **Security note:** Keyword ISOSHA-1 creates a TR-31 key block with an algorithm of "H" and no optional block with a block ID of "HM". Under ISO 20038, this key block allows only SHA-1 as the hash algorithm to use with the HMAC key. However, ASC X9 TR 31-2018 also allows a key block with an algorithm of "H" and no optional block with a block ID of "HM", which is interpreted as an HMAC key with no hash algorithm limits. There is no limit to SHA-1. For this reason, it is recommended to use the ISOSHA-1 keyword only when sending a key to a partner that is known to require and understand the ISO 20038 version of the hash limit, or to have a clear understanding that the partner will receive an HMAC key with no hash algorithm limits under TR-31-2018. When possible, the SHA-1 keyword should be used instead, if the partner can receive a key block with the HM optional block that limits hash algorithm.	
ISOSHA-2		Specifies to use the SHA-2 hash algorithm with the HMAC key as defined by ISO 20038. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII "I" and does not include an optional block with a block ID of "HM". The *source_key_identifier* parameter must identify an HMAC key in a Version X'05' variable-length symmetric key-token that allows a hash method of SHA-2 (KUF2 HOB = B'x1xx xxxx' for SHA-224, KUF2 HOB = B'xx1x xxxx' for SHA-256, KUF2 HOB = B'xxx1 xxxx' for SHA-384, or KUF2 HOB = B'xxxx 1xxx' for SHA-512).	
SHA-1		Specifies to use the SHA-1 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII "H" and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "10". The *source_key_identifier* parameter must identify an HMAC key in a Version X'05' variable-length symmetric key-token that allows a hash method of SHA-1 (KUF2 HOB = B'1xxx xxxx').	
SHA-224		Specifies to use the SHA-224 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII "H" and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "20". The *source_key_identifier* parameter must identify an HMAC key in a Version X'05' variable-length symmetric key-token that allows a hash method of SHA-224 (KUF2 HOB = B'x1xx xxxx').	
SHA-256		Specifies to use the SHA-256 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII "H" and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "21". The *source_key_identifier* parameter must identify an HMAC key in a Version X'05' variable-length symmetric key-token that allows a hash method of SHA-256 (KUF2 HOB = B'xx1x xxxx').	

Table 413. Keywords for TR-31 Translate Rule Array Control Information (continued)

Keyword	TR-31 key usage	CCA key types	Meaning
SHA-384		Specifies to use the SHA-384 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII "H" and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "22". The *source_key_identifier* parameter must identify an HMAC key in a Version X'05' variable-length symmetric key-token that allows a hash method of SHA-384 (KUF2 HOB = B'xxx1 xxxx').	
SHA-512		Specifies to use the SHA-512 hash algorithm with the HMAC key as defined by ASC X9 TR 31-2018. Sets the algorithm (offset 7 of the TR-31 key block) to ASCII "H" and includes an optional block with a block ID of "HM". Sets the hash algorithm used with the HMAC key (offset 4 of the optional block) to ASCII "23". The *source_key_identifier* parameter must identify an HMAC key in a Version X'05' variable-length symmetric key-token that allows a hash method of SHA-512 (KUF2 HOB = B'xxxx 1xxx').	
CPACF Control (One, optional). If creating an external key block, the OB-IBM keyword must also be specified.			
XPRTCPAC		Allow export to CPACF protected key format.	
NOEXCPAC		Prohibit export to CPACF protected key format. This is the default.	
DK PIN Enable (One, optional). If creating an external key block, the OB-IBM keyword must also be specified.			
DKPINOP		Allow the key to be used for DK operations.	
Optional Blocks to be added to an internal/external output key block (Optional, one or more keywords may be specified from this group). The following keywords must be specified to include the associated OB in the output key block.			
OB-DA		Specifies to add a DA optional block to *tr31_key_block*. This will not build a new DA OB. It can only take an existing DA block from the *source_key_identifier*. If the *source_key_identifier* does not contain a DA OB, this keyword is ignored. If you would like to add a new DA block, instead of using this keyword, the DA block must be built using the TR-31 Optional Data Build (CSNBT31O/CSNET31O) service and passed to this service via the *opt_blocks* parameter. Only valid with usage B3.	
OB-LB		Specifies to add an LB optional block to *tr31_key_block*. If the input is a version 05 CCA key token, this keyword will take the stored key name and use it to build an LB OB. If the input is a TR-31 key block, the LB OB will be used. This will not build a new LB OB. It can only take an existing key name or LB block from the *source_key_identifier*. If the *source_key_identifier* does not contain a key name or LB OB, this keyword is ignored. If you would like to add a new LB block, instead of using this keyword, the LB block must be built using the TR-31 Optional Data Build (CSNBT31O/CSNET31O) service and passed to this service via the *opt_blocks* parameter.	
Optional Blocks to be added to an external output key block. (Optional, one or more keywords may be specified from this group). If the input key is a TR-31 key block, OB-xx must be specified for each optional block that should be taken from the input key block and added to the output key block. If the input key block does not contain the specified OB, it will be created if allowed. If the OB cannot be added, the keyword will be ignored. **Note:** For internal output key blocks, these optional blocks will all be added, if possible, without needing to specify these keywords.			
OB-IBM		Specifies to add a proprietary IBM optional block to *tr31_key_block.*	

Table 413. Keywords for TR-31 Translate Rule Array Control Information (continued)

Keyword	TR-31 key usage	CCA key types	Meaning
OB-KC		Specifies to add a KC optional block to *tr31_key_block*. This keyword is not valid with single length DES keys or HMAC keys.	
OB-KP		Specifies to add a KP optional block to *tr31_key_block*. If the key is being wrapped under a new KEK as part of this service, a new KP optional block will be created and added to the key block.	
OB-TC		Specifies to add a TC optional block to *tr31_key_block*. This will not build a new TC OB. It can only take an existing TC block from the *source_key_identifier*. If the *source_key_identifier* does not contain a TC OB, this keyword is ignored.	
OB-TS		Specifies to add a TS optional block to *tr31_key_block*.	
OB-WP		Specifies to add a WP optional block to *tr31_key_block*. This will only build a new WP OB if the *source_key_identifier* is a version 05 CCA token and it will check the KUF bits to see if it was wrapped by a weaker key. Otherwise, it can only take an existing WP block from the *source_key_identifier*. If the *source_key_identifier* does not contain a WP OB, this keyword is ignored.	

Table 414. Export translation table for a initialization vector

Key usage keyword	Key block protection method keyword	Initialization vector algorithm	TR-31 exportability	Mode of use keyword
INITVEC ("I0")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	IV-DES, IV-TDES, IV-AES	EXP-ANY, EXP-TRST, EXP-NONE	ANY ("N")

Table 415. Export translation table for a TR-31 BDK base derivation key

Key usage keyword	Key block protection method keyword	CCA key type and required key usage attributes	Mode of use keyword	Access control name	Offset (hex)
BDK ("B0")	VARXOR-A	DES KEYGENKY, double length, UKPT (CV bit 18 = B'1')	ANY ("N")	T31X - Permit DES KEYGENKY: DUKPT to B0:N/X	0180
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
BDK ("B0")	VARDRV-D	AES DKYGENKY, A-DUKPT KUF set to 1	DERIVE ("X")	T31X - Permit AES DKYGENKY: DUKPT BDK to B0:X	01CF
DUKPT (“B1”)	VARDRV-B, VARXOR-C, VARDRV-D	DES KEYGENKY double length, UKPT (CV bit 18 = B'1')	DERIVE ("X")	T31X - Permit DES KEYGENKY:DUKPT, AES DKYGENKY:DUKPT to B1	03DF
	VARDRV-D	AES DKYGENKY			

Table 415. Export translation table for a TR-31 BDK base derivation key (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required key usage attributes	Mode of use keyword	Access control name	Offset (hex)
KDK (“B3”)	VARDRV-B, VARXOR-C, VARDRV-D	DES DKYGENKY	DERIVE ("X")	T31X - Permit DES DKYGENKY, AES KDKGENKY to B3	03E0
	VARDRV-D	AES DKYGENKY			

Notes:

1. These are the base keys from which derived unique key per transaction (DUKPT) initial keys are derived for individual devices such as PIN pads.
2. The following defines the only supported translations for this TR-31 usage. Usage must be the following:

   "B0"
   BDK base derivation key.

   "B1"
   Initial DUKPT derivation key.

   "B3"
   Key derivation key.
3. KEYGENKY keys are double length only.

Table 416. Export translation table for a TR-31 CVK card verification key (CVK)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes		Mode of use keyword	Access control name	Offset (hex)
CVK ("C0")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	MAC, single or double length, AMEX-CSC (CV bits 0 - 3 = B'0100')	MAC generate on, MAC verify off (CV bits 20 - 21 = B'10')	GEN-ONLY ("G")	T31X - Permit DES MAC/ MACVER:AMEX-CSC to C0:G/C/V	0181
			MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')	VER-ONLY ("V")		
			MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')	GENVER ("C")		
		MAC, double length, CVVKEY-A (CV bits 0 - 3 = B'0010')	MAC generate on, MAC verify off (CV bits 20 - 21 = B'10')	GEN-ONLY ("G")	T31X - Permit DES MAC/MACVER: CVV-KEYA to C0:G/C/V	0182
			MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')	VER-ONLY ("V")		
			MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')	GENVER ("C")		
		MAC, double or triple length, ANY-MAC (CV bits 0 - 3 = B'0000')	MAC generate on, MAC verify off (CV bits 20 - 21 = B'10')	GEN-ONLY ("G")	T31X - Permit DES MAC/MACVER: ANY-MAC to C0:G/C/V	0183
			MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')	VER-ONLY ("V")		
			MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')	GENVER ("C")		
		DATA, double or triple length	MAC generate on, MAC verify off (CV bits 20 - 21 = B'10')	GEN-ONLY ("G")	T31X - Permit DES DATA/DATAM/ DATAMV to C0:G/C/V	0184
			MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')	GENVER ("C")		

Table 416. Export translation table for a TR-31 CVK card verification key (CVK) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)

Security considerations:

1. There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "C0" key, provided that one or both applicable MAC generate and MAC verify control vector bits are on. However, a TR-31 "C0" key cannot be imported to the lower-security CCA DATA key, it can only be imported to a CCA key type of MAC or MACVER. This restriction eliminates the ability to export a CCA MAC or MACVER key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to Encipher, Decipher, or both.
2. Since the translation from TR-31 usage "C0" is controlled by rule array keywords when using the TR31 Key Import service, it is possible to convert an exported CCA CVVKEY-A or CVVKEY-B key into an AMEX-CSC key or the other way around. This can be restricted by not enabling access controls X'015A' (T31I - Permit C0:G/C/V to DES MAC/MACVER:CVVKEY-A) and X'015B' (T31I – Permit C0:G/C/V to DES MAC/MACVER:AMEX-CSC) at the same time. However, if both CVVKEY-x and AMEX-CSC translation types are required, then offsets X'015A' and X'015B' must be enabled. In this case, control is up to the development, deployment, and execution of the applications themselves.

Notes:

1. Card verification keys are used for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2, and CVV2 algorithms. In CCA, this corresponds to keys used with two algorithms:
   - Visa CVV and MasterCard CVC codes are generated and verified using the CVV Generate and CVV Verify services. These services require a key type of DATA or MAC/MACVER with a subtype extension (CV bits 0 - 3) of ANY-MAC, single-length CVVKEY-A and single-length CVVKEY-B, and a double-length CVVKEY-A (see CVV Key Combine service). The MAC generate and the MAC verify (CV bits 20 - 21) key usage values must be set appropriately.
   - American Express CSC codes are generated and verified using the Transaction Validation service. This service requires a key type of MAC or MACVER with a subtype extension of ANY-MAC or AMEX-CSC.
2. The following defines the only supported translations for this TR-31 usage. Usage must be the following:

   "C0"
   CVK card verification key.
3. CCA and TR-31 represent CVV keys differently. This makes representations between the two incompatible. CCA represents the key-A and key-B keys as two 8-byte (single length) keys, while TR-31 represents these keys as one 16-byte (double length) key. Current Visa standards now require one 16-byte key. The CVV Generate and CVV Verify services accept one 16-byte CVV key, using left and right key parts as key-A and key-B. See CVV Key Combine service This service provides a way to combine two single-length MAC-capable keys into one double-length CVV key.
4. Import and export of 8-byte CVVKEY-A and CVVKEY-B MAC/MACVER keys will only be allowed using the IBM proprietary TR-31 usage and mode values ("10" and "1", respectively) to indicate encapsulation of the IBM control vector in an optional block, since the 8-byte CVVKEY-A is meaningless and useless as a TR-31 "C0" usage key of any mode.

Table 417. Export translation table for a TR-31 data encryption key (ENC)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
ENC ("D0")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	DES ENCIPHER, single, double, or triple length	ENC-ONLY ("E")	T31X - Permit DES ENCIPHER/ DECIPHER/CIPHER to D0:E/D/B	0185
		DES DECIPHER, single, double, or triple length	DEC-ONLY ("D")		
		DES CIPHER, single, double, or triple length	ENCDEC ("B")		
		DES DATA, single, double, or triple length, Encipher on, Decipher on (CV bits 18 - 19 = B'11')	ENCDEC ("B")	T31X - Permit DES DATA to D0:E/D/B	0186
ENCSENS (“D3”)	VARDRV-B, VARDRV-D	DES CIPHERXO	ENC-ONLY ("E")	T31X - Permit CIPHER:XLATE to D3	03E1
		DES CIPHERXI	DEC-ONLY ("D")		
		DES CIPHERXL	ENCDEC ("B")		
	VARDRV-D	AES CIPHER, C-XLATE, ENCRYPT key usage attributes enabled	ENC-ONLY ("E")		
		AES CIPHER, C-XLATE, DECRYPT key usage attributes enabled	DEC-ONLY ("D")		
		AES CIPHER, C-XLATE, ENCRYPT, DECRYPT key usage attributes enabled	ENCDEC ("B")		

Security considerations:

- There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "D0" key, provided that one or both applicable Encipher or Decipher control vector bits are on. However, a TR-31 "D0" key cannot be imported to the lower-security CCA DATA key, it **can only be imported** to a CCA key type of ENCIPHER, DECIPHER, or CIPHER. This restriction eliminates the ability to export a CCA DATA key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to MAC generate and MAC verify.

Notes:

1. Data encryption keys are used for the encryption and decryption of data.
2. The following defines the only supported translations for this TR-31 usage. Usage must be the following:

   "D0"
   Data encryption.

   "D3"
   Cipher text translation.

Table 418. Export translation table for a TR-31 key encryption or wrapping, or key block protection key (KEK or KEK-WRAP)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
KEK ("K0")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	EXPORTER, double or triple length, EXPORT on (CV bit 21 = B'1')	ENC-ONLY ("E")	T31X - Permit DES EXPORTER/OKEYXLAT to K0:E	0187
		OKEYXLAT, double length			
		DES SECMSG, SMKEY bit on (CV bit 18 = B'1')		T31X – Permit SECMSG:SMKEY to K0	03E3
		IMPORTER, double or triple length, IMPORT on (CV bit 21 = B'1')	DEC-ONLY ("D")	T31X - Permit DES IMPORTER/IKEYXLAT to K0:D	0188
		IKEYXLAT, double length			
		EXPORTER, double or triple length, EXPORT on (CV bit 21 = B'1')	ENCDEC("B")	T31X - PERMIT EXPORTER to K0:B	02AD
		IMPORTER, double or triple length, IMPORT on (CV bit 21 = B'1')		T31X - PERMIT IMPORTER to K0:B	02AE
KEK-WRAP ("K1")	VARDRV-B, VARXOR-C, VARDRV-D	EXPORTER, double or triple length, EXPORT on (CV bit 21 = B'1')	ENC-ONLY ("E")	T31X - Permit DES EXPORTER/OKEYXLAT to K1/K4:E	0189
		OKEYXLAT, double length			
		IMPORTER, double or triple length, IMPORT on (CV bit 21 = B'1')	DEC-ONLY ("D")	T31X - Permit DES IMPORTER/IKEYXLAT to K1/K4:D	018A
		IKEYXLAT, double length			

Table 418. Export translation table for a TR-31 key encryption or wrapping, or key block protection key (KEK or KEK-WRAP) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)

Security considerations: The CCA OKEYXLAT, EXPORTER, IKEYXLAT, or IMPORTER KEK translation to a TR-31 "K0" key with mode "B" (both wrap and unwrap) is not allowed for security reasons. Even with access-control point control, this capability would give an immediate path to turn a CCA EXPORTER key into a CCA IMPORTER key, and the other way around.

Notes:

1. Key encryption or wrapping keys are used only to encrypt or decrypt other keys, or as a key used to derive keys that are used for that purpose.
2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:

   "K0"
   Key encryption or wrapping.

   "K1"
   TR-31 key block protection key.
3. CCA mode support is the same for version IDs "B" and "C". That is because the distinction between TR-31 "K0" and "K1" does not exist in CCA keys. CCA does not distinguish between targeted protocols, and so there is no good way to represent the difference. Also note that most wrapping mechanisms now involve derivation or key variation steps.
4. There is asymmetry in the SECMSG:SMKEY -> K0 translation. There is no way to translate a TR-31 K0 key to a CCA SECMSG:SMKEY.

Table 419. Export translation table for a TR-31 ISO MAC algorithm key (ISOMACn)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
ISOMAC0 ("M0")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	MAC, double or triple length, MAC generate on (CV bit 20 = B'1')	GEN-ONLY ("G")	T31X - Permit DES MAC/DATA/DATAM to M0:G/C	018B
		DATA, double or triple length, MAC generate on (CV bit 20 = B'1')			
		MAC, double or triple length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')	GENVER ("C")		
		DATAM, double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')			
		DATA, double or triple length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')			
		MACVER, double or triple length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')	VER-ONLY ("V")	T31X - Permit DES MACVER/DATA/ DATAMV to M0:V	018C
		DATAMV, double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')			

Table 419. Export translation table for a TR-31 ISO MAC algorithm key (ISOMACn) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
ISOMAC1 ("M1")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	MAC, single, double, or triple length, MAC generate on (CV bit 20 = B'1')	GEN-ONLY ("G")	T31X - Permit DES MAC/DATA/DATAM to M1:G/C	018D
		DATA, single, double, or triple length, MAC generate on (CV bit 20 = B'1')			
		MAC, single, double, or triple length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')	GENVER ("C")		
		DATAM, double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')			
		DATA, single, double, or triple length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')			
		MACVER, single, double, or triple length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')	VER-ONLY ("V")	T31X - Permit DES MACVER/DATA/ DATAMV to M1:V	018E
		DATAMV, double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')			

Table 419. Export translation table for a TR-31 ISO MAC algorithm key (ISOMACn) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
ISOMAC3 ("M3")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	MAC, single, double, or triple length, MAC generate on (CV bit 20 = B'1')	GEN-ONLY ("G")	T31X - Permit DES MAC/DATA/DATAM to M3:G/C	018F
		DATA, single, double, or triple length, MAC generate on (CV bit 20 = B'1')			
		MAC, single, double, or triple length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')	GENVER ("C")		
		DATAM, double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')			
		DATA, single, double, or triple length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')			
		MACVER, single, double, or triple length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')	VER-ONLY ("V")	T31X - Permit DES MACVER/DATA/ DATAMV to M3:V	0190
		DATAMV, double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')			

Table 419. Export translation table for a TR-31 ISO MAC algorithm key (ISOMACn) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
ISOMAC6 ("M6")	VARDRV-B,VARXOR-C,VARDRV-D	MAC, double, or triple length, MAC generate on (CV bit 20= B'1')	GEN-ONLY ("G")	T31X - Permit DES MAC to M6	03E7
		DATA, double, or triple length, MAC generate on (CV bit 20= B'1')			
		MAC, double, or triple length, MAC generate on, MAC verify on (CV bits 20 -21 = B'11')	GENVER ("C")		
		DATAM, double length, MAC generate on, MAC verify on (CV bits 20 -21 = B'11')			
		DATA, double or triple length, MAC generate on, MAC verify on (CV bits 20 -21 = B'11')			
		MACVER, double or triple length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')	VER-ONLY ("V")		
		DATAMV, double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01'			

Table 419. Export translation table for a TR-31 ISO MAC algorithm key (ISOMACn) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)

Security considerations: There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "M0", "M1", "M3", or “M6” key, provided that one or both applicable MAC generate and MAC verify control vector bits are on. However, a TR-31 "M0", "M1", "M3", or “M6” key cannot be imported to the lower-security CCA DATA key, it can only be imported to a CCA key type of MAC or MACVER. This restriction eliminates the ability to export a CCA MAC or MACVER key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to Encipher, Decipher, or both.

Notes:

1. MAC keys are used to compute or verify a code for message authentication.
2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:

   "M0" ISO 16609 MAC algorithm 1, TDEA
   The ISO 16609 MAC algorithm 1 is based on ISO 9797. It is identical to "M1", except that it does not support 8-byte DES keys.

   "M1" ISO 9797 MAC algorithm 1
   The ISO 9797 MAC algorithm 1 is identical to "M0", except that it also supports 8-byte DES keys.

   "M3" ISO 9797 MAC algorithm 3
   This is the X9.19 style of Triple-DES MAC.

   "M6"
   The ISO 9797-1:2011 MAC algorithm 5/CMAC.
3. A CCA control vector has no bits defined to limit key usage by algorithm, such as CBC MAC (TR-31 usage "M0" and "M1"), X9.19 (TR-31 usage "M3"), or usage “M6”. When importing a TR-31 key block, the resulting CCA key token deviates from the restrictions of usages "M0", "M1", "M3", and "M6". Importing a TR-31 key block which allows MAC generation ("G" or "C") results in a control vector with the ANY-MAC attribute rather than for the restricted algorithm that is set in the TR-31 key block. The ANY-MAC attribute provides the same restrictions as what CCA currently uses for generating and verifying MACs.

Table 420. Export translation table for a TR-31 EMV/chip card key (DKYGENKY, DATA)

Key usage keyword	Key block protection method keyword	CCA key type and required key usage attributes	Mode of use keyword	Access control name	Offset (hex)
EMVAC-F ("F0")	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DMAC (CV bits 19- 22 = B'0010')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DMAC to F0:N/X	03E4
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DMV (CV bits 19 -22 = B'0011')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DMV to F0:N/X	03E5
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DALL (CV bits 19 -22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DALL to F0:N/X	03E6
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
EMVAC-F ("F1")	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DDATA (CV bits 19- 22 = B'0001')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DDATA to F1:N/X	03F5
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 144 =B'000'), DMPIN (CV bits 19- 22 = B'1001')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DMPIN to F1:N/X	03F6
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DALL (CV bits 19 -22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DALL to F1:N/X	03F7
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		

Table 420. Export translation table for a TR-31 EMV/chip card key (DKYGENKY, DATA) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required key usage attributes	Mode of use keyword	Access control name	Offset (hex)
EMVAC-F ("F2")	VARXOR-A	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 =B'001'), DMAC (CV bits 19- 22 = B'0010')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DMAC to F2:N/X	03F8
	VARDRV-B, VARXOR-C, VARDRV-D		AANY ("N") or DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DALL (CV bits 19 -22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DALL to F2:N/X	03F9
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
EMVAC-F ("F3")	VARXOR-A	DATA, double length	ANY ("N")	T31X - Permit DES DATA/MAC/ CIPHER/ ENCIPHER to F3:N/G/E/X	03FA
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
	VARXOR-A	MAC (not MACVER), double length	ANY ("N")		
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
	VARXOR-A	CIPHER, double length	ANY ("N")		
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
	VARXOR-A	ENCIPHER, double length	ANY ("N")		
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
EMVAC-F ("F4")	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DDATA (CV bits 19- 22 = B'0001')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DDATA to F4:N/X	03FB
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 =B'000'), DALL (CV bits 19 -22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DALL to F4:N/X	03FC
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N") or DERIVE ("X")		

Table 420. Export translation table for a TR-31 EMV/chip card key (DKYGENKY, DATA) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required key usage attributes	Mode of use keyword	Access control name	Offset (hex)

Notes:

1. EMV/chip issuer master-key keys are used by the chip cards to perform cryptographic operations or, in some cases, to derive keys used to perform operations. In CCA, these are (a) diversified key-generating keys (key type DKYGENKY), allowing derivation of operational keys, or (b) operational keys. Note that in this context, "master key" has a different meaning than for CCA. These master keys, also called KMCs, are described by EMV as DES master keys for personalization session keys. They are used to derive the corresponding chip card master keys, and not typically used directly for cryptographic operations other than key derivation. In CCA, these are usually key generating keys with derivation level DKYL1 (CV bits 12 - 14 = B'001'), used to derive other key generating keys (the chip card master keys). For some cases, or for older EMV key derivation methods, the issuer master keys could be level DKYL0 (CV bits 12 - 14 =B'000').
2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:
   - "F0" Application cryptograms.
   - "F1" Secure messaging for confidentiality.
   - "F2" Secure messaging for integrity.
   - "F3" Data authentication code.
   - "F4" Dynamic numbers.
3. EMV support in CCA is quite different than TR-31 support, and CCA key types do not match TR-31 types.
4. DKYGENKY keys are double length only.

Table 421. Export translation table for a TR-31 HMAC algorithm key (MAC)

Key usage keyword	Key block protection method keyword	CCA key type	Required key usage	TR-31 mode of key use keyword	Access control name	Offset (Hex)
HMAC ("M7")	VARDRV-D	MAC	MAC generate on	GEN-ONLY ("G")	TR31 Export - Permit HMAC MAC to M7:G/V/C	020D
			MAC generate off, MAC verify on	VER-ONLY ("V")		
			MAC generate on, MAC verify on	GENVER ("C")		

Security note: The ISOSHA-1 keyword creates an HMAC key block that has a dual meaning.

- For an ISO 20038 implementation, the resulting key block is limited to SHA-1 hash MAC.
- For an ANSI X9 TR-31-2018 implementation, the key does not have any hash algorithm limit because the optional block with identifier 'HM' is not present.

Table 422. Export translation table for a TR-31 PIN encryption or PIN verification key (PINENC, PINVO, PINV3624, VISAPVV)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
PINENC ("P0")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	OPINENC, double or triple length	ENC-ONLY ("E")	T31X - Permit DES OPINENC to P0:E	0191
		DES SECMSG, SMPIN bit on (CV bit 19 = B'1')		T31X – Permit SECMSG:SMPIN to P0	03E2
		IPINENC, double or triple length	DEC-ONLY ("D")	T31X - Permit DES IPINENC to P0:D	0192
		OPINENC, double or triple length	ENCDEC ("B")	T31X - Permit DES OPINENC/IPINENC to P0:B	039E
		IPINENC, double or triple length			
PINVO ("V0")	VARXOR-A, VARDRV-B, VARXOR-C	PINVER, double or triple length, NO-SPEC (CV bits 0 - 4 = B'0000')	ANY ("N") (requires both access controls)	T31X - Permit DES PINVER: NO-SPEC to V0:N/V	0193
				T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N	01B0
	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	PINVER, double or triple length, NO-SPEC (CV bits 0 - 4 = B'0000'), CPINGEN off, EPINGENA off, EPINGEN off, CPINGENA off (CV bits 18 - 21 = B'0000')	VER-ONLY ("V")	T31X - Permit DES PINVER: NO-SPEC to V0:N/V	0193
	VARXOR-A, VARDRV-B, VARXOR-C	PINGEN, double or triple length, NO-SPEC (CV bits 0 - 4 = B'0000')	ANY ("N") (requires both access controls)	T31X - Permit DES PINGEN: NO-SPEC to V0:N/C	0194
				T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N	01B0
	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	PINGEN, double or triple length, NO-SPEC (CV bits 0 - 4 = B'0000'), EPINVER off (CV bit 22 = B'0')	GEN-ONLY ("G")	T31X - Permit DES PINGEN: NO-SPEC to V0:N/C	0194
		PINGEN, double or triple length, NO-SPEC (CV bits 0 - 4 = B'0000'), EPINVER on (CV bit 22 = B'1')	GENVER ("C")		

Table 422. Export translation table for a TR-31 PIN encryption or PIN verification key (PINENC, PINVO, PINV3624, VISAPVV) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
PINV3624 ("V1")	VARXOR-A, VARDRV-B, VARXOR-C	PINVER, double or triple length, NO-SPEC or IBM-PIN/IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001')	ANY ("N") (requires both access controls)	T31X - Permit DES PINVER: NO-SPEC/ IBM-PIN/IBM-PINO to V1:N/V	0195
				T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/ V2:N	01B0
	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	PINVER, double or triple length, NO-SPEC or IBM-PIN/ IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001'), CPINGEN off, EPINGENA off, EPINGEN off, CPINGENA off (CV bits 18 - 21 = B'0000')	VER-ONLY ("V")	T31X - Permit DES PINVER: NO-SPEC/ IBM-PIN/IBM-PINO to V1:N/V	0195
	VARXOR-A, VARDRV-B, VARXOR-C	PINGEN, double or triple length, NO-SPEC or IBM-PIN/IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001')	ANY ("N") (requires both access controls)	T31X - Permit DES PINGEN: NO-SPEC/ IBM-PIN/IBM-PINO to V1:N/V	0196
				T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/ V2:N	01B0
	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	PINGEN, double or triple length, NO-SPEC or IBM-PIN/IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001'), EPINVER off (CV bit 22 = B'0')	GEN-ONLY ("G")	T31X - Permit DES PINGEN: NO-SPEC/ IBM-PIN/IBM-PINO to V1:N/V	0196
		PINGEN, double or triple length, NO-SPEC or IBM-PIN/IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001'), EPINVER on (CV bit 22 = B'1)	GENVER ("C")		

Table 422. Export translation table for a TR-31 PIN encryption or PIN verification key (PINENC, PINVO, PINV3624, VISAPVV) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
VISAPVV ("V2")	VARXOR-A, VARDRV-B, VARXOR-C	PINVER, double or triple length, NO-SPEC or VISA-PVV (CV bits 0 - 4 = B'0000' or B'0010')	ANY ("N") (requires both access controls)	T31X - Permit DES PINVER: NO-SPEC/VISA-PVV to V2:N/V	0197
				T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N	01B0
	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	PINVER, double or triple length, NO-SPEC or VISA-PVV (CV bits 0 - 4 = B'0000' or B'0010'), CPINGEN off, EPINGENA off, EPINGEN off, CPINGENA off (CV bits 18 - 21 = B'0000')	VER-ONLY ("V")	T31X - Permit DES PINVER: NO-SPEC/VISA-PVV to V2:N/V	0197
	VARXOR-A, VARDRV-B, VARXOR-C	PINGEN, double or triple length, NO-SPEC or VISA-PVV (CV bits 0 - 4 = B'0000' or B'0010')	ANY ("N") (requires both access controls)	T31X - Permit DES PINGEN: NO-SPEC/VISA-PVV to V2:N/C	0198
				T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N	01B0
	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	PINGEN, double or triple length, NO-SPEC or VISA-PVV (CV bits 0 - 4 = B'0000' or B'0010'), EPINVER off (CV bit 22 = B'0')	GEN-ONLY ("G")	T31X - Permit DES PINGEN: NO-SPEC/VISA-PVV to V2:N/C	0198
		PINGEN, double or triple length, NO-SPEC or VISA-PVV (CV bits 0 - 4 = B'0000' or B'0010'), EPINVER on (CV bit 22 = B'1')	GENVER ("C")		

Security notes:

1. There is asymmetry in the SECMSG:SMPIN -> P0 translation. There is no way to translate a TR-31 P0 key to a CCA SECMSG:SMPIN.
2. It is highly recommended that the INCL-CV keyword be used when exporting PINGEN, PINVER, IPINENC or OPINENC keys. Using this keyword ensures that importing the TR-31 key block back into CCA will have the desired attributes.

3. TR-31 key blocks that are protected under legacy version ID "A" (keyword VARXOR-A, using the Key Variant Binding Method 2005 Edition) use the same mode of use "N" (keyword ANY) for PINGEN and PINVER keys. For version ID "A" keys only, for a given PIN key usage, enabling both the PINGEN and PINVER access controls at the same time while enabling offset X'01B0' (for mode "N") is NOT recommended. In other words, for a particular PIN verification usage, you should not simultaneously enable the four commands shown below for that usage:

Failure to comply with this recommendation allows changing PINVER keys into PINGEN and the other way around.

	Key type, mode, or version	**Offset**	**Access control**
For usage "V0", a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. Avoid simultaneously enabling these four controls.			
"V0"	Key type PINVER	X'0193'	T31X - Permit DES PINVER: NO-SPEC to V0:N/V
	Key type PINGEN	X'0194'	T31X - Permit DES PINGEN: NO-SPEC to V0:N/C
	Mode ANY	X'01B0'	T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N
	Version VARXOR-A	X'014D'	T31X - Permit Version A TR-31 Key Blocks
For usage "V1", a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. Avoid simultaneously enabling these four controls.			
"V1"	Key type PINVER	X'0195'	T31X - Permit DES PINVER: NO-SPEC/ IBM-PIN/IBM-PINO to V1:N/V
	Key type PINGEN	X'0196'	T31X - Permit DES PINGEN: NO-SPEC/ IBM-PIN/IBM-PINO to V1:N/V
	Mode ANY	X'01B0'	T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N
	Version VARXOR-A	X'014D'	T31X - Permit Version A TR-31 Key Blocks
For usage "V2", a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. Avoid simultaneously enabling these four controls.			
"V2"	Key type PINVER	X'0197'	T31X - Permit DES PINVER: NO-SPEC/ VISA-PVV to V2:N/V
	Key type PINGEN	X'0198'	T31X - Permit DES PINGEN: NO-SPEC/ VISA-PVV to V2:N/C
	Mode ANY	X'01B0'	T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N
	Version VARXOR-A	X'014D'	T31X - Permit Version A TR-31 Key Blocks

Notes:

1. PIN encryption keys are used to protect PIN blocks. PIN verification keys are used to generate or verify a PIN using a particular PIN-calculation method for that key type.
2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:

"P0"
PIN encryption.

"V0"
PIN verification, KPV, other algorithm. "V0" is intended to be a PIN-calculation method "other" than those defined for "V1" or "V2". Because CCA does not have a PIN-calculation method of "other" defined, it maps usage "V0" to the subtype extension of NO-SPEC (CV bits 0 - 3 = B'0000'). Be aware that NO-SPEC allows any method, including "V1" and "V2", and that this mapping is suboptimal.

"V1"
PIN verification, IBM 3624.

"V2"
PIN verification, Visa PVV.

3. Mode must be one of the following:

"E"
Encrypt/wrap only. This restricts PIN encryption keys to encrypting a PIN block. May be used to create or reencipher an encrypted PIN block (for key-to-key translation).

"D"
Decrypt/unwrap only. This restricts PIN encryption keys to decrypting a PIN block. Generally used in a PIN translation to decrypt the incoming PIN block.

"N"
No special restrictions (other than restrictions implied by the key usage). This is used by several vendors for a PIN generate or PIN verification key when the key block version ID is "A".

"G"
Generate only. This is used for a PINGEN key that may not perform a PIN verification. This is the only mode available when the control vector in the CCA key-token (applicable when INCL-CV keyword is not provided) does NOT have the EPINVER control vector bit on.

"V"
Verify only. This is used for PIN verification only. This is the only mode available when the control vector in the CCA key-token (applicable when INCL-CV is not provided) ONLY has the EPINVER control vector usage bit on (CV bits 18 - 22 = B'00001').

"C"
Both generate and verify (combined). This is the only output mode available for TR-31 when any of the CCA key-token PIN generating bits are on in the control vector (CPINGENA, EPINGENA, EPINGEN, or CPINGENA) in addition to the EPINVER bit.

Table 423. Export translation table for a TR-31 EMV/chip issuer master-key key (DKYGENKY, DATA)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
EMVACMK ("E0")	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DMAC to E0:N/X	0199
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMV (CV bits 19 - 22 = B'0011')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DMV to E0:N/X	019A
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DALL to E0:N/X	019B
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMAC (CV bits 19 - 22 = B'0010')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL1+DMAC to E0:N/X	019C
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMV (CV bits 19 - 22 = B'0011')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL1+DMV to E0:N/X	019D
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DALL (CV bits 19 - 22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL1+DALL to E0:N/X	019E
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
EMVSCMK ("E1")	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DDATA to E1:N/X	019F
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 144 = B'000'), DMPIN (CV bits 19 - 22 = B'1001')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DMPIN to E1:N/X	01A0
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DALL to E1:N/X	01A1
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DDATA (CV bits 19 - 22 = B'0001')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL1+DDATA to E1:N/X	01A2
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL1 (CV bits 12 - 144 = B'001'), DMPIN (CV bits 19 - 22 = B'1001')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL1+DMPIN to E1:N/X	01A3
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DALL (CV bits 19 - 22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL1+DALL to E1:N/X	01A4
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		

Table 423. Export translation table for a TR-31 EMV/chip issuer master-key key (DKYGENKY, DATA) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
EMVSIMK ("E2")	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DMAC to E2:N/X	01A5
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DALL to E2:N/X	01A6
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMAC (CV bits 19 - 22 = B'0010')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL1+DMAC to E2:N/X	01A7
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DALL (CV bits 19 - 22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL1+DALL to E2:N/X	01A8
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
EMVDAMK ("E3")	VARXOR-A	DATA, double length	ANY ("N")	T31X - Permit DES DATA/DATAM/ CIPHER/MAC/ENCIPHER to E3:N/G/E/X	01A9
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	MAC (not MACVER), double length	ANY ("N")		
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	CIPHER, double length	ANY ("N")		
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	ENCIPHER, double length	ANY ("N")		
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
EMVDNMK ("E4")	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DDATA to E4:N/X	01AA
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DALL to E4:N/X	01AB
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		

Table 423. Export translation table for a TR-31 EMV/chip issuer master-key key (DKYGENKY, DATA) (continued)

Key usage keyword	Key block protection method keyword	CCA key type and required control vector attributes	Mode of use keyword	Access control name	Offset (hex)
EMVCPMK ("E5")	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DEXP (CV bits 19 - 22 = B'0101')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DEXP to E5:N/X	01AC
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DMAC to E5:N/X	01AD
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')	ANY ("N")	T31X - Permit DES DKYGENKY: DKYL0+DDATA to E5:N/X	01AE
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		
	VARXOR-A	DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')	ANY ("N")	T31X - Permit DES DKYGENKY:DKYL0+DALL to E5:N/X	01AF
	VARDRV-B, VARXOR-C, VARDRV-D		ANY ("N"), DERIVE ("X")		

Notes:

1. EMV/chip issuer master-key keys are used by the chip cards to perform cryptographic operations or, in some cases, to derive keys used to perform operations. In CCA, these are (a) diversified key-generating keys (key type DKYGENKY), allowing derivation of operational keys, or (b) operational keys. Note that in this context, "master key" has a different meaning than for CCA. These master keys, also called KMCs, are described by EMV as DES master keys for personalization session keys. They are used to derive the corresponding chip card master keys,and not typically used directly for cryptographic operations other than key derivation. In CCA, these are usually key generating keys with derivation level DKYL1 (CV bits 12 - 14 = B'001'), used to derive other key generating keys (the chip card master keys). For some cases, or for older EMV key derivation methods, the issuer master keys could be level DKYL0 (CV bits 12 - 14 = B'000').
2. The following defines the only supported translations for this TR-31 usage. Usage must be one of the following:

   "E0"
   Application cryptograms.

   "E1"
   Secure messaging for confidentiality.

   "E2"
   Secure messaging for integrity.

   "E3"
   Data authentication code.

   "E4"
   Dynamic numbers.

   "E5"
   Card personalization.
3. EMV support in CCA is quite different than TR-31 support, and CCA key types do not match TR-31 types.
4. DKYGENKY keys are double length only.

Table 424. Export translation table for a TR-31 key with proprietary DK key usage

CCA key type (required attributes)	Key usage keyword	Key block protection method keyword	Mode of use keyword	Access control name	Offset (hex)
AES KDKGENKY (KDKTYPEB)	TYPBTO10 ("10")	VARDRV-D	DERIVE ("X")	T31X - Permit AES KDKGENKY: KDKTYPEB to 10:X	0384

Table 424. Export translation table for a TR-31 key with proprietary DK key usage (continued)

CCA key type (required attributes)	Key usage keyword	Key block protection method keyword	Mode of use keyword	Access control name	Offset (hex)
AES KDKGENKY (KDKTYPEA)	TYPATO11 ("11")	VARDRV-D	DERIVE ("X")	T31X - Permit AES KDKGENKY: KDKTYPEA to 11:X	0383
DES DKYGENKY (DKYL0 and DMPIN; CV bits 12 - 14 = B'000' and 19 - 22 = B'1001')	DMP0TO12 ("12")	VARDRV-D	DERIVE ("X")	T31X - Permit DES DKYGENKY: DKYL0:DMPIN to 12:X	0385

Table 425. Export translation table for an AES TR-31 key

CCA key type (required attributes)	Key usage keyword	Key block protection method keyword	Mode of use keyword	Access control name	Offset (hex)
AES CIPHER Encrypt/ decrypt modes require matching key usage	ENC ("D0")	VARDRV-D	ENC-ONLY ("E") DEC-ONLY ("D") ENCDEC ("B")	T31X - Permit AES CIPHER to D0:E/D/B	01D0
AES MAC (CMAC) Generate/verify modes require matching key usage	ISOMAC6 ("M6")	VARDRV-D	GEN-ONLY ("G") VER-ONLY ("V") GENVER ("C")	T31X - Permit AES MAC: CMAC to M6:G/C/V	01D1
AES PINPROT Encrypt/ decrypt modes require matching key usage	PINENC ("P0")	VARDRV-D	ENC-ONLY ("E") DEC-ONLY ("D")	T31X - Permit AES PINPROT to P0:E/D	01D2
AES EXPORTER	KEK ("K0")	VARDRV-D	ENC-ONLY ("E")	T31X - Permit AES EXPORTER to K0:E	01D3
AES EXPORTER (EXPTT31D)	KEK-WRAP ("K1")	VARDRV-D	ENC-ONLY ("E")	T31X - Permit AES EXPORTER to K1:E	01D4
AES EXPORTER (EXPTT31D)	KEK-WRK4 ("K4")	VARDRV-D	ENC-ONLY ("E")	T31X - Permit AES EXPORTER to K4:E	01D5
AES IMPORTER	KEK ("K0")	VARDRV-D	DEC-ONLY ("D")	T31X - Permit AES IMPORTER to K0:D	01D6
AES IMPORTER (IMPTT31D)	KEK-WRAP ("K1")	VARDRV-D	DEC-ONLY ("D")	T31X - Permit AES IMPORTER to K1:D	01D7
AES IMPORTER (IMPTT31D)	KEK-WRK4 ("K4")	VARDRV-D	DEC-ONLY ("D")	T31X - Permit AES IMPORTER to K4:D	01D8
AES DKYGENKY (DKYL0 or DKYL1 or DKYL2; D-MAC or D-ALL) EMV Issuer Master Key: App Cryptograms	EMVACMK ("E0")	VARDRV-D	DERIVE ("X")	T31X - Permit AES DKYGENKY:D-ALL/DMAC to E0:X	01D9
AES DKYGENKY (DKYL0 or DKYL1 or DKYL2; D-SECMSG or D-ALL) EMV Issuer Master Key: Sec Msg for Confidentiality	EMVSCMK ("E1")	VARDRV-D	DERIVE ("X")	T31X - Permit AES DKYGENKY:D-ALL/D-SECMSG to E1:X	01DA

Table 425. Export translation table for an AES TR-31 key (continued)

CCA key type (required attributes)	Key usage keyword	Key block protection method keyword	Mode of use keyword	Access control name	Offset (hex)
AES DKYGENKY (DKYL0 or DKYL1 or DKYL2; D-MAC or D-ALL) EMV Issuer Master Key: Sec Msg for Integrity	EMVSIMK ("E2")	VARDRV-D	DERIVE ("X")	T31X - Permit AES DKYGENKY:D-ALL/D-MAC to E2:X	01DB
AES CIPHER (no required attributes) or AES DKYGENKY (DKYL0 or DKYL1 or DKYL2; D-CIPHER or D-ALL) • EMV Issuer Master Key: Data Auth Code. May be used directly to create DAC , or for derivation. • Encrypt mode is required for CIPHER, DAC is created with encrypt.	EMVDAMK ("E3")	VARDRV-D	CIPHER: ENC-ONLY ("E") ENCDEC ("B") DKYGENKY: DERIVE ("X")	T31X - Permit AES CIPHER to E3/E/B,DKYGENKY:D-ALL/DCIP to E3:X	01DC
AES DKYGENKY (DKYL0 or DKYL1 or DKYL2; D-CIPHER or D-ALL) EMV Issuer Master Key: Dynamic Numbers	EMVDNMK ("E4")	VARDRV-D	DERIVE ("X")	T31X - Permit AES DKYGENKY:D-ALL/D-CIPHER to E4:X	01DD
AES DKYGENKY + (DKYL0 or DKYL1 or DKYL2l D-MAC or D-ALL) EMV Issuer Master Key: Card Personalization	EMVCPMK ("E5")	VARDRV-D	DERIVE ("X")	T31X - Permit AES DKYGENKY:D-MAC to E5:X	01DE

Table 426. Export translation table for CCA key types

CCA key type (required attributes)	Key usage keyword	Key block protection method keyword	Mode of use keyword	Access control name	Offset (hex)
AES DKYGENKY (DKYL0; D-MAC or D-ALL) EMV Issuer Master Key: App Cryptograms	EMVAC-F ("F0")	VARDRV-D	DERIVE ("X")	T31X - Permit AES DKYGENKY:D-ALL/DMAC to F0:X	03FD
AES DKYGENKY (DKYL0; D-SECMSG or D-ALL) EMV Issuer Master Key: Sec Msg for Confidentiality	EMVSC-F ("F1")	VARDRV-D	DERIVE ("X")	T31X - Permit AES DKYGENKY:D-ALL/DCIPHER to F1:X	03FE

Table 426. Export translation table for CCA key types (continued)

CCA key type (required attributes)	Key usage keyword	Key block protection method keyword	Mode of use keyword	Access control name	Offset (hex)
AES DKYGENKY (DKYL0; D-MAC or D-ALL) EMV Issuer Master Key: Sec Msg for Integrity	EMVSI-F ("F2")	VARDRV-D	DERIVE ("X")	T31X - Permit AES DKYGENKY:D-ALL/DMAC to F2:X	03FF
AES CIPHER (no required attributes) or AES DKYGENKY (DKYL0; D-CIPHER or D-ALL) • EMV Issuer Master Key: Data Auth Code. May be used directly to create DAC or for derivation. • Encrypt mode is required for CIPHER. DAC is created with encrypt.	EMVDA-F ("F3")	VARDRV-D	CIPHER: ENC-ONLY ("E") ENCDEC ("B") DKYGENKY: DERIVE ("X")	T31X - Permit AES CIPHER, DKYGENKY:D-ALL/DCIPHER to F3:E/B/X	0500
AES DKYGENKY (DKYL0; D-CIPHER or D-ALL) EMV Issuer Master Key: Dynamic Numbers	EMVDN-F ("F4")	VARDRV-D	DERIVE ("X")	T31X - Permit AES DKYGENKY:D-ALL/DCIPHER to F4:X	0501

key_version_number

Direction	Type
Input	String

The two bytes from this parameter are copied into the Key Version Number field of the output TR-31 key block. If no key version number is needed, the value must be EBCDIC ("00"). If the CCA key in parameter *source_key_identifier* is a key part (CV bit 44 is 1) then the key version number in the TR-31 key block is set to "C0" (0x6330) according to the TR-31 standard, which indicates that the TR-31 block contains a key part. In this case, the value passed to the callable service in the *key_version_number* parameter is ignored. When TR-31 usage keyword is INITVEC, the key version number cannot be "C0".

key_field_length

Direction	Type
Input	Integer

This parameter specifies the length of the key field which is encrypted in the TR-31 block. This value does not allow for ASCII encoding of the encrypted data stored in the key field according to the TR-31 specification.

For example, when a value of 32 is specified here, the length of the final ASCII-encoded encrypted data in the key field in the output TR-31 key block is 64 bytes.

Table 427 on page 1046 shows the recommended values for the *key_field_length* parameter. They are determined based on the cipher block size of the underlying algorithm used to wrap the key block and the minimum number of pad bytes for the maximum key size that can be wrapped (32 bytes for DES, 48 bytes for AES, and 272 bytes for HMAC).

Table 427. Recommended values for the key_field_length parameter

Key block version ID	Cipher block size in bytes	Key algorithm	Recommended *key_field_length* value
"A" (DES wrap KEK)	8	DES	32
"B" (DES wrap KEK)			
"C" (DES wrap KEK)			
"D" (AES wrap KEK)	16	DES	32
		AES	48
		HMAC	272

source_key_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *source_key_identifier* parameter, in bytes.

When the TR-31 usage keyword is INITVEC, the length is 8 for a DES/TDES initialization vector or 16 for an AES initialization vector.

For other usage keywords, when the *source_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 9992.

source_key_identifier

Direction	Type
Input/Output	String

This is either a key or a clear initialization vector (TR-31 usage keyword INITVEC).

When this is a key, the key identifier is an internal or external CCA key token, X9.143 (TR-31) key block, or the label of an operational token or block in key storage. If the source key is an external token, an identifier for the KEK that wraps the source key must be passed in the *unwrap_kek_identifier* parameter.

The key must be an internal or external CCA fixed-length DES key token, CCA variable-length AES or HMAC key token, or TR-31 key block.

When the COMP-CHK or COMP-TAG rule is specified, the key must be an internal DES or AES TR-31 key block.

When an internal token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

unwrap_kek_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *unwrap_kek_identifier* parameter, in bytes. The value must be zero if the *source_key_identifier* is an internal key or an initialization vector (TR-31 usage keyword INITVEC).

The value must be 64 if the *unwrap_kek_identifier* is specified by label.

Otherwise, the value must be between the actual length of the token and 9992.

unwrap_kek_identifier

Direction	Type
Input/Output	String

The key that wraps the external token in the *source_key_identifier* parameter. The key identifier is an operational key token or key block or the label of an operational token or block in key storage.

When the *unwrap_kek_identifier_length* is zero, this parameter is ignored.

When the source key is DES, the unwrap KEK must be one of the following:

- A CCA DES EXPORTER with the control vector bit EXPORT enabled or OKEYXLAT key.
- A TR-31 DES or AES exporter. Key usage K0 or K1, algorithm A or T, and mode of use E.
  - K0 may be used whether source key is a CCA key token or TR-31 key block.
  - K1 is valid only when source key is a TR-31 key block.

When the source key is AES or HMAC, the unwrap KEK must be one of the following:

- A CCA AES EXPORTER with WR-AES or WR-HMAC capability, matching the wrapped key.
- A TR-31 AES exporter. Key usage K0 or K1, algorithm A, and mode of use E.
  - K0 may be used whether source key is a CCA key token or TR-31 key block.
  - K1 is valid only when source key is a TR-31 key block.

When the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

wrap_kek_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *wrap_kek_identifier* parameter, in bytes.

The value must be zero if the *wrap_kek_identifier* parameter is not used.

The value must be 64 if the *wrap_kek_identifier* is specified by label.

Otherwise, the value must be between the actual length of the token and 9992.

wrap_kek_identifier

Direction	Type
Input/Output	String

The identifier of the key to wrap the TR-31 key block. The key identifier is an operational key token or the label of an operational token in key storage.

When *wrap_kek_identifier_length* is 0, this parameter is ignored.

If creating an external key and this parameter is not specified, the *unwrap_kek_identifier* will be used to wrap the output key. In that case, the *unwrap_kek_identifier* must also satisfy the requirements of the *wrap_kek_identifier* for the operation to be successful.

For key block protection methods VARDRV-A, VARDRV-B, or VARDRV-C, the key is one of the following:

- A CCA DES EXPORTER with the control vector bit EXPORT enabled or OKEYXLAT key.
- A TR-31 DES or AES exporter. Key usage K1, algorithm A or T, and mode of use E.

For key block protection method VARDRV-D, the key is one of the following:

- A CCA AES key-encrypting key of type EXPORTER with key usages EXPTT31D, VARDRV-D, and with WR-DES, WR-AES, or WR-HMAC capability, matching the wrapped key/initialization vector, and one of the following if necessary:
  - WRDERIVE when key usage value keyword DUKPT, KDK, EMVACMK, EMVSCMK, EMVSIMK, EMVDAMK, EMVDNMK, EMVCPMK, EMVAC-F, EMVSC-F, EMVSI-F, EMVDA-F, EMVDN-F, TYPBTO10, or TYPATO11 is specified in the rule array.
  - WR-DATA when key usage value keyword CVK, ENC, ENCSENS, ISOMAC6, HMAC, or INITVEC is specified in the rule array.
  - WR-KEK when key usage value keyword KEK, KEK-WRAP, or KEK-WRK4 is specified in the rule array.
  - WR-PIN when key usage keyword PINENC, PINVO, PINV3624, or VISAPVV is specified in the rule array.
- A TR-31 AES exporter. Key usage K1, algorithm A, and mode of use E.

When the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

Note: ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap/unwrap TR-31 version 'B'/'C' key blocks that have/will have 'E' exportability, because ECB-mode does not comply with ANSI X9.24 Part 1.

This parameter exists to allow for KEK separation, it is possible that KEKs will be restricted as to what they can wrap, such that a KEK for wrapping CCA external keys may not be usable for wrapping TR-31 external keys, or vice versa.

opt_blks_length

Direction	Type
Input	Integer

This parameter specifies the length of parameter *opt_blocks* in bytes. If no optional data is to be included in the TR-31 key block, this parameter must be set to zero.

opt_blocks

Direction	Type
Input	String

This parameter contains optional block data which is to be included in the output TR-31 key block. The optional block data is prepared using the TR-31 Optional Data Build callable service and must be in ASCII. This parameter is ignored if *opt_blks_length* is zero.

If keywords TYPATO10, TYPBTO11, or DMP0TO12 are specified, the ICSF will build the proprietary DK optional block based on the type of key being exported.

Optional blocks specified here are mutually exclusive with the OB-*xx* keywords. If OB-*xx* keywords are specified in the rule array, ensure that these specific optional blocks are not also present in this parameter. You cannot use this parameter to add the following optional blocks: TC, TS, WP, KP, KC, HM, IBM proprietary optional block (10), BI, IK, and PB. It is recommended that you use the OB-*xx* keywords to add these optional blocks if allowed.

TR31_key_block_length

Direction	Type
Input/Output	Integer

This parameter specifies the length of the *TR31_key_block* parameter, in bytes. On input, it must specify the size of the buffer available for the output TR-31 key block, and on return it is updated to contain the actual length of that returned key block. If the provided buffer is not large enough for the output TR-31 key block an error is returned. The maximum size of the output TR-31 key block is 9992 bytes.

If the COMP-CHK keyword is specified, this value must be 0.

TR31_key_block

Direction	Type
Output	String

This parameter specifies the location of the exported TR-31 key block wrapped with the export key provided in the *wrap_kek_identifier* parameter.

Restrictions

Proprietary values for the TR-31 header fields are not supported by this callable service with the exception of the proprietary values used by IBM CCA when carrying a control vector in an optional block in the header.

Usage notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned. TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

If keyword INCL-CV or ATTR-CV is specified, the service inserts the CCA control vector from the source key into an optional data field in the TR-31 header. The TR-31 Import callable service can extract this CV and use it as the CV for the CCA key it creates when importing the TR-31 block. This provides a way to use TR-31 for transport of CCA keys and to make the CCA key have identical control vectors on the sending and receiving nodes. The difference between INCL-CV and ATTR-CV is that INCL-CV is a normal TR-31 Translate in which the TR-31 key attributes are set based on the supplied rule array keywords but the CV is also included in the TR-31 block to provide additional detail. In contrast, the ATTR-CV causes the service to include the CV but to set both the TR-31 usage and mode of use fields to proprietary values which indicate that the usage and mode information are specified in the CV and not in the TR-31 header. For option INCL-CV, the export operation is still subject to the restrictions imposed by the settings of the relevant access control points. For option ATTR CV, those access control points are not checked and any CCA key can be exported as long as the export control fields in the CV permit it.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Note that the optional data, if present, must not already contain a padding Block, ID "PB". A Padding Block of the appropriate size, if needed, will be added when building the TR-31 key block. If this callable service encounters a padding block in the optional block data, an error will occur.

Refer to the PDF version of this book for a list the valid attribute translations for export of CCA keys to TR-31 key blocks along with the access control points which govern those translations.

If ICSF is configured to audit the lifecycle of tokens [AUDITKEYLIFECKDS(TOKEN(YES),...) is specified] and a token is passed as input to be exported, an additional request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the exported key.

Access control points

The access control points in the domain role that control the general function of this service are:

Rule array keyword	Access control name
VARXOR-A	T31X - Permit version A TR-31 key blocks
VARDRV-B	T31X - Permit version B TR-31 key blocks
VARXOR-C	T31X - Permit version C TR-31 key blocks
VARDRV-D	T31X - Permit version D TR-31 key blocks
INCL-CV	T31X - Permit any CCA key if INCL-CV is specified

If the wrap KEK identifier is a weaker key than the key being exported, then:

- The service will fail with return code 8 reason code 39 (X'27') if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- The service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

Table 428 on page 1050 lists the valid attribute translations for export of CCA keys to TR-31 key blocks along with the access control points which govern those translations. Any translation not listed here will result in an error. If an individual cell is blank, it represents the value of the cell immediately above it.

Note: In order to export a CCA key to a TR-31 key block, the appropriate key block version ACP needs to be enabled in addition to any required translation specific ACPs.

Table 428. Valid CCA to TR-31 Translate Translations and Required Access Controls

TR-31 key usage keyword	Access control name	Offset (hex)	Specific key type and control vector attributes
"B0": TR-31 BDK base derivation keys			
BDK	T31X - Permit DES KEYGENKY: DUKPT to B0:N/X	0180	See Table 415 on page 1020.
	T31X - Permit AES DKYGENKY: DUKPT BDK to B0:X	01CF	
"C0": TR-31 CVK card verification keys			
CVK	T31X - Permit DES MAC/MACVER:AMEX-CSC to C0:G/C/V	0181	See Table 416 on page 1022.
	T31X - Permit DES MAC/MACVER: CVV-KEYA to C0:G/C/V	0182	
	T31X - Permit DES MAC/MACVER: ANY-MAC to C0:G/C/V	0183	
	T31X - Permit DES DATA/DATAM/DATAMV to C0:G/C/V	0184	
"D0": TR-31 data encryption keys			
ENC	T31X - Permit DES ENCIPHER/DECIPHER/CIPHER to D0:E/D/B	0185	See Table 417 on page 1024.
	T31X - Permit DES DATA to D0:E/D/B	0186	
"E0", "E1", "E2", "E3", "E4", and "E5": TR-31 EMC/chip issuer master-key keys			

Table 428. Valid CCA to TR-31 Translate Translations and Required Access Controls (continued)

TR-31 key usage keyword	Access control name	Offset (hex)	Specific key type and control vector attributes
EMVACMK	T31X - Permit DES DKYGENKY: DKYL0+DMAC to E0:N/X	0199	See Table 418 on page 1025.
	T31X - Permit DES DKYGENKY: DKYL0+DMV to E0:N/X	019A	
	T31X - Permit DES DKYGENKY: DKYL0+DALL to E0:N/X	019B	
	T31X - Permit DES DKYGENKY: DKYL1+DMAC to E0:N/X	019C	
	T31X - Permit DES DKYGENKY: DKYL1+DMV to E0:N/X	019D	
	T31X - Permit DES DKYGENKY: DKYL1+DALL to E0:N/X	019E	
EMVSCMK	T31X - Permit DES DKYGENKY: DKYL0+DDATA to E1:N/X	019F	See Table 418 on page 1025.
	T31X - Permit DES DKYGENKY: DKYL0+DMPIN to E1:N/X	01A0	
	T31X - Permit DES DKYGENKY: DKYL0+DALL to E1:N/X	01A1	
	T31X - Permit DES DKYGENKY: DKYL1+DDATA to E1:N/X	01A2	
	T31X - Permit DES DKYGENKY: DKYL1+DMPIN to E1:N/X	01A3	
	T31X - Permit DES DKYGENKY: DKYL1+DALL to E1:N/X	01A4	
EMVSIMK	T31X - Permit DES DKYGENKY: DKYL0+DMAC to E2:N/X	01A5	See Table 418 on page 1025.
	T31X - Permit DES DKYGENKY: DKYL0+DALL to E2:N/X	01A6	
	T31X - Permit DES DKYGENKY: DKYL1+DMAC to E2:N/X	01A7	
	T31X - Permit DES DKYGENKY: DKYL1+DALL to E2:N/X	01A8	
EMVDAMK	T31X - Permit DES DATA/DATAM/CIPHER/MAC/ENCIPHER to E3:N/G/E/X	01A9	See Table 418 on page 1025.
EMVDNMK	T31X - Permit DES DKYGENKY: DKYL0+DDATA to E4:N/X	01AA	See Table 418 on page 1025.
	T31X - Permit DES DKYGENKY: DKYL0+DALL to E4:N/X	01AB	
EMVCPMK	T31X - Permit DES DKYGENKY: DKYL0+DEXP to E5:N/X	01AC	See Table 418 on page 1025.
	T31X - Permit DES DKYGENKY: DKYL0+DMAC to E5:N/X	01AD	
	T31X - Permit DES DKYGENKY: DKYL0+DDATA to E5:N/X	01AE	
	T31X - Permit DES DKYGENKY:DKYL0+DALL to E5:N/X	01AF	
"K0" and "K1": TR-31 key encryption or wrapping, or key block protection keys			
KEK	T31X - Permit DES EXPORTER/OKEYXLAT to K0:E	0187	See Table 419 on page 1027.
	T31X - Permit DES IMPORTER/IKEYXLAT to K0:D	0188	
	T31X - PERMIT EXPORTER to K0:B	02AD	
	T31X - PERMIT IMPORTER to K0:B	02AE	
KEK-WRAP	T31X - Permit DES EXPORTER/OKEYXLAT to K1/K4:E	0189	
	T31X - Permit DES IMPORTER/IKEYXLAT to K1/K4:D	018A	
"M0", "M1", and "M3": TR-31 ISO MAC algorithm keys			
ISOMAC0	T31X - Permit DES MAC/DATA/DATAM to M0:G/C	018B	See Table 422 on page 1035.
	T31X - Permit DES MACVER/DATA/DATAMV to M0:V	018C	
ISOMAC1	T31X - Permit DES MAC/DATA/DATAM to M1:G/C	018D	
	T31X - Permit DES MACVER/DATA/DATAMV to M1:V	018E	
ISOMAC3	T31X - Permit DES MAC/DATA/DATAM to M3:G/C	018F	
	T31X - Permit DES MACVER/DATA/DATAMV to M3:V	0190	
"M7": TR-31 HMAC keys			

Table 428. Valid CCA to TR-31 Translate Translations and Required Access Controls (continued)			
TR-31 key usage keyword	**Access control name**	**Offset (hex)**	**Specific key type and control vector attributes**
HMAC	TR31 Export - Permit HMAC MAC to M7:G/V/C	020D	See Table 421 on page 1034.
"P0", "V0", "V1": TR-31 PIN encryption or PIN verification keys			
PINENC	T31X - Permit DES OPINENC to P0:E	0191	See Table 423 on page 1040.
	T31X - Permit DES IPINENC to P0:D	0192	
	T31X - Permit DES OPINENC/IPINENC to P0:B	039E	
PINVO	T31X - Permit DES PINVER: NO-SPEC to V0:N/V	0193	
	T31X - Permit DES PINGEN: NO-SPEC to V0:N/C	0194	
	TT31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N	01B0	
PINV3624	T31X - Permit DES PINVER: NO-SPEC/IBM-PIN/IBM-PINO to V1:N/V	0195	
	T31X - Permit DES PINGEN: NO-SPEC/IBM-PIN/IBM-PINO to V1:N/V	0196	
	T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N	01B0	
VISAPVV	T31X - Permit DES PINVER: NO-SPEC/VISA-PVV to V2:N/V	0197	
	T31X - Permit DES PINGEN: NO-SPEC/VISA-PVV to V2:N/C	0198	
	T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N	01B0	

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 429. TR-31 Translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Keywords SKEY-AES, SKEY-DES, VARDRV-D, TYPATO11, TYPBTO10, and DMP0TO12 and triple-length DES keys require the July 2019 or later licensed internal code (LIC). Keywords INITVEC, IV-DES, IV-TDES, and IV-AES are not supported. Compliant-tagged key tokens are not supported. The combination of keywords PINENC and ENCDEC requires a crypto express coprocessor with the June 2020 or later licensed internal code (LIC). Keywords HMAC, SKEYHMAC, ISOSHA-1, ISOSHA-2, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512, and TR-31 key block containing an HMAC key require the June 2020 or later licensed internal code. Export of AES DKYGENKY DUKPT BDK to key block B0 is not supported. Mode of use "N" with B, C, and D key block version IDs and Mode of use "B" for K0 export is not supported. Keywords COMP-CHK, COMP-TAG, STOREXCH, INTERNAL, and EXCHANGE are not supported. Keywords DUKPT, KDK, ENCSENS, EMVAC-F, EMVSC-F, EMVSI-F, EMVDA-F, and EMVDN-F are not supported. Keywords XPRTCPAC, NOEXCPAC, and DKPINOP are not supported. Keywords OB-DA, OB-LB, OB-IBM, OB-KC, OB-KP, OB-TC, OB-TS, OB-WP are not supported. Keyword ISOMAC6 is not supported with a DES key. Keyword KEK is not supported with a SECMSG key. Keyword PINENC is not supported with a SECMSG key. Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.

Table 429. TR-31 Translate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Keywords SKEY-AES, SKEY-DES, VARDRV-D, TYPATO11, TYPBTO10, and DMP0TO12 and triple-length DES keys require the July 2019 or later licensed internal code (LIC). Keywords INITVEC, IV-DES, IV-TDES, and IV-AES are not supported. Compliant-tagged key tokens are not supported. The combination of keywords PINENC and ENCDEC requires a crypto express coprocessor with the June 2020 or later licensed internal code (LIC). Keywords HMAC, SKEYHMAC, ISOSHA-1, ISOSHA-2, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512, and TR-31 key block containing an HMAC key require the June 2020 or later licensed internal code. Export of AES DKYGENKY DUKPT BDK to key block B0 is not supported. Mode of use "N" with B, C, and D key block version IDs and Mode of use "B" for K0 export is not supported.

Table 429. TR-31 Translate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express6 CCA Coprocessor	Keywords SKEY-AES, SKEY-DES, VARDRV-D, TYPATO11, TYPBTO10, and DMP0TO12 and triple-length DES keys require the December 2018 or later licensed internal code (LIC). Keywords INITVEC, IV-DES, IV-TDES, and IV-AES require the July 2019 or later licensed internal code. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). The combination of keywords PINENC and ENCDEC requires a crypto express coprocessor with the June 2020 or later licensed internal code (LIC). Keywords HMAC, SKEYHMAC, ISOSHA-1, ISOSHA-2, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512, and TR-31 key block containing an HMAC key require the June 2020 or later licensed internal code. Export of AES DKYGENKY DUKPT BDK to key block B0 requires the October 2020 or later licensed internal code (LIC). Mode of use "N" with B, C, and D key block version IDs and Mode of use "B" for K0 export is not supported. Keywords COMP-CHK, COMP-TAG, STOREXCH, INTERNAL, and EXCHANGE are not supported. Keywords DUKPT, KDK, ENCSENS, EMVAC-F, EMVSC-F, EMVSI-F, EMVDA-F, and EMVDN-F are not supported. Keywords XPRTCPAC, NOEXCPAC, and DKPINOP are not supported. Keywords OB-DA, OB-LB, OB-IBM, OB-KC, OB-KP, OB-TC, OB-TS, OB-WP are not supported. Keyword ISOMAC6 is not supported with a DES key. Keyword KEK is not supported with a SECMSG key. Keyword PINENC is not supported with a SECMSG key. Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.

Table 429. TR-31 Translate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Keywords INITVEC, IV-DES, IV-TDES, and IV-AES are not supported. Compliant-tagged key tokens are not supported. The combination of keywords PINENC and ENCDEC requires a crypto express coprocessor with the June 2020 or later licensed internal code (LIC). Keywords HMAC, SKEYHMAC, ISOSHA-1, ISOSHA-2, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512, and TR-31 key block containing an HMAC key require the June 2020 or later licensed internal code. Export of AES DKYGENKY DUKPT BDK to key block B0 is not supported. Mode of use "N" with B, C, and D key block version IDs and Mode of use "B" for K0 export require CCA release (5.7, 6.7, 7.4, or 8.0) or later licensed internal code (LIC). Keywords COMP-CHK, COMP-TAG, STOREXCH, INTERNAL, and EXCHANGE are not supported. Keywords DUKPT, KDK, ENCSENS, EMVAC-F, EMVSC-F, EMVSI-F, EMVDA-F, and EMVDN-F are not supported. Keywords XPRTCPAC, NOEXCPAC, and DKPINOP are not supported. Keywords OB-DA, OB-LB, OB-IBM, OB-KC, OB-KP, OB-TC, OB-TS, OB-WP are not supported. Keyword ISOMAC6 is not supported with a DES key. Keyword KEK is not supported with a SECMSG key. Keyword PINENC is not supported with a SECMSG key. Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.

Table 429. TR-31 Translate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	The combination of keywords PINENC and ENCDEC requires a crypto express coprocessor with the June 2020 or later licensed internal code (LIC). Keywords HMAC, SKEYHMAC, ISOSHA-1, ISOSHA-2, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512, and TR-31 key block containing an HMAC key require the June 2020 or later licensed internal code. Export of AES DKYGENKY DUKPT BDK to key block B0 requires the September 2020 or later licensed internal code (LIC). Mode of use "N" with B, C, and D key block version IDs and Mode of use "B" for K0 export require CCA release (5.7, 6.7, 7.4, or 8.0) or later licensed internal code (LIC). Keywords COMP-CHK, COMP-TAG, STOREXCH, INTERNAL, and EXCHANGE are not supported. Keywords DUKPT, KDK, ENCSENS, EMVAC-F, EMVSC-F, EMVSI-F, EMVDA-F, and EMVDN-F are not supported. Keywords XPRTCPAC, NOEXCPAC, and DKPINOP are not supported. Keywords OB-DA, OB-LB, OB-IBM, OB-KC, OB-KP, OB-TC, OB-TS, OB-WP are not supported. Keyword ISOMAC6 is not supported with a DES key. Keyword KEK is not supported with a SECMSG key. Keyword PINENC is not supported with a SECMSG key. Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.

Table 429. TR-31 Translate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	Mode of use "N" with B, C, and D key block version IDs and Mode of use "B" for K0 export require CCA release (5.7, 6.7, 7.4, or 8.0) or later licensed internal code (LIC). Keywords COMP-CHK, COMP-TAG, STOREXCH, INTERNAL, and EXCHANGE are not supported. Keywords DUKPT, KDK, ENCSENS, EMVAC-F, EMVSC-F, EMVSI-F, EMVDA-F, and EMVDN-F are not supported. Keywords XPRTCPAC, NOEXCPAC, and DKPINOP are not supported. Keywords OB-DA, OB-LB, OB-IBM, OB-KC, OB-KP, OB-TC, OB-TS, OB-WP are not supported. Keyword ISOMAC6 is not supported with a DES key. Keyword KEK is not supported with a SECMSG key. Keyword PINENC is not supported with a SECMSG key. Operational X9.143 key blocks are not supported. External X9.143 key blocks with a key context field of ASCII '1' are not supported.
	Crypto Express8 CCA Coprocessor	Keywords COMP-CHK, COMP-TAG, STOREXCH, INTERNAL, and EXCHANGE require the CCA release 8.1 or later licensed internal code (LIC). Keywords DUKPT, KDK, ENCSENS, EMVAC-F, EMVSC-F, EMVSI-F, EMVDA-F, and EMVDN-F require the CCA release 8.1 or later licensed internal code (LIC). Keywords XPRTCPAC, NOEXCPAC, and DKPINOP require the CCA release 8.1 or later licensed internal code (LIC). Keywords OB-DA, OB-LB, OB-IBM, OB-KC, OB-KP, OB-TC, OB-TS, OB-WP require the CCA release 8.1 or later licensed internal code (LIC). Keyword ISOMAC6 with a DES key requires the CCA release 8.1 or later licensed internal code (LIC). Keyword KEK with a SECMSG key requires the CCA release 8.1 or later licensed internal code (LIC). Keyword PINENC with a SECMSG key requires the CCA release 8.1 or later licensed internal code (LIC). Support for operational X9.143 key blocks requires the CCA release 8.1 or later licensed internal code (LIC). Support for external X9.143 key blocks with a key context field of ASCII ‘1’ requires the CCA release 8.1 or later licensed internal code (LIC).

Chapter 11. TR-34 symmetric key management

This section provides information on services for the ANSI TR-34 protocol for key distribution.

The callable services that support the TR-34 key distribution protocol are:

- “TR-34 Bind-Begin (CSNDT34B and CSNFT34B)” on page 1072
- “TR-34 Bind-Complete (CSNDT34C and CSNFT34C)” on page 1079
- “TR-34 Key Distribution (CSNDT34D and CSNFT34D)” on page 1086
- “TR-34 Key Receive (CSNDT34R and CSNFT34R)” on page 1098

TR-34 protocol

The TR-34 protocol provides a method of securely distributing symmetric keys using asymmetric techniques. It is an implementation of the Unilateral Key Transport Method defined in ANSI X9.24-2.

TR-34 specifies an interoperable method for one situation – the Distribution of Symmetric Keys using Asymmetric Techniques from a Single Key Distribution Host (KDH) to many Key Receiving Devices (KRDs). The KDH is assumed to be operated in a controlled environment. The KRDs may operate in uncontrolled environments.

TR-34 is a Technical Report. This is different from a standard, which is a mandatory set of rules that must be followed. A Technical Report is not mandatory, but provides guidance to those who are using the standards. In this case, TR-34 is a companion to the ANSI X9.24-2 standard, which defines requirements for key management performed using asymmetric key techniques.

User flows

This section explains user actions to accomplish various tasks and how the TR-34 services are used in context of the protocol. There are two scenarios:

1. Peer-to-peer exchange where two parties use the protocol to establish a common transport key to exchange symmetric keys securely. In the peer-to-peer scenario, one party is the KDH (Key Distribution Host) and the other party is the KRD (Key Receiving Device).
2. One KDH (Key Distribution Host) to many KRDs (Key Receiving Device) where the host uses the protocol to establish a transport key for distributing symmetric keys with each of the receiving devices in service.

Parameters used in the protocol

The following are descriptions of some of the common parameters in the TR-34 protocol. They are credentials, certificates, TR-34 tokens, and CCA key tokens.

CSR-KDH
PKCS #10 certificate signing request for KDH.

CSR-KRD
PKCS #10 certificate signing request for KRD.

CredKDH
KDH credential (X.509 certificate) with ID and public key.

CredKRD
KRD credential (X.509 certificate) needed for key distribution.

CredCA
Certificate Authority credential.

CRL-CA
Certificate Revocation List from CA.

CT-KDH
KDH credential token, KDH bind token.

RBT-KDH
KDH rebind token.

UBT-KDH
KDH unbind token.

CT-KRD
KRD credential token, containing CredKRD.

RT-KRD
Random number token, generated by KRD.

KT-KDH
KDH key token – 1 or 2 pass.

D-kdh-T
CCA RSA private key token for KDH, contains the private key (D-kdh) and public key (E-kdh).

D-krd-T
CCA RSA private key token for KRD, contains the private key (D-krd) and public key (E-krd).

Kn
Key to be exported using TR-34.

Kn-T
CCA key token.

Note: These scenarios assume that the KDH and KRD are using IBM CCA services and servers for the TR-34 setup and applications. When CCA services are not being used, other appropriate cryptographic services should be substituted and used. The same situation applies to hardware and key storage.

Setup

These are the setup steps at each party: the Key Receiving Device (KRD) and the Key Distribution Host (KDH).

On the KDH (Key Distribution Host)

1. Create the KDH administrative RSA key pair for TR-34 use with target (or set of targets):
   - **Overview:** Generate RSA private key token (D-kdh-T).
   - Use CCA service: CSNDPKG

     INPUT:
     Key strength and type (RSA 2048, SIGN-ONLY).

     OUTPUT:
     The private key (D-kdh-T) token contains the private key (D-kdh) and the public key (E-kdh).

     KEY STORAGE:
     Store the output private key token (D-kdh-T) in the PKDS.
2. Create PKCS #10 certificate signing request for the public key of the administrative key pair:
   - **Overview:** Extract public key E-kdh into PKCS #10 certificate signing request (CSR-KDH).
   - Use CCA service: CSNDPIC

     INPUT:
     Private key token (D-kdh-T).

     OUTPUT:
     PKCS #10 certificate signing request (CSR-KDH).

KEY STORAGE:
Store the signing request (CSR-KDH) in the application space or key ring until the certificate can be generated.

Note: The signing request (CSR-KDH) is an intermediate object, but might be useful to have stored if the same private key is to be used with multiple ATM vendors or as a backup. It is recommended to have a unique private key for each group of receiving devices.

3. Create the KDH credential (CredKDH) from the certificate signing request (CSR-KDH):

- **Overview:** KDH sends the CSR-KDH to a Certificate Authority (CA) trusted by all parties to be turned into a certificate CredKDH under the agreed CA.
  - Application step or manual step such as loading the (CSR-KDH) to a web portal.
- KDH receives from the CA the following TR-34 objects:

CredKDH
Certificate holding (E-kdh). Store certificate for use with all devices during BIND.

KEY STORAGE
Store the (CredKDH) in application space or key ring.

CRL-CA
Current Certificate Revocation List. Store for use until not considered fresh any longer.

KEY STORAGE
Store the (CRL-CA) in application space or key ring.

CredCA
A certificate for CA. Install this in the CCA HSM PKI manually from the TKE workstation for each domain that will perform the TR-34 protocol. Also keep this available for new HSM provisioning.

KEY STORAGE
Store the (CredCA) in application space or key ring.

4. At the TKE workstation: Install the CA credential (CredCA) to the HSM domains that will perform TR-34 operations.

- **Overview:** The KDH administrator saves the CredCA in the internal PKI of the appropriate domains of the HSM as a trust root, using the TKE workstation.

On the KRD (Key Receiving Device)

1. Create the RSA key pair for TR-34 key management use. Each KRD will have an RSA key pair.

- **Overview:** Generate RSA private key token (D-kdh-T).
  - For peer-to-peer exchange, use CCA service: CSNDPKG.

INPUT:
Key strength and type (RSA 2048, KEY-MGT).

OUTPUT:
The private key (D-krd-T) token contains the private key (D-krd) and the public key (E-krd).

KEY STORAGE:
Store the output private key token (D-krd-T) in the PKDS.

  - For one host to many devices, typically the RSA key is generated and installed in the device at the factory.
    - The private key (D-krd) is installed on the device.

2. Create PKCS #10 certificate signing request for the public key E-krd:

- **Overview:** Extract public key E-kdh into PKCS #10 certificate signing request (CSR-KRD).
  - For peer-to-peer exchange, use CCA service: CSNDPIC.

INPUT:
Private key token (D-krd-T).

OUTPUT:
PKCS #10 certificate signing request (CSR-KRD).

KEY STORAGE:
Store the signing request (CSR-KRD) in application space or key ring until the certificate can be generated.

– For one host to many devices, generate the CSR-KRD for each device with the devices public key E-krd.

3. Create the KRD credential (CredKRD) from the certificate signing request (CSR-KRD):

- **Overview:** KRD sends the CSR-KRD to a Certificate Authority (CA) trusted by all parties to be turned into a certificate CredKRD under the agreed CA.
  – Application step or manual step such as loading the (CSR-KRD) to a web portal.
- KRD receives from the CA the following TR-34 objects:

CredKRD
Certificate holding (E-kdh). Store certificate for use to BIND with the KDH.

KEY STORAGE

– Peer-to-peer exchange: Store the (CredKRD) in application space or key ring.
– One host to many devices: Typically, installed on device at the factory.

CRL-CA
Current Certificate Revocation List. Store for use until not considered fresh any longer.

KEY STORAGE
Store the (CRL-CA) in application space or key ring.

CredCA
A certificate for CA. Install this in the KRD.

KEY STORAGE

– Peer-to-peer exchange: Store the (CredKRD) in application space or key ring.
– One host to many devices: Typically, installed on device at the factory.

4. Install the KRD credential holding the KRD public key (CredKRD):

- For one host to many devices: This is typically installed on device at the factory.
- For peer-to-peer exchange: The CredKRD might be stored in application space for the TR-34 implementation.

KEY STORAGE

– Store the (CredCA, CredKRD) in application space or key ring.
– Store the KRD private key token (D-krd-T) in the PKDS.

5. Install the CA credential (CredCA) (primary, secondary, 'higher' primary/secondary, and so on):

- Peer-to-peer exchange: The CredCA is installed in the CCA HSM PKI manually from the TKE workstation, for each domain that will use the TR-34 protocol. Also, keep the CredCA available for new HSM provisioning.
- One host to many devices: This is typically installed on the device at the factory.

BIND

Bind a Key Receiving Device (KRD) to a Key Distribution Host (KDH). These are the steps, in sequence, with the CCA service APIs identified:

1. **On KDH**

- The KDH TR-34 application requests the CredKRD from the KRD.

2. **On KRD**
   - The CredKRD request is received and processed by the TR-34 application.
   - Create the token that contains the CredKRD.
     a. **Overview**: Call CCA service CSNDT34C: "BINDKRDC" to create the TR-34 token that contains CredKRD for the KDH.
     b. **INPUT**:
     – CredKRD: KRD credential with ID and public key.
     - **KEY STORAGE**: Stored in application space or key ring.
     c. **OUTPUT**:
     – CT-KRD: Credential token for KRD, containing CredKRD.
     - **KEY STORAGE**: Stored in application space until sending. This is an opaque blob useable only in this protocol step.
   - TR-34 application sends CT-KRD to KDH.
3. **On KRH**
   - Refresh CRL-CA if needed:
     – If CRL-CA held by the KDH, representing the CA shared between the KRD and KDH, is not fresh any longer, the KDH should obtain a new CRL-CA before doing the next step.
   - Create the 'BIND' token needed for the next protocol step:
     – **Overview**: Call CCA service CSNDT34B: "BINDCR"
     – **INPUT**:
     - CT-KRD: Credential token received from KRD, containing CredKRD.
     • **KEY STORAGE**: Stored in application space until calling into the service.
     - CRL-CA: Certificate Revocation List from CA
     • **KEY STORAGE**: Stored in application space or key ring.
     - CredKDH: KDH credential with ID and public key.
     • **KEY STORAGE**: Stored in application space or key ring.
     – **OUTPUT**:
     a. CredKRD: Credential needed for future key distribution calls.
     - **KEY STORAGE**: Stored in application space or key ring.
     b. CT-KDH: BIND token.
     - **KEY STORAGE**: Stored in application space until send to KRD
   - Application stores CredKRD so that it is available for future key distribution calls.
   - KDH TR-34 application sends the CT-KDH token to the KRD.
4. **On KRD**
   - The KRD receives the CT-KDH token from the KDH and processes it to complete the BIND.
     a. **Overview**: Call CCA service CSNDT34C: "BINDRV".
     b. **INPUT**:
     – CT-KDH: Token BIND token received from KDH.
     - **KEY STORAGE**: Stored in application space until processed in the service call.
     c. **OUTPUT**:

- CredKDH: Credential for the KDH, needs to be stored in the KRD.
  - **KEY STORAGE**: Stored in application space or key ring.

- The application on the KRD stores the CredKDH to complete the 'Bind' phase.

UNBIND

Unbind a Key Receiving Device (KRD) from a Key Distribution Host (KDH). This is done before the KRD is taken out of service or if the KDH otherwise needs to remove ownership/bind/keys from a KRD. These are the steps, in sequence, with the CCA service APIs identified:

1. **On KDH**
   - The KDH requests a random number from the KRD.
2. **On KRD**
   - Call CCA service CSNBRNGL: "RT-KRD" to create the TR-34 token that contains the random number that is needed by the KDH.
     a. **INPUT**:
        - RN-Length: Length of the random number needed.

     b. **OUTPUT**:
        - RT-KRD: Random number token.
   - Send RT-KRD to KDH and also store RT-KRD locally in application space for the later validation step.
3. **On KDH**
   a. If CRL-CA held by the KDH, representing the CA shared between the KRD and KDH, is not fresh any longer, the KDH should obtain a new CRL-CA before doing the next step.
   b. Call CCA service CSNDT34B: "UNBINDCR"
      - **INPUT**:
        i) RT-KRD: Random number token received from KRD.
        ii) CRL-CA: Certificate Revocation List from CA.
        iii) CredKRD: KRD credential with ID and public key.
        iv) CredKDH: KDH credential with ID and public key.
        v) D-kdh: Private key to sign data block.
      - **OUTPUT**:
        - UBT-KDH: UNBIND token.

   c. KDH sends the UBT-KDH token to the KRD.
4. **On KRD**
   a. The KRD receives the UBT-KDH token from the KDH and now must process it to complete the UNBIND.
   b. Call CCA service CSNDT34C: "UNBINDRV"
      - **INPUT**:
        i) UBT-KDH token: UNBIND token received from KDH.
        ii) CredKRD: KRD credential with ID and public key.
        iii) CredKDH: KDH credential with ID and public key.
        iv) RT-KRD: Token originally sent by the KRD to the KDH, used now for validation.
      - **OUTPUT**:
        - UBT-KDH – is – valid: (yes or no/error).

c. The application or ICSF on the KRD removes all the keys associated with the KDH that sent the UNBIND request, which completes the 'UNBIND' phase.

REBIND

Rebind a Key Receiving Device (KRD) to a Key Distribution Host (KDH). This is done when a KDH needs to update the credentials held at the KRD that represent the KDH; for example, if the certificate is about to expire. These are the steps, in sequence, with the CCA service APIs identified:

1. **On KDH**
   a. Generate a new RSA key pair (E-kdh-new, D-kdh-new), made by the HSM in their mainframe.
      - Use CCA service: CSNDPKG
   b. Extract public key (E-kdh-new) into PKCS #10 certificate signing request (CSR-KDH-new).
      - Use CCA service: CSNDPIC
   c. KDH sends CSR-KDH-new to the Certificate Authority to be turned into a certificate CredKDH-new under the agreed CA.
   d. KDH receives from CA these TR-34 objects.
      i) CredKDH-new: A certificate holding (E-kdh-new). Store certificate for use with all KRDs during REBIND.
      ii) CRL-CA: New Certificate Revocation List. Store for use until not considered fresh any longer.
      iii) CredCA: A certificate for CA. Install this in the CCA HSM PKI manually from the TKE for each domain that will use the TR-34 protocol. Also, keep this for new card provisioning.
   e. The KDH requests a random number from the KRD.
2. **On KRD**
   a. Call CCA service CSNBRNGL: "RT-KRD" to create the TR-34 token that contains the random number that is needed by the KDH.
      i) **INPUT**:
         - RN-length: Length of the random number needed.
      ii) **OUTPUT**:
         - RT-KRD: Random number token.
      iii) **KEY STORAGE**:
         - RT-KRD stored in application space.
   b. Send RT-KRD to KDH.
3. **On KDH**
   a. If CRL-CA is not fresh any longer, the KDH should obtain a new CRL-CA before doing the next step.
   b. Call CCA service CSNDT34B: "REBINDCR".
      i) **INPUT**:
         a) RT-KRD: Random number token received from KRD.
         b) CRL-CA: Certificate Revocation List from CA.
         c) CredKRD: KRD credential with ID and public key.
         d) CredKDH-new: New KDH credential with ID and public key.
         e) CredKDH-old: Old KDH credential with ID and public key.
         f) D-kdh: Old private key needed to sign the REBIND data block.
      ii) **OUTPUT**:
         - UBT-KDH: REBIND token.

c. KDH sends the UBT-KDH token to the KRD.

4. **On KRD**

a. The KRD receives the UBT-KDH token from the KDH and now must process it to complete the REBIND.

b. Call CCA service CSNDT34C: "REBINDRV"

i) **INPUT**:

a) UBT-KDH: REBIND token received from KDH.

b) CredKRD: KRD credential with ID and public key.

c) CredKDH: Old KDH credential with ID and public key.

d) RT-KRD. Token originally sent by the KRD to the KDH and used now for validation.

ii) **OUTPUT**:

a) UBT-KDH – is – valid: (yes or no/error)

b) CredKDH-new: New KRD credential.

iii) **KEY STORAGE**:

- CredKDH-new stored in application space.

c. The application or ICSF on the KRD removes all the keys associated with the CredKDH that sent the REBIND request (they are regarded as invalid). The KRD stores the CredKDH-new so that future key distribution events can be handled. This completes the 'REBIND' phase.

2-pass key transport

This key transport typically installs a Terminal Master Key (TMK) at the Key Receiving Device. Another name for the key is the TR-31 wrapping key name: Key Block Protection Key (KBPK). These represent the same key, a key-encrypting key (KEK), that is shared by the KDH and KRD so that more keys can be shared without going through the entire TR-34 protocol again. However, the type of key transported can be any type of key supported by the TR-31 key block that is carried in the TR-34 protocol token. For this flow, the assumption is that a typical KEK is being exchanged.

'2-pass' indicates that the KRD is expected to be online for the initial part of the protocol where a random number token is requested. If the KRD is expected to be offline, the '1-pass' protocol may be more appropriate.

Setup steps for key transport

On KDH

1. Refresh CRL-CA if needed:

- If CRL-CA held by the KDH, representing that the CA shared between the KRD and KDH is not fresh any longer, the KDH should obtain a new CRL-CA before doing the next step.

2. Create TMK/KBPK: The KEK to be shared with the other party.

- CCA has restrictions on the type of key that can be used to wrap another key for TR-31 Translate (the CSNBT31X *wrap_kek_identifier*). Therefore, one of these key types should be generated for exchange:

  DES:

  a. KDH-Wrapping-KEK:

  i) This is an known or random value EXPORTER KEK created using the TKE workstation or CSNBKPI.

  ii) This key is used with CSNBKGN to create the KRD-Import-KEK, which must be generated in external form under a KEK according to CSNBKGN processing restrictions.

  iii) **KEY STORAGE**: Stored in CKDS.

b. KDH-Export-KEK:

i) This is the KDH copy of the TMK key that will be used later to wrap keys being exported to the ATM.

ii) Type: DES EXPORTER key for wrapping output TR-31 key blocks.

iii) **KEY STORAGE**: Stored in CKDS.

c. KRD-Import-KEK:

i) This is the KRD copy of the TMK key that will be used later by the ATM to unwrap TR-31 key blocks for key distribution completion.

ii) Type: DES IMPORTER or IKEYXLAT key for unwrapping input TR-31 key blocks.

iii) This is the key (Kn) to be exported using TR-34. Because the key is in a CCA token, the notation (Kn-T) is used.

iv) **KEY STORAGE**: Stored in CKDS until export.

d. CCA services:

i) Use the CSNBKTB service to create skeleton tokens for the EXPORTER/IMPORTER copies of the key.

ii) Use the CSNBKGN service to generate a new DES key and populate to each token.

AES:

a. KDH-Wrapping-KEK:

i) This is an known or random value EXPORTER KEK created using the TKE workstation or CSNBKPI2.

ii) This key is used with CSNBKGN2 to create the KRD Import KEK, which must be generated in external form under a KEK according to CSNBKGN2 processing restrictions.

iii) **KEY STORAGE**: Stored in CKDS.

b. KDH-Export-KEK:

i) This is the KDH copy of the TMK key that will be used later to wrap keys being exported to the ATM.

ii) Type: AES EXPORTER key with usage EXPTT31D for wrapping output TR-31 key blocks.

iii) **KEY STORAGE**: Stored in CKDS.

c. KRD-Import-KEK:

i) This is the KRD copy of the TMK key that will be used later by the ATM to unwrap TR-31 key blocks for key distribution completion.

ii) Type: AES IMPORTER key with usage IMPTT31D for unwrapping input TR-31 key blocks.

iii) This is the key (Kn) to be exported using TR-34. Since the key is in a CCA token, the notation (Kn-T) is used.

iv) **KEY STORAGE**: Stored in CKDS until export.

d. CCA services:

– Use the CSNBKTB2 service to create skeleton tokens for the EXPORTER/IMPORTER copies of the key.

– Use the CSNBKGN2 service to generate a new AES key and populate each token. Note that the key strength must be 128 bit if key strength rules are to be followed for PCI PTS HSM compliance and other standards. The maximum TR-34 wrapping key strength is AES 128-bit.

Protocol steps for key transport

1. **On KDH**

a. The KDH TR-34 application requests a random number from the KRD.

2. **On KRD**

   a. KRD TR-34 application receives the random number request and processes it.

      i) **Overview:** Call CCA service CSNBRNGL: "RT-KRD" to create the TR-34 token that contains random number that is needed by the KDH.

      ii) **INPUT**:

      - RN-length: Length of the random number needed.

      iii) **OUTPUT**:

      - RT-KRD: Random number token.
        - **KEY STORAGE**: Stored in application space or key ring until the TR-34 Key Transport token (KT-KDH) is received that matches this RT-KRD random number.

   b. Send RT-KRD to KDH. Also, store RT-KRD locally in application space for a later validation step.

3. **On KDH**

   a. Create the Key Transport token:

      i) **Overview:** Call CCA service CSNDT34D: "2PASSCRE".

      ii) **INPUT**:

      a) Kn-T: CCA key token to be exported to the KRD.

      - **KEY STORAGE**: Stored in CKDS.

      b) RT-KRD: Random number token received from KRD.

      - **KEY STORAGE**: Stored in application space until this call happens.

      c) CRL-CA: Certificate Revocation List from CA.

      - **KEY STORAGE**: Stored in application space or key ring.

      d) CredKRD: KRD credential with ID and public key. The public key from this is used to RSA-encipher the RSA-encrypted part of the key block.

      - **KEY STORAGE**: Stored in application space or key ring.

      e) CredKDH: KDH credential with ID and public key.

      - **KEY STORAGE**: Stored in application space or key ring.

      f) D-kdh: Private key to sign data block.

      - **KEY STORAGE**: Stored in PKDS.

      g) **KBH inputs:** TR-31 KBH for a terminal master key is not actually variable, but a peer-to-peer exchange will have more inputs to describe keys.

      iii) **OUTPUT**:

      - KT-KDH: Key transport (2-pass) token.
        - **KEY STORAGE**: Stored in application space until sent to KRD.

   b. KDH sends the KT-KDH token to the KRD.

4. **On KRD**

   a. The KRD receives the KT-KDH token from the KDH and must process it to complete the Key Transport.

   b. Create key token to hold output TMK (CCA / peer-to-peer step).

   - Use CCA service CSNBKTB or CSNBKTB2 to create Kn-TS, a skeleton CCA key token appropriate for a importing the TMK/KBPK.

   c. Process the KT-KDH token received from the KDH.

i) **Overview:** Call CCA service CSNDT34R: "2PASSRCV".

ii) **INPUT**:

a) KT-KDH: Key transport (2-pass) token received from KDH.

- **KEY STORAGE**: Stored in application space until processed.

b) CredKDH: KDH credential with ID and public key.

- **KEY STORAGE**: Stored in application space or key ring.

c) RT-KRD: Token originally sent by the KRD to the KDH and used now for validation.

- **KEY STORAGE**: Stored in application space.

d) D-krd: The private key matching the public key in CredKRD. D-krd is stored in the KRD (but not in the HSM). Used to RSA-decipher part of the KT-KDH key block.

- **KEY STORAGE**: Stored in PKDS for peer-to-peer / CCA TR-34.

iii) **OUTPUT**:

- Kn-T: CCA key token containing the transported TMK/KBPK.
  - **KEY STORAGE**: Stored in CKDS on KRD.

5. **On KRD**

a. Key Check Value generated and returned to KDH.

- The KRD generates a Key Check Value (KCV) for Kn-T using CCA service CSNBKYT2: "GENERATE" and "ENC-ZERO" or "CMACZERO" service depending on the key algorithm. KRD sends this to the KDH. There is no special encoding for the KCV.

6. **On KDH**

a. Key Check Value verified by KDH:

- KDH verifies the KCV against the original Kn-T that was sent for export using TR-34. The CCA service CSNBKYT2: "VERIFY" and "ENC-ZERO" or "CMACZERO" service depending on the key algorithm can be used for this value.

1-pass key transport

This key transport typically installs a Terminal Master Key (TMK) at the key receiving device. Another name for the key is the TR-31 wrapping key name: Key Block Protection Key (KBPK). These represent the same key, a key-encrypting key (KEK), that is shared by the KDH and KRD so that more keys can be shared without going through the entire TR-34 protocol again. However, the type of key transported can be any type of key supported by the TR-31 key block that is carried in the TR-34 protocol token. For this flow, the assumption is that a typical KEK is being exchanged.

'1-pass' indicates that the KRD is not expected to be online. The protocol includes a timestamp (or other monotonic increasing number) to be sent by the KDH so that the KRD can defend against replay attacks

Setup steps for key transport

On KDH

1. Refresh CRL-CA if needed:

- If CRL-CA held by the KDH, representing that the CA shared between the KRD and KDH is not fresh any longer, the KDH should obtain a new CRL-CA before doing the next step.

2. Create TMK/KBPK: The KEK to be shared with the other party.

- CCA has restrictions on the type of key that can be used to wrap another key for TR-31 Translate (the CSNBT31X *wrap_kek_identifier*). Therefore, one of these key types should be generated for exchange:

DES:

a. KDH-Wrapping-KEK:

i) This is an known or random value EXPORTER KEK created using the TKE workstation (required for comp-tag keys) or CSNBKPI.

ii) This key is used with CSNBKGN to create the KRD-Import-KEK, which must be generated in external form under a KEK according to CSNBKGN processing restrictions.

iii) **KEY STORAGE**: Stored in CKDS.

b. KDH-Export-KEK:

i) This is the KDH copy of the TMK key that will be used later to wrap keys being exported to the ATM.

ii) Type: DES EXPORTER key for wrapping output TR-31 key blocks.

iii) **KEY STORAGE**: Stored in CKDS.

c. KRD-Import-KEK:

i) This is the KRD copy of the TMK key that will be used later by the ATM to unwrap TR-31 key blocks for key distribution completion.

ii) Type: DES IMPORTER or IKEYXLAT key for unwrapping input TR-31 key blocks.

iii) This is the key (Kn) to be exported using TR-34. Because the key is in a CCA token, the notation (Kn-T) is used.

iv) **KEY STORAGE**: Stored in CKDS until export.

d. CCA services:

i) Use the CSNBKTB service to create skeleton tokens for the EXPORTER/IMPORTER copies of the key.

ii) Use the CSNBKGN service to generate a new DES key and populate to each token.

AES:

a. KDH-Wrapping-KEK:

i) This is an known or random value EXPORTER KEK created using the TKE workstation (required for comp-tag keys) or CSNBKPI2.

ii) This key is used with CSNBKGN2 to create the KRD Import KEK, which must be generated in external form under a KEK according to CSNBKGN2 processing restrictions.

iii) **KEY STORAGE**: Stored in CKDS.

b. KDH-Export-KEK:

i) This is the KDH copy of the TMK key that will be used later to wrap keys being exported to the ATM.

ii) Type: AES EXPORTER key with usage EXPTT31D for wrapping output TR-31 key blocks.

iii) **KEY STORAGE**: Stored in CKDS.

c. KRD-Import-KEK:

i) This is the KRD copy of the TMK key that will be used later by the ATM to unwrap TR-31 key blocks for key distribution completion.

ii) Type: AES IMPORTER key with usage IMPTT31D for unwrapping input TR-31 key blocks.

iii) This is the key (Kn) to be exported using TR-34. Since the key is in a CCA token, the notation (Kn-T) is used.

iv) **KEY STORAGE**: Stored in CKDS until export.

d. CCA services:

– Use the CSNBKTB2 service to create skeleton tokens for the EXPORTER/IMPORTER copies of the key.

- Use the CSNBKGN2 service to generate a new AES key and populate each token. Note that the key strength must be 128 bit if key strength rules are to be followed for PCI PTS HSM compliance and other standards. The maximum TR-34 wrapping key strength is AES 128-bit.

Protocol steps for key transport

1. **On KDH**

   a. TR-34 application creates a Timestamp. This is the freshness test that will be used by the KRD.

   b. Create the Key Transport token:

      i) **Overview:** Call CCA service CSNDT34D: "1PASSCRE".

      ii) **INPUT**:

         a) Kn-T: CCA key token to be exported to the KRD.

            • **KEY STORAGE**: Stored in CKDS.

         b) Timestamp freshness indicator to defend against replay attacks.

         c) CRL-CA: Certificate Revocation List from CA.

            • **KEY STORAGE**: Stored in application space or key ring.

         d) CredKRD: KRD credential with ID and public key. The public key from this is used to RSA-encipher the RSA-encrypted part of the key block.

            • **KEY STORAGE**: Stored in application space or key ring.

         e) CredKDH: KDH credential with ID and public key.

            • **KEY STORAGE**: Stored in application space or key ring.

         f) D-kdh: Private key to sign data block.

            • **KEY STORAGE**: Stored in PKDS.

         g) **KBH inputs:** TR-31 KBH for a terminal master key is not actually variable, but a peer-to-peer exchange will have more inputs to describe keys.

      iii) **OUTPUT**:

         • KT-KDH: Key transport (1-pass) token.

            – **KEY STORAGE**: Stored in application space until sent to KRD.

   c. KDH TR-34 application sends the KT-KDH token to the KRD.

2. **On KRD**

   a. The KRD receives the KT-KDH token from the KDH and must process it to complete the Key Transport.

   b. Create key token to hold output TMK (CCA / peer-to-peer step).

      • Use CCA service CSNBKTB or CSNBKTB2 to create Kn-TS, a skeleton CCA key token appropriate for a importing the TMK/KBPK.

   c. Process the KT-KDH token received from the KDH.

      i) **Overview:** Call CCA service CSNDT34R: "1PASSRCV".

      ii) **INPUT**:

         a) KT-KDH: Key transport (1-pass) token received from KDH.

            • **KEY STORAGE**: Stored in application space until this call occurs.

         b) CredKDH: KDH credential with ID and public key.

            • **KEY STORAGE**: Stored in application space or key ring.

c) Timestamp-min: Minimum timestamp value allowed. It could be the last timestamp received or a bootstrap value. If passed, card validates that the KT-KDH contains a newer timestamp than Timestamp-min.

d) D-krd: The private key matching the public key in CredKRD. D-krd is stored in the KRD (but not in the HSM). Used to RSA-decipher part of the KT-KDH key block.

- **KEY STORAGE**: Stored in PKDS for peer-to-peer / CCA TR-34.

iii) **OUTPUT**:

- Kn-T: CCA key token containing the transported TMK/KBPK.
  - **KEY STORAGE**: Stored in CKDS on KRD.
- Timestamp-rcv: Timestamp value received from the KDH in the KT-KDH.
  - **STORAGE**: Stored in application space as the next minimum.

d. If necessary, the application validates Timestamp-rcv in a customized fashion.

3. **On KRD**

a. Key Check Value generated and returned to KDH.

- The KRD generates a Key Check Value (KCV) for Kn-T using CCA service CSNBKYT2: "GENERATE" and "ENC-ZERO" or "CMACZERO" service depending on the key algorithm. KRD sends this to the KDH. There is no special encoding for the KCV.

4. **On KDH**

a. Key Check Value verified by KDH:

- KDH verifies the KCV against the original Kn-T that was sent for export using TR-34. The CCA service CSNBKYT2: "VERIFY" and "ENC-ZERO" or "CMACZERO" service depending on the key algorithm can be used for this value.

TR-34 Bind-Begin (CSNDT34B and CSNFT34B)

The TR-34 Bind-Begin service is used for operations that take place at the Key Distribution Host (KDH) during TR-34 Protocol Bind related operations, as specified in X9 TR34-2012.

The TR-34 protocol describes the data objects and cryptograms needed for securely provisioning a symmetric key from a Key Distribution Host (KDH) to a Key Receiving Device (KRD). Although designed for 1-way key distribution, such as between a host computer (KDH) and an ATM (KRD), the protocol can be used for peer-to-peer key distribution as well. The main requirement for either use of the protocol is a Certificate Authority (CA) trusted by both the KDH and the KRD.

The callable service name for AMODE(64) invocation is CSNFT34B.

The TR-34 Bind-Begin service is used to perform these operations:

- BINDCR: The TR34 BIND token (CT-KDH) CREATE service.
  - CT-KRD: (INPUT, *input_token*). Credential token received from KRD, containing Cred-KRD.
  - CRL-CA: (INPUT, *crl*). Certificate Revocation List from CA.
  - CredKDH: (INPUT, *cred_kdh*). KDH credential (X.509 certificate) with ID and public key.
  - CredKRD: (OUTPUT, *cred_krd*). (X.509 certificate) needed for future key distribution calls.
  - CT-KDH: (OUTPUT, *output_token*). BIND token in DER format.
- UNBINDCR: The TR34 UNBIND token (UBT-KDH) CREATE service.
  - RT-KRD: (INPUT, *input_token*). Random number token received from KRD.
  - CRL-CA: (INPUT, *crl*). Certificate Revocation List from CA.
  - CredKDH: (INPUT, *cred_kdh*). KDH credential (X.509 certificate) with ID and public key.
  - CredKRD: (INPUT, *cred_krd*). KRD credential (X.509 certificate) with ID and public key.

  - D-kdh: (INPUT, *private_key_identifier*). Private key to sign data block.
  - UBT-KDH: (OUTPUT, *output_token*) UNBIND token in DER format.
- REBINDCR: The TR34 REBIND token (RBTKDH) CREATE service.
  - RT-KRD: (INPUT, *input_token*). Random number token received from KRD.
  - CRL-CA: (INPUT, *crl*). Certificate Revocation List from CA.
  - CredKDH-new: (INPUT, *cred_kdh*). New KDH credential (X.509 certificate) with ID and public key.
  - CredKDH-old: (INPUT, *old_cred_kdh*). Old KDH credential (X.509 certificate) with ID and public key.
  - CredKRD: (INPUT, *cred_krd*). KRD credential (X.509 certificate) with ID and public key.
  - D-kdh: (INPUT, *private_key_identifier*). Old private key, needed to sign the REBIND data block.
  - RBT-KDH: (OUTPUT, *output_token*) REBIND token in DER format.

Notes:

1. The RT-KRD token can be created with correct formatting using the RT-KRD processing of the CSNBRNGL service. See "Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)" on page 388 for more details.
2. RSA keys with 2048 bit and 3072 bit modulus are supported by CCA. Public exponent 65537 is currently the only supported exponent reflecting the support documented in ASC X9 TR-34-2019. This allows strength equivalent to an AES 128-bit key. TR-34 explicitly supports only RSA 2048-bit keys so some vendors will only support that key size.

Format

```
CALL CSNDT34B(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              input_token_length,
              input_token,
              crl_length,
              crl,
              cred_kdh_length,
              cred_kdh,
              old_cred_kdh_length,
              old_cred_kdh,
              cred_krd_length,
              cred_krd,
              private_key_identifier_length,
              private_key_identifier,
              output_token_length,
              output_token,
              reserved_length,
              reserved_data)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be between 1 and 4, inclusive.

rule_array

Direction	Type
Input	Character

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 430. Keywords for TR-34 Bind-Begin

Keyword	Meaning
Requested action (one, required).	
BINDCR	TR34 BIND token (CTKDH) CREATE service. Creates the token sent by the KDH to the KRD to accomplish the BIND action in the TR-34 protocol. This binds the KRD to the KDH for a later key distribution action.
UNBINDCR	TR34 UNBIND token (UBTKDH) CREATE service. Creates the token sent by the KDH to the KRD to accomplish the UNBIND action in the TR-34 protocol. This frees the KRD from the currently bound KDH and causes the KRD to remove all keys received while bound to this KDH.
REBINDCR	TR34 REBIND token (RBTKDH) CREATE service. Creates the token sent by the KDH to the KRD to accomplish the REBIND action in the TR-34 protocol. This frees the KRD from the current binding key of the KDH and binds the KRD to a new binding key from the KDH. This also causes the KRD to remove all keys received while bound to the KDH under the prior binding key.
Public Key Infrastructure Usage (one, optional).	

Table 430. Keywords for TR-34 Bind-Begin (continued)	
Keyword	**Meaning**
PKI-CHK	Specifies that the X.509 certificate for the other party (KRD) is to be validated against the trust chain of the PKI hosted in the adapter. This requires that the CA credentials have been installed using the Trusted Key Entry (TKE) workstation. This is required for compliance-tag key token export with TR-34 services. This is the default.
PKI-NONE	Specifies that the X.509 certificate for the other party (KRD) is not to be validated against the trust chain of the PKI hosted in the adapter. This is suitable if the certificate has been validated using host-based PKI services.
CRL expiration date checking	
CRLEXPCK	Fail the request if the certificate revocation list (CRL) in *crl* is expired. This is the default.
CRLEXPAL	Do not fail the request if the certificate revocation list (CRL) in *crl* is expired. Instead, return an informational *reason_code*.
KRD certificate date checking	
RCTEXPCK	Fail the request if the X.509 certificate in *cred_krd* is expired. This is the default.
RCTEXPAL	Do not fail the request if the X.509 certificate in *cred_krd* is expired. Instead, return an informational *reason_code*.

input_token_length

Direction	**Type**
Input	Integer

The length of the *input_token* parameter in bytes. The maximum length is 3500 bytes.

input_token

Direction	**Type**
Input	String

The DER encoded TR-34 token object. When the requested action keyword is BINDCR, the object is the TR-34 credential token from KRD (the CT-KRD). When the request action keyword is UNBINDCR or REBINDCR, the object is the TR-34 random number token from the KRD (the RT-KRD).

crl_length

Direction	**Type**
Input	Integer

The length of the *crl* parameter in bytes. The maximum length is 6000 bytes.

crl

Direction	**Type**
Input	String

The certificate revocation list (CRL) from the certificate authority the is in common with the KRD for the requested service. The CRL may be in DER or PEM format.

Note: The CSNDT34B service is acting as the KDH so the *crl* is not expected to validate against the internal PKI of the adapter.

cred_kdh_length

Direction	Type
Input	Integer

The length of the *cred_kdh* parameter in bytes. The maximum length is 3500 bytes.

cred_kdh

Direction	Type
Input	String

The X.509 certificate that is the credential of the KDH for the requested service. The certificate may be in DER or PEM format.

The meaning is determined by the requested action keyword:

BINDCR and UNBINDCR
This parameter must contain the X.509 certificate which is the TR-34 credential for the KDH (the CredKDH).

REBINDCR
This parameter must contain the new X.509 certificate which is the TR-34 credential for the KDH (the CredKDH-NEW).

Note: This service is acting as the KDH so the *cred_kdh* is not expected to validate against the internal PKI of the adapter.

old_cred_kdh_length

Direction	Type
Input	Integer

The length of the *old_cred_kdh* parameter in bytes. The maximum length is 3500 bytes. When the requested action keyword is BINDCR or UNBINDCR, the value must be 0.

old_cred_kdh

Direction	Type
Input	String

The X.509 certificate that is the credential of the KDH for the requested service. The certificate may be in DER or PEM format.

When the *old_cred_kdh_length* is zero, this parameter is ignored.

The meaning is determined by the requested action keyword:

REBINDCR
This parameter must contain the old X.509 certificate which is the TR-34 credential for the KDH (the CredKDH-OLD). The identifier and serial number are needed for the creation of the Rebind Token.

Note: This service is acting as the KDH so the *old_cred_kdh* is not expected to validate against the internal PKI of the adapter.

cred_krd_length

Direction	Type
Input/Output	Integer

The length of the *cred_krd* parameter in bytes. The maximum length is 3500 bytes.

cred_krd

Direction	Type
Input/Output	String

The X.509 certificate that is the credential of the KRD for the requested service (the CredKRD). The certificate may be in DER or PEM format.

The meaning is determined by the requested action keyword:

BINDCR
On input, this parameter must be an empty buffer of size *cred_krd_length*. On output, this parameter will contain the CredKRD extracted from the CT-KRD.

UNBINDCR, REBINDCR
This parameter must contain the CredKRD extracted by a previous BINDCR service.

Note: This service is acting as the KDH so the *cred_krd* is normally expected to validate against the internal PKI of the adapter. Use the PKI-NONE keyword to override this validation.

private_key_identifier_length

Direction	Type
Input	Integer

The length of the *private_key_identifier* parameter. When the requested action keyword is BINDCR, the value must be zero. When the keyword is UNBINDCR or REBINDCR, the value is the length of the key token or label. If the *private_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 3500.

private_key_identifier

Direction	Type
Input	String

The identifier of the private key used to sign the output token. The key identifier is an operational RSA secure token or the label of such a token in key storage. When the *private_key_identifier_length* is zero, this parameter is ignored.

The key usage of the token must allow digital signature. Retained private keys are not supported in this service.

output_token_length

Direction	Type
Input/Output	Integer

The length of the *output_token* parameter in bytes. The maximum length is 9000 bytes. On input, the value is the size of the buffer to receive the *output_token*. On output, the value is the actual size of the data returned in the *output_token* parameter.

output_token

Direction	Type
Output	String

The generated DER encoded TR-34 token.

BINDCR
This parameter will contain the TR-34 BIND token (CTKDH).

UNBINDCR
This parameter will contain the TR-34 UNBIND token (UBTKDH).

REBINDCR
This parameter will contain the TR-34 REBIND token (RBTKDH).

reserved_length

Direction	Type
Input/Output	Integer

This parameter is reserved. The value must be zero.

reserved

Direction	Type
Input/Output	String

This parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the PKDS.

Access control points

The **TR-34 Bind-Begin** access control in the domain role controls the function of this service.

The following table shows the access controls in the domain role that control the function of this service:

Table 431. Access control points for TR-34 Bind-Begin

Rule array keyword	Access control
BINDCR	TR-34 Bind-Begin - Allow BINDCR.
UNBINDCR	TR-34 Bind-Begin - Allow UNBINDCR.
REBINDCR	TR-34 Bind-Begin - Allow REBINDCR.
PKI-NONE	Permit X.509 without PKI root validation.
RCTEXPAL	TR-34 - Allow expired KRD certificate.
CRLEXPAL	TR-34 - Allow expired CRL.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 432. TR-34 Bind-Begin required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**		This service is not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	This service is not supported.
	Crypto Express6 CCA Coprocessor	This service requires the July 2019 or later licensed internal code. Rules CRLEXPCK, CRLEXPAL, RCTEXPCK, RCTEXPAL, *crl_length* greater than 3500 bytes, and *output_token_length* greater than 3500 bytes require CCA release 6.7.18 or higher.
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	This service is not supported.
	Crypto Express6 CCA Coprocessor	Rules CRLEXPCK, CRLEXPAL, RCTEXPCK, RCTEXPAL, *crl_length* greater than 3500 bytes, and *output_token_length* greater than 3500 bytes require CCA release 6.7.18 or higher.
	Crypto Express7 CCA Coprocessor	Rules CRLEXPCK, CRLEXPAL, RCTEXPCK, RCTEXPAL, *crl_length* greater than 3500 bytes, and *output_token_length* greater than 3500 bytes require CCA release 7.4.25 or higher.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

TR-34 Bind-Complete (CSNDT34C and CSNFT34C)

The TR-34 Protocol Bind-Complete service is used for operations that take place at the Key Receiving Device (KRD) during TR-34 Protocol Bind related operations, as specified in X9 TR34-2012.

The TR-34 protocol describes the data objects and cryptograms needed for securely provisioning a symmetric key from a Key Distribution Host (KDH) to a Key Receiving Device (KRD). Although designed for 1-way key distribution, such as between a host computer (KDH) and an ATM (KRD), the protocol can be used for peer-to-peer key distribution as well. The main requirement for either use of the protocol is a Certificate Authority (CA) trusted by both the KDH and the KRD.

The callable service name for AMODE(64) invocation is CSNFT34C.

This service is used to perform these operations:

- BINDKRDC: The TR34 BIND token (CT-KRD) CREATE service creates the TR-34 token that contains Cred-KRD that is needed by the KDH.
  - CredKRD: (INPUT, *cred_krd*). KRD credential (X.509 certificate) with ID and public key.
  - CT-KRD: (OUTPUT, *output_token*). Credential token for KRD, containing Cred-KRD in DER format.
- BINDRV: The TR34 BIND token (CT-KDH) RECEIVE service processes the BIND request on the KRD.

  - CT-KDH token: (INPUT, *input_token*). BIND token received from KDH.
  - CredKDH; (OUTPUT, *output_token*). Credential (X.509 certificate), in DER format, for the KDH, needs to be stored in the KRD.
- UNBINDRV: The TR34 UNBIND token (UBT-KDH) RECEIVE service processes the UNBIND request on the KRD.
  - UBT-KDH token: (INPUT, *input_token*). UNBIND token received from KDH.
  - CredKDH: (INPUT, *cred_kdh*). KDH credential (X.509 certificate) with ID and public key.
  - CredKRD: (INPUT, *cred_krd*). KRD credential (X.509 certificate) with ID and public key.
  - RT-KRD: (INPUT, *random_number_token*). Token originally sent by the KRD to the KDH and now used for validation.
  - <validity> : (OUTPUT, return/reason code). UBT-KDH – is – valid.
- REBINDRV: The TR34 REBIND token (RBT-KDH). RECEIVE service processes the REBIND request on the KRD.
  - RBT-KDH: (INPUT, *input_token*). REBIND token received from KDH.
  - CredKDH: (INPUT, *cred_kdh*). Old KDH credential (X.509 certificate) with ID and public key.
  - CredKRD: (INPUT, *cred_krd*). KRD credential (X.509 certificate) with ID and public key.
  - RT-KRD: (INPUT, *random_number_token*). Token originally sent by the KRD to the KDH and now used for validation.
  - <validity>: (OUTPUT, return/reason code). RBT-KDH – is – valid.
  - Cred-KDH-NEW: (OUTPUT, *output_token*). New KRD credential (X.509 certificate), in DER format, needs to be stored in the KRD.

Notes:

1. The RT-KRD token can be created with correct formatting using the RT-KRD processing of the CSNBRNGL service. See “Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)” on page 388 for more details.
2. RSA keys with 2048 bit and 3072 bit modulus are supported by CCA. Public exponent 65537 is currently the only supported exponent reflecting the support documented in ASC X9 TR-34-2019. This allows strength equivalent to an AES 128-bit key. TR-34 explicitly supports only RSA 2048-bit keys so some vendors will only support that key size.

Format

```
CALL CSNDT34C(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              input_token_length,
              input_token,
              cred_kdh_length,
              cred_kdh,
              cred_krd_length,
              cred_krd,
              random_number_token_length,
              random_number_token,
              output_token_length,
              output_token,
              reserved_length,
              reserved_data)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be between 1 and 4, inclusive.

rule_array

Direction	Type
Input	Character

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 433. Keywords for TR-34 Bind-Complete

Keyword	Meaning
Requested action (one, required).	

Table 433. Keywords for TR-34 Bind-Complete (continued)

Keyword	Meaning
BINDKRDC	TR34 BIND CTKRD creation service. Creates the KRD credential token that is needed by the KDH to take the next step in the TR-34 BIND action.
BINDRV	TR34 BIND CTKDH RECEIVE service. Receives and processes the token sent by the KDH to the KRD to accomplish the BIND action in the TR-34 protocol. This binds the KRD to the KDH for a later key distribution action.
UNBINDRV	TR34 UNBIND UBTKDH RECEIVE service. Receives and processes the token sent by the KDH to the KRD to accomplish the UNBIND action in the TR-34 protocol. This frees the KRD from the currently bound KDH and causes the KRD to remove all keys received while bound to this KDH.
REBINDRV	TR34 REBIND RBTKDH RECEIVE service. Receives and processes the token sent by the KDH to the KRD to accomplish the REBIND action in the TR-34 protocol. This frees the KRD from the current binding key of the KDH and binds the KRD to a new binding key from the KDH. This also causes the KRD to remove all keys received while bound to the KDH under the prior binding key.
Public Key Infrastructure Usage (one, optional).	
PKI-CHK	Specifies that the X.509 certificate for the other party (KRD) is to be validated against the trust chain of the PKI hosted in the adapter. This requires that the CA credentials have been installed using the Trusted Key Entry (TKE) workstation. This is required for compliance-tag key token export with TR-34 services. This is the default. Cannot be combined with BINDKRDC. There are no other-party credentials to evaluate.
PKI-NONE	Specifies that the X.509 certificate for the other party (KRD) is not to be validated against the trust chain of the PKI hosted in the adapter. This is suitable if the certificate has been validated using host-based PKI services. Cannot be combined with BINDKRDC. There are no other-party credentials to evaluate.
CRL expiration date checking	
CRLEXPCK	Fail the request if the certificate revocation list (CRL) in *crl* is expired. This is the default.
CRLEXPAL	Do not fail the request if the certificate revocation list (CRL) in *crl* is expired. Instead, return an informational *reason_code*.
KRD certificate date checking	
RCTEXPCK	Fail the request if the X.509 certificate in *cred_krd* is expired. This is the default.
RCTEXPAL	Do not fail the request if the X.509 certificate in *cred_krd* is expired. Instead, return an informational *reason_code*.

input_token_length

Direction	Type
Input	Integer

The length of the *input_token* parameter in bytes. The maximum length is 9000 bytes. When the requested action keyword is BINDKRDC, the value must be 0.

input_token

Direction	Type
Input	String

The DER encoded TR-34 token object. The requested action keyword defines the object.

When the *input_token_length* is zero, this parameter is ignored.

The requested action keyword determines the input token:

BINDRV
The BIND token received from the KDH (CTKDH).

UNBINDRV
The UNBIND token received from the KDH (UBTKDH).

REBINDRV
The REBIND token received from the KDH (RBTKDH).

cred_kdh_length

Direction	Type
Input	Integer

The length of the *cred_kdh* parameter in bytes. The maximum length is 3500 bytes. When the requested action keyword is BINDKRDC or BINDRV, the value must be 0.

cred_kdh

Direction	Type
Input	String

The X.509 certificate that is the credential of the KDH for the requested service. The certificate may be in DER or PEM format.

When the *cred_kdh_length* is zero, this parameter is ignored.

Note: This service is acting as the KDH so the *cred_kdh* is not expected to validate against the internal PKI of the adapter. Use the PKI-NONE keyword to override this validation.

cred_krd_length

Direction	Type
Input	Integer

The length of the *cred_krd* parameter in bytes. The maximum length is 3500 bytes. When the requested action keyword is BINDRV, the value must be 0.

cred_krd

Direction	Type
Input	String

The X.509 certificate that is the credential of the KRD for the requested service (the CredKRD). The certificate may be in DER or PEM format.

When the *cred_krd_length* is zero, this parameter is ignored.

Note: This service is acting as the KDH so the *cred_krd* is normally expected to validate against the internal PKI of the adapter. Use the PKI-NONE keyword to override this validation.

random_number_token_length

Direction	Type
Input	Integer

The length of the *random_number_token* parameter. The maximum length is 200 bytes. When the requested action keyword is BINDKRDC or BINDRV, the value must be zero.

random_number_token

Direction	Type
Input	String

The DER encoded random number token RTKRD that was sent to the KDH. The *random_number_token* is used by the KRD to validate the random number sent by the KDH in the *input_token* parameter.

When the *random_number_token_length* is zero, this parameter is ignored.

output_token_length

Direction	Type
Input/Output	Integer

The length of the *output_token* parameter in bytes. The maximum length is 3500 bytes. On input, the value is the size of the buffer to receive the *output_token*. On output, the value is the actual size of the data returned in the *output_token* parameter.

When the requested action keyword is UNBINDRV, the value must be zero.

output_token

Direction	Type
Output	String

The generated DER encoded TR-34 token.

BINDKRDC
The TR-34 credential token for the KRD (CTKRD).

BINDRV and REBINDRV
The TR-34 credential X.509 certificate for the KDH (CredKDH).

When the *output_token_length* is zero, this parameter is ignored.

reserved_length

Direction	Type
Input/Output	Integer

This parameter is reserved. The value must be zero.

reserved

Direction	Type
Input/Output	String

This parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the PKDS.

Access control points

The **TR-34 Bind-Complete** access control in the domain role controls the function of this service.

The following table shows the access controls in the domain role that control the function of this service:

Table 434. Access control points for TR-34 Bind-Complete

Rule array keyword	Access control
BINDKRDC	TR-34 Bind-Complete - Allow BINDKRDC.
BINDRV	TR-34 Bind-Complete - Allow BINDRV.
UNBINDRV	TR-34 Bind-Complete - Allow UNBINDRV.
REBINDRV	TR-34 Bind-Complete - Allow REBINDRV.
PKI-NONE	Permit X.509 without PKI root validation.
RCTEXPAL	TR-34 - Allow expired KRD certificate.
CRLEXPAL	TR-34 - Allow expired CRL.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 435. TR-34 Bind-Complete required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**		This service is not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	This service is not supported.
	Crypto Express6 CCA Coprocessor	This service requires the July 2019 or later licensed internal code. Rules CRLEXPCK, CRLEXPAL, RCTEXPCK, RCTEXPAL, and *input_token_length* greater than 3500 bytes require CCA release 6.7.18 or higher.

Table 435. TR-34 Bind-Complete required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	This service is not supported.
	Crypto Express6 CCA Coprocessor	Rules CRLEXPCK, CRLEXPAL, RCTEXPCK, RCTEXPAL, and *input_token_length* greater than 3500 bytes require CCA release 6.7.18 or higher.
	Crypto Express7 CCA Coprocessor	Rules CRLEXPCK, CRLEXPAL, RCTEXPCK, RCTEXPAL, and *input_token_length* greater than 3500 bytes require CCA release 7.4.25 or higher.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

TR-34 Key Distribution (CSNDT34D and CSNFT34D)

The TR-34 Protocol Key Distribution service is used for operations that take place at the Key Distribution Host (KDH) during TR-34 Protocol Key Transport related operations, as specified in X9 TR34-2012.

The TR-34 Protocol describes the data objects and cryptograms needed for securely provisioning a symmetric key from a Key Distribution Host (KDH) to a Key Receiving Device (KRD). Although designed for 1-way key distribution, such as between a host computer (KDH) and an ATM (KRD), the protocol can be used for peer-to-peer key distribution as well. The main requirement for either use of the protocol is a Certificate Authority (CA) trusted by both the KDH and the KRD.

The callable service name for AMODE(64) invocation is CSNFT34D.

This service is used to perform these operations:

- 2PASSCRE: The TR34 Key Transport (2-pass) token (KT-KDH) CREATE service.
  - Kn-T: (INPUT, *source_key_identifier*). CCA key token to be exported to the KRD.
  - KEK-N: (INPUT, *unwrap_kek_identifier*). CCA internal key token for KEK to unwrap Kn-T.
  - RT-KRD: (INPUT, *freshness_indicator*). Random number token received from KRD.
  - CRL-CA: (INPUT, *crl*). Certificate Revocation List from CA.
  - CredKDH: (INPUT, *cred_kdh*). KDH credential (X.509 certificate) with ID and public key.
  - CredKRD: (INPUT, *cred_krd*). KRD credential (X.509 certificate) with ID and public key. The public key from this is used to RSA-encipher the RSA-encrypted part of the key block.
  - D-kdh: (INPUT, *private_key_identifier*). Private key to sign data block.
  - KVN: (INPUT, *key_version_number*). Two-byte version number for TR-31 Key block header,
  - Opt-blocks: (INPUT, *opt_blocks*). Application generated optional blocks for TR-31 Key block header.
  - KT-KDH: (OUTPUT, *output_token*). Key transport (2-pass) token in DER format.
- 1PASSCRE: The TR34 Key Transport (1-pass) token (KT-KDH) CREATE service.
  - Kn-T: (INPUT, *source_key_identifier*). CCA key token to be exported to the KRD.
  - KEK-N: (INPUT, *unwrap_kek_identifier*). CCA internal key token for KEK to unwrap Kn-T.
  - Timestamp: (INPUT, *freshness_indicator*). Freshness indicator to defend against replay attacks.
  - CRL-CA: (INPUT, *crl*). Certificate Revocation List from CA.

- CredKDH: (INPUT, *cred_kdh*). KDH credential (X.509 certificate) with ID and public key.
- CredKRD: (INPUT, *cred_krd*). KRD credential (X.509 certificate) with ID and public key. The public key from this is used to RSA-encipher the RSA-encrypted part of the key block.
- D-kdh: (INPUT, *private_key_identifier*). Private key to sign data block.
- KVN: (INPUT, *key_version_number*). Two-byte version number for TR-31 Key block header.
- Opt-blocks: (INPUT, *opt_blocks*). Application generated optional blocks for TR-31 Key block header.
- KT-KDH: (OUTPUT, *output_token*), Key transport (1-pass) token in DER format.

Notes:

1. Comp-tag AES and DES tokens are exportable using this service. A comp-tag RSA private key is required.
2. The RT-KRD token can be created with correct formatting using the RT-KRD processing of the CSNBRNGL service. See "Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)" on page 388 for more details.
3. RSA keys with 2048 bit and 3072 bit modulus are supported by CCA. Public exponent 65537 is currently the only supported exponent reflecting the support documented in ASC X9 TR-34-2019. This allows strength equivalent to an AES 128-bit key. TR-34 explicitly supports only RSA 2048-bit keys so some vendors will only support that key size.

Format

```
CALL CSNDT34D(
            return_code,
            reason_code,
            exit_data_length,
            exit_data,
            rule_array_count,
            rule_array,
            source_key_identifier_length,
            source_key_identifier,
            unwrap_kek_identifier_length,
            unwrap_kek_identifier,
            freshness_indicator_length,
            freshness_indicator,
            crl_length,
            crl,
            cred_kdh_length,
            cred_kdh,
            cred_krd_length,
            cred_krd,
            private_key_identifier_length,
            private_key_identifier,
            key_version_number,
            opt_blocks_length,
            opt_blocks,
            output_token_length,
            output_token,
            reserved_length,
            reserved_data)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be between 5 and 11, inclusive.

rule_array

Direction	Type
Input	Character

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 436. Keywords for TR-34 Key Distribution

Keyword	Meaning
Requested action (one, required).	
2PASSCRE	TR34 Key Transport (2-pass) KTKDH CREATE service.
1PASSCRE	TR34 Key Transport (1-pass) KTKDH CREATE service.
Public Key Infrastructure Usage (one, optional).	
PKI-CHK	Specifies that the X.509 certificate for the other party (KRD) is to be validated against the trust chain of the PKI hosted in the adapter. This requires that the CA credentials have been installed using the Trusted Key Entry (TKE) workstation. This is required for compliance-tag key token export with TR-34 services. This is the default.

Table 436. Keywords for TR-34 Key Distribution (continued)	
Keyword	**Meaning**
PKI-NONE	Specifies that the X.509 certificate for the other party (KRD) is not to be validated against the trust chain of the PKI hosted in the adapter. This is suitable if the certificate has been validated using host-based PKI services.
Source key algorithm (one, required).	
SKEY-DES	Specifies that the *source_key_identifer* is a DES key. If the *source_key_identifier* is external then the *unwrap_kek_identifier* may be a DES or AES key..
SKEY-AES	Specifies that the *source_key_identifer* is an AES key. If the *source_key_identifier* is external then the *unwrap_kek_identifier* is also an AES key.
Key block version (one, required). Specifies which version of the TR-31 key block to use. However, this value has no impact to the wrapping of the TR-34 key block or format of the key block. Choose this value for compatibility with the target KRD.	
VARXOR-A	Specifies to use TR-31 Key Block Version ID of "A" (X'41'). This is typically a DES version.
VARDRV-B	Specifies to use TR-31 Key Block Version ID of "B" (X'42'). This is typically a DES version.
VARXOR-C	Specifies to use TR-31 Key Block Version ID of "C" (X'43'). This is typically a DES version.
VARDRV-D	Specifies to use TR-31 Key Block Version ID of “D” (X’44’). This is typically for AES or DES.
EncryptedContent format (one, optional). TR-34-2012 in examples B.8 and B.9 places the EncryptedContent according to X9.73, after the ContentEncryptionAlgorithmIdentifier. TR-34-2019 has changed this, introducing a small incompatibility, placing the EncryptedContent as the last field inside of the ContentEncryptionAlgorithmIdentifier. Check with your ATM vendor to ensure that you are using the correct option.	
T34-2012	Builds the EncryptedContentInfo section of the EnvelopedData of the *output_token* according to the sample description in X9 TR-34-2012 sections B.8 and B.9. The EncryptedContent appears after the ContentEncryptionAlgorithmIdentifier. This keyword impact applies to either the 1PASSCRE or 2PASSCRE *output_token* as the compatibility issue applies to both. This is the default.
T34-2019	Builds the EncryptedContentInfo section of the EnvelopedData of the *output_token* according to the sample description in X9 TR-34-2019 sections B.8 and B.9. The EncryptedContent appears inside the ContentEncryptionAlgorithmIdentifier. This keyword impact applies to either the 1PASSCRE or 2PASSCRE *output_token* as the compatibility issue applies to both.
SignedAttributes order (One, optional). TR-34-2012 and TR-34-2019 define a SignedAttributes object that is part of the SignerInfo in the output_token. The SignedAttributes object is a 'Set-of' as defined in ITU T-REC-X.690-201508. The components of a 'Set-of' are meant to be ordered by increasing size. Some ATM vendors implemented the 'Set-of' object with a static order of components matching the order shown in the TR-34-2012 (repeated in TR-34-2019) section B.9.1. Check with your ATM vendor to ensure you are using the correct option. Only valid with 2PASSCRE.	
SASORTSZ	Use the flexible ordering for 'Set-of' components as defined in ITU T-REC-X.690-201508, which requires order by increasing size. This is the default.

Table 436. Keywords for TR-34 Key Distribution (continued)	
Keyword	**Meaning**
SASORTEX	Use the static ordering shown in the parsed example of TR-34-2012 section B.9.1.
CRL expiration date checking	
CRLEXPCK	Fail the request if the certificate revocation list (CRL) in *crl* is expired. This is the default.
CRLEXPAL	Do not fail the request if the certificate revocation list (CRL) in *crl* is expired. Instead, return an informational *reason_code*.
KRD certificate date checking	
RCTEXPCK	Fail the request if the X.509 certificate in *cred_krd* is expired. This is the default.
RCTEXPAL	Do not fail the request if the X.509 certificate in *cred_krd* is expired. Instead, return an informational *reason_code*.

Table 437. TR-31 key usage value for output key block			
Keyword	**TR-31 key usage**	**CCA key types**	**Meaning**
TR-31 key usage value for output key block (one required). The TR-34 key transport key block contains a TR-31 key header contains the key usage.			
KEK	"K0"	IMPORTER or EXPORTER DES or AES.	Specifies to export a key-encrypting key (KEK) to an external TR-31 key block. You must select one TR-31 mode of key use keyword from Table 443 on page 1097 with this usage keyword. The table shows all the supported translations for key usage keyword KEK. It also shows the access control commands that must be enabled in the active role to use the combination of inputs shown.
KEK-WRAP	"K1"	IMPORTER or EXPORTER DES or AES. AES must have TR-31 wrap permission.	AES must have TR-31 wrap permission. Specifies to export a TR-31 key block protection key (KEK-WRAP) to an external TR-31 key block. You must select one TR-31 mode of key use keyword from Table 443 on page 1097 with this usage keyword. The table shows all the supported translations for key usage keyword KEK-WRAP. It also shows the access control commands that must be enabled in the active role to use the combination of inputs shown.

Table 438. TR-31 mode of key use			
Keyword	**TR-31 key usage**	**CCA key types**	**Meaning**
TR-31 mode of key use (one required). Only those TR-31 modes shown are supported. Modes requested must match or be included in the capabilities of the key being exported. For example, DEC-ONLY and GENVER cannot be used if the exported key has ENC-ONLY capability.			
DEC-ONLY	"D"	KEK, KEK-WRAP	Specifies to decrypt and unwrap only.
ENC-ONLY	"E"	KEK, KEK-WRAP	Specifies to encrypt and wrap only.

Table 439. TR-31 exportability

Keyword	TR-31 mode	Meaning
TR-31 exportability (one, optional). Use to set exportability field in TR-31 key block. Defines whether the key may be transferred outside the cryptographic domain in which the key is found.		
EXP-ANY	"S"	Specifies that the key in the TR-31 key block is sensitive, exportable under a key-encrypting key in a form not necessarily meeting the requirements of X9.24 parts 1 or 2. This is the default.
EXP-TRST	"E"	Specifies that the key in the TR-31 key block is exportable under a key-encrypting key in a form that meets the requirements of X9.24 Parts 1 or 2. **Note:** A TR-31 key block with a key block version ID of "B" or "C" and an exportability field value of "E" cannot be wrapped by a key-encrypting key that is wrapped in ECB mode (legacy wrap mode). This is because ECB mode does not comply with ANS X9.24 Part 1.
EXP-NONE	"N"	Specifies that the key in the TR-31 key block is non-exportable.

The following tables map the CCA key types for the DES and AES keys to the corresponding allowed key usage keywords and mode of use keywords, as well as the access control points that are required.

Table 440. Export translation table for DES keys in TR-34 key blocks

CCA key type (and required attributes)	Key usage keyword	Key block protection method keyword	TR-31 mode of key use keyword	Offset (hex)	Access control name
Security note: The TR-31 modes requested must match or be included in the capabilities of the key being exported. For example, DEC-ONLY and GENVER cannot be used if the exported key has ENC-ONLY capability.					
DES EXPORTER	KEK ("K0") KEK-WRAP ("K1")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	ENC-ONLY ("E")	0242	TR-34 Key Distribution - Permit DES EXPORTER to K0 or K1
DES IMPORTER	KEK ("K0") KEK-WRAP ("K1")	VARXOR-A, VARDRV-B, VARXOR-C, VARDRV-D	DEC-ONLY ("D")	0243	TR-34 Key Distribution - Permit DES IMPORTER to K0 or K1

Table 441. Export translation table for AES keys in TR-34 key blocks

CCA key type (and required attributes)	Key usage keyword	Key block protection method keyword	TR-31 mode of key use keyword	Offset (hex)	Access control name
Security note: The TR-31 modes requested must match or be included in the capabilities of the key being exported. For example, DEC-ONLY and GENVER cannot be used if the exported key has ENC-ONLY capability.					

Table 441. Export translation table for AES keys in TR-34 key blocks (continued)

CCA key type (and required attributes)	Key usage keyword	Key block protection method keyword	TR-31 mode of key use keyword	Offset (hex)	Access control name
AES EXPORTER	KEK ("K0")	VARDRV-D	ENC-ONLY ("E")	0244	TR-34 Key Distribution - Permit AES EXPORTER to K0
AES EXPORTER (EXPTT31D)	KEK-WRAP ("K1")	VARDRV-D	ENC-ONLY ("E")	0245	TR-34 Key Distribution - Permit AES EXPORTER to K1
AES IMPORTER	KEK ("K0")	VARDRV-D	DEC-ONLY ("D")	0246	TR-34 Key Distribution - Permit AES IMPORTER to K0
AES IMPORTER (IMPTT31D)	KEK-WRAP ("K1")	VARDRV-D	DEC-ONLY ("D")	0247	TR-34 Key Distribution - Permit AES IMPORTER to K1

source_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *source_key_identifier* parameter.

If the *source_key_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992

source_key_identifier

Direction	Type
Input/Output	String

The identifier of the key to be exported using the TR-34 protocol. The key identifier is an internal or external key token or key block or the label of an operational token or block in key storage. If the source key is an external token, an identifier for the KEK that wraps the source key must be passed in the *unwrap_kek_identifier* parameter.

For CCA keys, the source key is a DES EXPORTER or IMPORTER in a 64-byte key token or an AES EXPORTER or IMPORTER in a variable-length key token. The control vector of a DES token or the key usage field of an AES token must not indicate that the key in the token is a partial key. Partial keys are not exportable using TR-34.

For X9.143 (TR-31) keys, the source is a TDES or AES key-encrypting key in a variable length key block. Key usage K0 or K1, algorithm A or T, and mode of use D or E.

When the internal token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key

unwrap_kek_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *unwrap_kek_identifier* parameter.

When the *source_key_identifier* is an internal key, the value must be 0.

When the *unwrap_kek_identifier* contains a label, the length must be 64.

Otherwise, the value must be between the actual length of the token and 9992.

unwrap_kek_identifier

Direction	Type
Input/Output	String

The identifier of the wrapping key of the source key identifier when the source key is an external token. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

When the *unwrap_kek_identifier_length* is zero, this parameter is ignored.

For CCA keys, the identifier is a 64-byte DES key token of type EXPORTER or OKEYXLAT or a variable-length AES key token of key type EXPORTER.

For X9.143 (TR-31) keys, the identifier is an AES or DES key-encrypting key in a variable-length key block: key usages K0 or K1, algorithm A or T, and mode of use E. The key usage must be K1 when the source key is a TR-31 key block and K0 when the source key is a CCA key token.

If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

freshness_indicator_length

Direction	Type
Input	Integer

The length of the *freshness_indicator* parameter in bytes. The maximum length is 200 bytes.

The value is determined by the requested action keyword:

2PASSCRE
This parameter contains the length of the random number token received from the KRD (RTKRD).

1PASSCRE
This parameter contains the length of the ASCII timestamp for use in the key transport token. The value must be 13 or 15.

freshness_indicator

Direction	Type
Input	String

The freshness indicator. The meaning is determined by the requested action keyword:

2PASSCRE
This parameter contains the DER encoded random number token received from the KRD (RT-KRD).

1PASSCRE
This parameter contains an ASCII timestamp for use in the key transport token. The timestamp is encoded as a SigningTime object in the token (OID of iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) signing-time(5)}, defined in IETF RFC 2985). Two input

encodings are accepted for this parameter, UTCTime encoded as YYMMDDHHMMSSZ (13 bytes) and GeneralizedTime encoded as YYYYMMDDHHMMSSZ (15 bytes). Date ranges allowed for each format are:

GeneralizedTime
Dates between 1 January 2050 and 31 December 2105 (inclusive).

UTCTime
Dates between 1 January 1950 and 31 December 2049 (inclusive).

There must be an ASCII 'Z' character at the end. CCA will verify that timestamps passed with either encoding are valid timestamps, but will make no attempt to validate the timestamp against the clock inside the coprocessor.

crl_length

Direction	Type
Input	Integer

The length of the *crl* parameter in bytes. The maximum length is 6000 bytes.

crl

Direction	Type
Input	String

The certificate revocation list (CRL) from the certificate authority the is in common with the KRD for the requested service. The CRL may be in DER or PEM format.

cred_kdh_length

Direction	Type
Input	Integer

The length of the *cred_kdh* parameter in bytes. The maximum length is 3500 bytes.

cred_kdh

Direction	Type
Input	String

The X.509 certificate that is the credential of the KDH for the requested service. The certificate may be in DER or PEM format.

Notes:

- This service is acting as the KDH so the *cred_kdh* is not expected to validate against the internal PKI of the adapter.
- The public key contained in the *cred_kdh* parameter must be the same as the one contained in the *private_key_identifier* parameter.

cred_krd_length

Direction	Type
Input	Integer

The length of the *cred_krd* parameter in bytes. The maximum length is 3500 bytes.

cred_krd

Direction	Type
Input	String

The X.509 certificate that is the credential of the KRD for the requested service (the CredKRD). The certificate may be in DER or PEM format.

The usage attributes in the X.509 certificate must allow keyEncipherment. The public key will be extracted and used to encrypt the ephemeral key wrapping key that protects the TMK that is being transported

Note: This service is acting as the KDH so the *cred_krd* is normally expected to validate against the internal PKI of the adapter. Use the PKI-NONE keyword to override this validation.

private_key_identifier_length

Direction	Type
Input	Integer

The length of the *private_key_identifier* parameter. If the *private_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 3500.

private_key_identifier

Direction	Type
Input	String

The identifier of the private key used to sign the output token. The key identifier is an operational RSA secure token or the label of such a token in key storage.

The key usage of the token must allow digital signature. Retained private keys are not supported in this service.

Note: The public key contained in the *cred_kdh* parameter must be the same as the one contained in the *private_key_identifier* parameter.

key_version_number

Direction	Type
Input	String

The two-byte value to be copied into the Key Version Number field of the output TR-31 key block. If no key version number is needed, the value must be EBCDIC ("00"). The value is not allowed to indicate a partial key.

opt_blks_length

Direction	Type
Input	Integer

The length of the *opt_blocks* parameter in bytes. If no optional data is to be included in the TR-31 key block, this parameter must be set to zero.

opt_blks

Direction	Type
Input	String

The optional block data to be included in the output TR-31 key block. The optional block data is prepared using the TR-31 Optional Data Build callable service and must be in ASCII. This parameter is ignored if *opt_blks_length* is zero.

output_token_length

Direction	Type
Input/Output	Integer

The length of the *output_token* parameter in bytes. On input, the value is the size of the buffer to receive the *output_token*. On output, the value is the actual size of the data returned in the *output_token* parameter. The maximum length on output is 9000 bytes.

output_token

Direction	Type
Output	String

The generated DER encoded TR-34 token.

2PASSCRE
This parameter will contain the 2 pass TR-34 key transport token (KTKDH).

1PASSCRE
This parameter will contain the 1 pass TR-34 key transport token (KTKDH).

reserved_length

Direction	Type
Input/Output	Integer

This parameter is reserved. The value must be zero.

reserved

Direction	Type
Input/Output	String

This parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS and PKDS.

Access control points

The **TR-34 Key Distribution** access control in the domain role controls the function of this service.

The following table shows the access controls in the domain role that control the function of this service:

Table 442. Access control points for TR-34 Key Distribution

Rule array keyword	Access control
2PASSCRE	TR-34 Key Distribution – Allow 2PASSCRE.
1PASSCRE	TR-34 Key Distribution – Allow 1PASSCRE.
PKI-NONE	Permit X.509 without PKI root validation.
RCTEXPAL	TR-34 - Allow expired KRD certificate.

Table 442. Access control points for TR-34 Key Distribution (continued)	
Rule array keyword	**Access control**
CRLEXPAL	TR-34 - Allow expired CRL.

Table 443. Valid CCA to TR-34 Export Translations and Required Access Controls			
TR-31 key usage keyword	**Access control name**	**Offset (hex)**	**Specific key type and control vector attributes**
"K0" and "K1": TR-31 key encryption or wrapping, or key block protection keys			
KEK KEK-WRAP	TR-34 Key Distribution - Permit DES EXPORTER to K0 or K1	0242	See Table 440 on page 1091.
KEK KEK-WRAP	TR-34 Key Distribution - Permit DES IMPORTER to K0 or K1	0243	
KEK	TR-34 Key Distribution - Permit DES EXPORTER to K0	0244	See Table 441 on page 1091.
KEK KEK-WRAP	TR-34 Key Distribution - Permit DES EXPORTER to K1	0245	
KEK	TR-34 Key Distribution - Permit DES IMPORTER to K0	0246	
KEK KEK-WRAP	TR-34 Key Distribution - Permit DES IMPORTER to K1	0247	

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 444. TR-34 Key Distribution required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**		This service is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	This service requires the July 2019 or later licensed internal code. Rule keywords SASORTSZ and SASORTEX are not supported. Rules CRLEXPCK, CRLEXPAL, RCTEXPCK, RCTEXPAL, and *crl_length* greater than 3500 bytes require CCA release 6.7.18 or higher. X9.143 key blocks are not supported.

Table 444. TR-34 Key Distribution required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule keywords SASORTSZ and SASORTEX are not supported. Rules CRLEXPCK, CRLEXPAL, RCTEXPCK, RCTEXPAL, and *crl_length* greater than 3500 bytes require CCA release 6.7.18 or higher. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule keywords SASORTSZ and SASORTEX require the CEX7S MCLs or later listed in APAR OA60365. Rules CRLEXPCK, CRLEXPAL, RCTEXPCK, RCTEXPAL, and *crl_length* greater than 3500 bytes require CCA release 7.4.25 or higher. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

TR-34 Key Receive (CSNDT34R and CSNFT34R)

The TR-34 Protocol Key Receive service is used for operations that take place at the Key Receiving Device (KRD) during TR-34 Protocol Key Transport related operations, as specified in X9 TR34-2012.

The TR-34 Protocol describes the data objects and cryptograms needed for securely provisioning a symmetric key from a Key Distribution Host (KDH) to a Key Receiving Device (KRD). Although designed for 1-way key distribution, such as between a host computer (KDH) and an ATM (KRD), the protocol can be used for peer-to-peer key distribution as well. The main requirement for either use of the protocol is a Certificate Authority (CA) trusted by both the KDH and the KRD.

The callable service name for AMODE(64) invocation is CSNFT34R.

This service is used to perform these operations:

- 2PASSRCV: The TR34 Key Transport (2-pass) token (KT-KDH) RECEIVE service.
  - KT-KDH: (INPUT, *input_token*). Key transport (2-pass) token received from KDH.
  - CredKDH: (INPUT, *cred_kdh*). KDH credential (X.509 certificate) with ID and public key.
  - RT-KRD: (INPUT, *input_freshness_indicator*).Token originally sent by the KRD to the KDH and now used for validation.
  - D-krd: (INPUT, *private_key_identifier*). The private key matching the public key in CredKRD.
  - Kn-T: (OUTPUT, *output_key_identifier*). CCA key token containing the transported TMK/KBPK.
- 1PASSRCV: The TR34 Key Transport (1-pass) token (KT-KDH) RECEIVE service.
  - KT-KDH: (INPUT, *input_token*). Key transport (1-pass) token received from KDH.

- CredKDH: (INPUT, *cred_kdh*). KDH credential (X.509 certificate) with ID and public key.
- Timestamp-min: (INPUT, *input_freshness_indicator*). Minimum timestamp value allowed, could be the last timestamp received or a bootstrap value.
- D-krd: (INPUT, *private_key_identifier*). The private key matching the public key in CredKRD.
- Kn-T: (OUTPUT, *output_key_identifier*). CCA key token containing the transported TMK/KBPK.
- Timestamp-rcv: (OUTPUT, *output_freshness_indicator*). The timestamp value received in the KT-KDH.

Notes:

1. Comp-tag AES and DES tokens are exportable using this service. A comp-tag RSA private key is required.
2. The RT-KRD token can be created with correct formatting using the RT-KRD processing of the CSNBRNGL service. See "Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)" on page 388 for more details.
3. RSA keys with 2048 bit and 3072 bit modulus are supported by CCA. Public exponent 65537 is currently the only supported exponent reflecting the support documented in ASC X9 TR-34-2019. This allows strength equivalent to an AES 128-bit key. TR-34 explicitly supports only RSA 2048-bit keys so some vendors will only support that key size.

Format

```
CALL CSNDT34R(
            return_code,
            reason_code,
            exit_data_length,
            exit_data,
            rule_array_count,
            rule_array,
            input_token_length,
            input_token,
            cred_kdh_length,
            cred_kdh,
            input_freshness_indicator_length,
            input_freshness_indicator,
            private_key_identifier_length,
            private_key_identifier,
            output_key_identifier_length,
            output_key_identifier,
            output_freshness_indicator_length,
            output_freshness_indicator,
            reserved_length,
            reserved_data)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter.

The value must be between 1 and 7, inclusive.

rule_array

Direction	Type
Input	Character

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 445. Keywords for TR-34 Key Receive

Keyword	Meaning
Requested action (one, required).	
2PASSRCV	TR34 Key Transport (2-pass) KTKDH RECEIVE service.
1PASSRCV	TR34 Key Transport (1-pass) KTKDH RECEIVE service.
Public Key Infrastructure Usage (one, optional).	
PKI-CHK	Specifies that the X.509 certificate for the other party (KRD) is to be validated against the trust chain of the PKI hosted in the adapter. This requires that the CA credentials have been installed using the Trusted Key Entry (TKE) workstation. This is required for compliance-tag key token export with TR-34 services. This is the default.
PKI-NONE	Specifies that the X.509 certificate for the other party (KRD) is not to be validated against the trust chain of the PKI hosted in the adapter. This is suitable if the certificate has been validated using host-based PKI services.
Freshness Indicator Usage (one, required with 1PASSRCV). Only valid with keyword 1PASSRCV.	

Table 445. Keywords for TR-34 Key Receive (continued)

Keyword	Meaning
TIME-CHK	Specifies to check the timestamp in the *input_freshness_indicator* parameter against the timestamp in the KTKDH received in the *input_token* parameter. The verification step will only check that the timestamp in the KTKDH is newer than timestamp in the *input_freshness_indicator* parameter. The timestamp from the KTKDH will still be returned in the *output_freshness_indicator* parameter.
TIMENONE	Specifies to not check the timestamp in the *input_freshness_indicator* parameter against the timestamp in the KTKDH received in the *input_token* parameter. The timestamp from the KTKDH will be returned in the *output_freshness_indicator* parameter so that the application can verify the value.
Output Token Type (one, optional).	
CCA-TOK	Create a CCA output token. This is the default.
TR31-TOK	Create a TR-31 output key block.
Key Wrapping Method (One Optional). Applicable only to rule CCA-TOK and DES algorithm imported keys.	
USECONFG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.
WRAP-ENH	Specifies that the enhanced wrapping method and SHA-1 is to be used to wrap the key.
WRAPENH2	Specifies to wrap the key using the enhanced wrapping method and SHA-256. Valid only with triple-length keys. This is the default for triple-length keys.
WRAPENH3	Specifies that the enhanced wrapping method and SHA-256 and CMAC authentication code is to be used to wrap the key.
WRAP-ECB	Specifies that the original wrapping method is to be used.
Translation Control (One Optional). Applicable only to DES algorithm imported keys.	
ENH-ONLY	Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the CCA legacy method. This restricts translation to the CCA legacy method. If the keyword is not specified, translation to the CCA legacy method will be allowed. This turns on bit 56 in the control vector. This is the default when the wrapping method is WRAPENH2 or WRAPENH3.
CRL expiration date checking	
CRLEXPCK	Fail the request if the certificate revocation list (CRL) in the TR-34 token object in *input_token* is expired. This is the default.
CRLEXPAL	Do not fail the request if the certificate revocation list (CRL) in the TR-34 token object in *input_token* is expired. Instead, return an informational *reason_code*.

Table 446. Input translation table for DES key usage

TR-31 mode of key use	Key block version ID	TR-31 mode of key use	Rule-array keywords	CCA key type and key-usage attributes of output key	Offset (hex)	Access control name
K0, K1	"A", "B", "C", "D"	"E"	N/A	EXPORTER	0248	TR-34 Key Receive – Permit DES EXPORTER
K0, K1	"A", "B", "C", "D"	"D"	N/A	IMPORTER	0249	TR-34 Key Receive – Permit DES IMPORTER

Table 447. Input translation table for AES key usage

TR-31 mode of key use	Key block version ID	TR-31 mode of key use	Rule-array keywords	CCA key type and key-usage attributes of output key	Offset (hex)	Access control name
K0	"D"	"E"	N/A	EXPORTER	024A	TR-34 Key Receive – Permit AES EXPORTER
		"D"	N/A	IMPORTER	024B	TR-34 Key Receive – Permit AES IMPORTER
K1	"D"	"E"	N/A	EXPORTER + EXPTT31D	024C	TR-34 Key Receive – Permit AES EXPORTER with EXPTT31D
		"D"	N/A	IMPORTER + IMPTT31D	024D	TR-34 Key Receive – Permit AES IMPORTER with IMPTT31D

input_token_length

Direction	Type
Input	Integer

The length of the *input_token* parameter in bytes. The maximum length is 9000 bytes.

input_token

Direction	Type
Input	String

The DER encoded TR-34 token object. The requested action keyword defines the object.

2PASSRCV
This parameter must contain the 2 pass key transport token received from the KDH (KTKDH).

1PASSRCV
This parameter must contain the 1 pass key transport token received from the KDH (KTKDH)

cred_kdh_length

Direction	Type
Input	Integer

The length of the *cred_kdh* parameter in bytes. The maximum length is 3500 bytes.

cred_kdh

Direction	Type
Input	String

The X.509 certificate that is the credential of the KDH for the requested service. The certificate may be in DER or PEM format.

Note: This service is acting as the KDH so the *cred_kdh* is not expected to validate against the internal PKI of the adapter.

input_freshness_indicator_length

Direction	Type
Input/Output	Integer

The length of the *input_freshness_indicator* parameter in bytes. The maximum length is 200 bytes.

The value is determined by the requested action keyword:

2PASSRCV
This parameter contains the length of the random number token received from the KRD (RTKRD).

1PASSRCV
This parameter contains the length of the ASCII timestamp for use in the key transport token. The value must be 13 or 15. When TIMENONE is passed, the value must be 0.

input_freshness_indicator

Direction	Type
Input/Output	String

The freshness indicator. When the value of *input_freshness_indicator_length* is 0, this parameter is ignored.

The meaning is determined by the requested action keyword:

2PASSRCV
This parameter contains the DER encoded random number token received from the KRD (RT-KRD).

1PASSRCV
This parameter contains an ASCII timestamp for use in the key transport token. The timestamp is encoded as a SigningTime object in the token (OID of iso(1) member-body(2) us(840)

rsadsi(113549) pkcs(1) pkcs-9(9) signing-time(5)}, defined in IETF RFC 2985). Two input encodings are accepted for this parameter, UTCTime encoded as YYMMDDHHMMSSZ (13 bytes) and GeneralizedTime encoded as YYYYMMDDHHMMSSZ (15 bytes). Date ranges allowed for each format are:

GeneralizedTime
Dates between 1 January 2050 and 31 December 2105 (inclusive).

UTCTime
Dates between 1 January 1950 and 31 December 2049 (inclusive).

There must be an ASCII 'Z' character at the end. CCA will verify that timestamps passed with either encoding are valid timestamps, but will make no attempt to validate the timestamp against the clock inside the coprocessor.

private_key_identifier_length

Direction	Type
Input	Integer

The length of the *private_key_identifier* parameter. If the *private_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 3500.

private_key_identifier

Direction	Type
Input	String

The identifier of the private key used to decrypt the key block in the input token. The key identifier is an operational RSA secure token or the label of such a token in key storage.

The key usage of the token must allow digital signature. Retained private keys are not supported in this service.

output_key_identifier_length

Direction	Type
Input/Output	Integer

The length of the *output_key_identifier* parameter, in bytes.

On input, it specifies the length of the buffer for *output_key_identifier*. The maximum length required for CCA key tokens is 725. The maximum length required for X9.143 (TR-31) key blocks is 9992.

On output, it will contain the length of the token returned

output_key_identifier

Direction	Type
Output	String

The received key token from the TR-34 key block. The output token will be a CCA internal key token or X9.143 (TR-31) key block containing the key received in the TR-34 key block.

output_freshness_indicator_length

Direction	Type
Input/Output	Integer

The length of the *output_freshness_indicator* parameter, in bytes. On input, it specifies the length of the buffer for *output_freshness_indicator* and must be at least 15 bytes long. On output, it will contain

the length of the indicator returned. When the requested action keyword is 2PASSRCV, the value must be 0.

output_freshness_indicator

Direction	Type
Input/Output	String

The output freshness indicator. When the value of *output_freshness_indicator_length* is 0, this parameter is ignored.

When the requested action keyword is 1PASSRCV, this parameter contains the ASCII timestamp received in the key transport token received from the KDH. Two encodings are used for this parameter: UTCTime encoded as YYMMDDHHMMSSZ (13 bytes) and GeneralizedTime encoded as YYYYMMDDHHMMSSZ (15 bytes). Date ranges allowed for each format are:

UTCTime
Dates between 1 January 1970 and 31 December 2049 (inclusive).

GeneralizedTime
Dates between 1 January 2050 and 31 December 2105 (inclusive).

There will be an ASCII 'Z' character at the end. ICSF will verify that the timestamp returned is a valid timestamp from the key transport token received from the KDH (KT-KDH), but makes no attempt to validate the timestamp against the clock inside the coprocessor.

reserved_length

Direction	Type
Input/Output	Integer

This parameter is reserved. The value must be zero.

reserved

Direction	Type
Input/Output	String

This parameter is ignored.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the PKDS.

Access control points

The **TR-34 Key Receive** access control in the domain role controls the function of this service.

The following table shows the access controls in the domain role that control the function of this service:

Table 448. Access control points for TR-34 Key Receive

Rule array keyword	Access control
2PASSRCV	TR-34 Key Receive – Allow 2PASSRCV.
1PASSRCV	TR-34 Key Receive – Allow 1PASSRCV.
PKI-NONE	Permit X.509 without PKI root validation.
CRLEXPAL	TR-34 - Allow expired CRL.

When key wrapping method keyword specifies a wrapping method that is not the default method, the **TR-34 Key Receive – Allow wrapping override keywords** access control must be enabled.

Table 449. Valid TR-34 to CCA import translations and required access controls

Rule array keyword	Access control name	Offset (hex)	Specific key type and control vector attributes
"K0" and "K1": TR-31 key encryption or wrapping, or key block protection keys			
N/A	TR-34 Key Receive - Permit DES EXPORTER	0248	See Table 446 on page 1102.
	TR-34 Key Receive - Permit DES IMPORTER	0249	
N/A	TR-34 Key Receive - Permit AES EXPORTER	024A	See Table 447 on page 1102.
	TR-34 Key Receive - Permit AES IMPORTER	024B	
	TR-34 Key Receive - Permit AES EXPORTER with EXPTT31D	024C	
	TR-34 Key Receive - Permit AES IMPORTER with IMPTT31D	024D	

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 450. TR-34 Key Receive required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**		This service is not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	This service requires the July 2019 or later licensed internal code. Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). Rules CRLEXPCK, CRLEXPAL, and *input_token_length* greater than 3500 require CCA release 6.7.18 or higher. X9.143 key blocks are not supported.

Table 450. TR-34 Key Receive required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	This service is not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). Rules CRLEXPCK, CRLEXPAL, and *input_token_length* greater than 3500 require CCA release 6.7.18 or higher. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keywords WRAPENH2 and WRAPENH3 require the May 2021 or later licensed internal code (LIC). Rules CRLEXPCK, CRLEXPAL, and *input_token_length* greater than 3500 require CCA release 7.4.25 or higher. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

Chapter 12. Using digital signatures

This topic describes the PKA callable services that support using digital signatures to authenticate messages.

- "Digital Signature Generate (CSNDDSG and CSNFDSG)" on page 1110
- "Digital Signature Verify (CSNDDSV and CSNFDSV)" on page 1119

Signature algorithms and formatting methods

ICSF supports signature generation and verification for RSA, EC, and CRYSTALS-Dilithium algorithms. This topic lists the hashing algorithms supported by CSNDDSG and CSNDDSV that are either recommended or required by the standard for the algorithms and formatting methods. ICSF will only enforce the hashing algorithm when required by a standard.

ICSF supports these hash formatting methods for the RSA algorithm:

ANSI X9.31
Required hash methods: RIPEMD-160, SHA-1, SHA-256, SHA-384, or SHA-512.

ISO 9796-1
Recommended hash methods: Any.

RSA PKCS 1.0
Recommended hash methods: MD5, SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.

RSA PKCS 1.1
Recommended hash methods: MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, or SHA3-512.

RSA PKCS 1.0 and PKCS 1.1
Recommended hash methods: MD5, SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.

RSA PKCS-PSS
Required hash methods: SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.

Padding on the left with zeros
Recommended hash methods: Any.

ICSF supports these elliptic curve algorithms:

ANSI X9.62 ECDSA (Brainpool, NIST prime, and Koblitz)
Recommended hash methods: SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.

ISO/IEC 14888 Schnorr Digital Signature Algorithm (EC-SDSA)

- The hash method for P256 keys is SHA-256.
- The hash method for P521 keys is SHA-512.

Edwards-curve Digital Signature Algorithm (EdDSA)

- Required hash methods for Ed25519: SHA-512.
- Required hash methods for Ed448: SHAKE-256.

ICSF supports these hash formatting methods for the CRYSTALS-Dilithium algorithm:

CRYSTALS-Dilithium Digital Signature Algorithm (CRDL-DSA)
Required hash methods for CRYSTALS-Dilithium: SHAKE-256.

Digital Signature Generate (CSNDDSG and CSNFDSG)

Use the Digital Signature Generate callable service to generate a digital signature using a PKA private key or, for some limited functions, a secure PKCS #11 private key. Private keys must be valid for signature usage.

This service supports these hash formatting methods for the RSA algorithm:

- ANSI X9.31
- ISO 9796-1
- RSA DSI PKCS #1 v1.5 and v2.1
- Padding on the left with zeros

This service supports the ANSI X9.62 ECDSA (Brainpool, NIST prime, and Koblitz), the Edwards-curve Digital Signature Algorithm (EDDSA), the ISO/IEC 14888 Schnorr Digital Signature Algorithm (EC-SDSA), and the CRYSTALS-Dilithium Digital Signature Algorithm (CRDL-DSA) algorithms.

This service accepts either the input message or a hash of the input message.

For CCA keys, when the *private_key_identifier* parameter specifies:

An RSA private key
Select the method of formatting the text by using the digital signature formatting method rule array keyword.

An ECC private key
Select the ECDSA, EDDSA, or EC-SDSA signature algorithm rule array keyword.

For the EC-SDSA algorithm, the key is restricted to secp256r1 (P256) and secp521r1 (P521) curves. The hash method for P256 keys is SHA-256. The hash method for P521 keys is SHA-512.

A CRYSTALS-Dilithium private key
Select the CRDL-DSA signature algorithm rule array keyword.

For secure PKCS #11 keys, when the *private_key_identifier* parameter specifies:

An RSA private key
Select the PKCS-1.1 formatting method keyword.

An ECC private key
Select the ECDSA or EDDSA algorithm keyword.

If keyword ECDSA is specified in the rule array, the Elliptic Curve Digital Signature Algorithm is used as the digital-signature hash formatting method. If keyword EDDSA is specified, the EDDSA algorithm and hashing method appropriate for Edwards curves is used. Otherwise, specify the optional digital-signature hash formatting method keyword in the rule array for the method used to generate the RSA digital signature.

The callable service name for AMODE(64) invocation is CSNFDSG.

Format

```
CALL CSNDDSG(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              private_key_identifier_length,
              private_key_identifier,
              data_length,
              data,
              signature_field_length,
              signature_bit_length,
              signature_field)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value may be 0 1, 2, 3, or 4.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 451. Keywords for Digital Signature Generate Control Information

Keyword	Meaning
Signature algorithm (One, optional)	

Table 451. Keywords for Digital Signature Generate Control Information (continued)	
Keyword	**Meaning**
CRDL-DSA	Specifies to generate a signature in the CRYSTALS-Dilithium digital signature scheme.
ECDSA	Specifies to generate a digital signature using the ECDSA algorithm.
EC-SDSA	Specifies to generate a digital signature using the EC-SDSA algorithm.
EDDSA	Specifies to generate a digital signature using the EDDSA algorithm.
RSA	Specifies to generate an RSA digital signature. This is the default.
Digital Signature Formatting Method (optional, valid for RSA digital signature generation only)	
ISO-9796	Specifies to format the hash according to the ISO/IEC 9796-1 standard and generate the digital signature. Any hash method is allowed. This is the default.
PKCS-1.0	Specifies to format the digital signature on the string supplied in the hash variable as defined in the RSA PKCS #1 standard for block-type 00. The PKCS #1 v2.0 standard no longer defines this signature scheme.
PKCS-1.1	Specifies to format the digital signature on the string supplied in the hash variable as defined in the RSA PKCS #1 v2.0 standard for the RSASSA-PKCS1-v1_5 signature scheme. This was formerly known as block-type 01.
PKCS-PSS	Specifies to format the hash as defined in the RSA PKCS #1 v2.2 standard for the RSASSA-PSS signature scheme. Only valid for RSA-AESM and RSA-AESC key tokens.
X9.31	Specifies to format the hash according to the ANSI X9.31 standard. The modulus length of a key used must be one of 1024, 1280, 1536, 1792, 2048, or 4096 bits.
ZERO-PAD	Specifies to format the hash by padding it on the left with binary zeros to the length of the RSA key modulus. Any supported hash function is allowed.
Data Input Type (One, optional)	
HASH	Specifies that the *data* parameter contains the hash that is to be signed. This is the default.
MESSAGE	Specifies that the *data* parameter contains the message that is to be hashed and signed. This keyword is required with EDDSA, EC-SDSA, and CRDL-DSA keywords.
Hash Method Specification (One required:) • ***When the MESSAGE keyword is specified,*** • ***When the HASH keyword is specified and the hash formatting method is PKCS-PSS or X9.31, or*** • ***When the signature algorithm keyword EDDSA or CRDL-DSA is specified.***	

Table 451. Keywords for Digital Signature Generate Control Information (continued)	
Keyword	**Meaning**
ED-HASH	Process the text supplied in the data variable using the hash method for the Edwards curve of the key passed in the *PKA_private_key_identifier* parameter. For Ed25519, this is SHA-512. For Ed448, this is SHAKE-256. This keyword is required with the EDDSA keyword.
CRDLHASH	Process the text supplied in the message variable using the hash method appropriate for the CRYSTALS-Dilithium algorithm. This is SHAKE-256. Required for CRDL-DSA.
MD5	Process the text supplied in the *data* parameter using the MD5 hash method. Not valid with PKCS-PSS or X9.31.
RPMD-160	Process the text supplied in the *data* parameter using the RIPEMD-160 hash method. Not valid with PKCS-PSS.
SHA-1	Process the text supplied in the *data* parameter using the SHA-1 hash method.
SHA-224	Process the text supplied in the *data* parameter using the SHA-224 hash method. Not valid with X9.31.
SHA-256	Process the text supplied in the *data* parameter using the SHA-256 hash method.
SHA-384	Process the text supplied in the *data* parameter using the SHA-384 hash method.
SHA-512	Process the text supplied in the *data* parameter using the SHA-512 hash method.

private_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *private_key_identifier* parameter. If the *private_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 8000.

For secure PKCS #11 keys, the length must be 64.

private_key_identifier

Direction	Type
Input	String

The identifier of the private key to generate the signature. The key identifier is an RSA, EC, or CRYSTALS-Dilithium private key token, or label of such a private key identifier in key storage.

For CCA, this is an RSA, EC, or CRYSTALS-Dilithium private key or a retained RSA key. The token key usage flag must permit the generation of signatures. The format restriction field of the RSA private key token will restrict the format allowed to be used with the key.

If formatting method PKCS-PSS is specified, the RSA key token must have an AES object protection key (private key section identifiers X'30' and X'31'), which can be created with key type RSA-AESM and RSA-AESC in PKA Key Token Build (CSNDPKB and CSNFPKB).

For secure PKCS #11 keys, this is the 44-byte handle of the private key, prefixed with an EBCDIC equal sign character (‘=’ or X’7E’), and padded on the right with spaces for a total length of 64 bytes.

data_length

Direction	Type
Input	Integer

The length of the *data* parameter in bytes. See Usage Notes for details of the formatting methods and the length requirements.

When the data input type keyword is MESSAGE, the value is the length of the message to be hashed. The maximum value is 2147483647 bytes. When the signature algorithm keyword is EDDSA or EC-SDSA, the maximum value is 8192. When CRDL-DSA is specified, the maximum *data_length* for the data passed with a CRYSTALS-Dilithium (6,5) private key is 5000 bytes on Crypto Express7 CCA Coprocessor. The maximum *data_length* for the data passed with a CRYSTALS-Dilithium private key is 15000 bytes on a Crypto Express8 CCA or later coprocessor. A hash method rule array keyword must be specified.

When the data input type keyword is HASH, the value must be the exact length of the hash to be signed. The maximum value is 516 bytes. The length of the hash for the supported hashing method is 16 for MD5, 20 for SHA-1 and RPMD-160, 28 for SHA-224, 32 for SHA-256, 48 for SHA-384, and 64 for SHA-512. You can use either the One-Way Hash Generate callable service or the MDC Generate callable service to generate the hash.

For the PKCS-PSS formatting rule, the first four bytes must be the salt length. The remaining bytes of the parameter must contain the hash or message. The value will be 4 + length of the hash or message.

For the ZERO-PAD format rule, the length is restricted to 36 for RSA keys that permit key management in the key usage field. For RSA keys that permit signature only in the key usage field, the maximum value is 512. This hash length limit is controlled by an access control point. Only RSA keys that permit key management are affected by this access control point. See the restrictions section of this service for details.

data

Direction	Type
Input	String

The application-supplied data to be signed. The data can be the message to be hashed and signed or the hash of the message. See Usage Notes for details of the formatting methods and the data requirements.

The data must be in the caller's primary address space.

For the PKCS-PSS formatting rule, the first four bytes must be the salt length. The remaining bytes of the parameter must contain the hash or message.

signature_field_length

Direction	Type
Input/Output	Integer

The length in bytes of the *signature_field* to contain the generated digital signature. Upon return, this field contains the actual length of the generated signature. The maximum size for RSA is 512. The maximum size of ECC is 132. For CRYSTALS-Dilithium, the maximum size is 5000.

For RSA, this must be at least the byte length of the modulus rounded up to a multiple of 32 bytes for the X9.31 signature format or one byte for all other signature formats.

For ECDSA NIST prime curves, the maximum is 2 * 521 bits. For brain pool curves, the maximum size is 2 * 512 bits. For Koblitz secp256k1 curves, the maximum is 64 bytes.

signature_bit_length

Direction	Type
Output	Integer

The bit length of the digital signature generated. For ISO-9796, this is 1 less than the modulus length. For other RSA processing methods, this is the modulus length.

signature_field

Direction	Type
Input/Output	String

The digital signature generated is returned in this field. The digital signature is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the digital signature. This string is left-justified within the *signature_field*. Any unused bytes to the right are undefined.

Restrictions

Although ISO-9796 does not require the input hash to be an integral number of bytes in length, this service requires you to specify the *hash_length* in bytes.

For CCA RSA keys, the hash length limit is controlled by the **DSG ZERO-PAD unrestricted hash length** access control point. If enabled, the maximum hash length limit for ZERO-PAD is the modulus length of the PKA private key. If disabled, the maximum hash length limit for ZERO-PAD is 36 bytes. Only RSA key management keys are affected by this access control point. The limit for RSA signature use only keys is 512 bytes. This access control point is disabled in the domain role. You must have a TKE workstation to enable it.

Authorization

To use this service with a secure PKCS #11 private key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access) to the containing PKCS #11 token.

To use this service with a secure PKCS #11 private key that is a private object, the caller must have USER (READ) authority (user access) to the containing PKCS #11 token.

See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS, PKDS, or TKDS.

Notes on formatting the message:

ISO-9796
The length of the hash must be less than or equal to one-half of the number of bytes required to contain the modulus of the RSA key.

PKCS-1.0 and PKCS-1.1
The length of the hash must be 11 bytes shorter than the number of bytes required to contain the modulus of the RSA key.

When the HASH keyword is specified, the hash must be BER encoded. See “Formatting hashes and keys in public-key cryptography” on page 1650 for a description of the formatting methods.

PKCS-PSS
The first four bytes of the data parameter will contain the salt length. The remaining bytes of the parameter contains the hash or message.

It is recommended that the salt length be either 0 or the byte length of the hash algorithm selected. The salt length should not be less than the byte length of the hash algorithm, but it can be greater. When the **Digital Signature Generate – PKCS-PSS allow small salt** access control is enabled in the domain role, the salt length may be less than the length of the hash.

The size of the data to be signed is governed by the size of the RSA modulus. The modulus size and hash length affect the maximum salt length for a given key modulus size and specified hash. The maximum salt length equals modulus size/8 - hash length - 2. For example, with a 4096 bit modulus key and SHA-1, the maximum salt length becomes – (4096/8) – 20 -2 => 512-20-2 = 490.

X9.31
There are no restrictions for the hash length or message.

ZERO-PAD
The length of the hash must be less than or equal to the number of bytes required to contain the modulus of the RSA key.

ECDSA
There are no restrictions for the hash length or message.

EDDSA and EC-SDSA
The length of the message must be less than or equal to 8192 bytes.

CRDL-DSA
The length of the message must be less than or equal to 5000 bytes with a CRYSTALS-Dilithium (6,5) private key when on a Crypto Express7 CCA Coprocessor, or less than or equal to 15000 bytes with a CRYSTALS-Dilithium private key when on a Crypto Express8 and later CCA coprocessor.

The Digital Signature Generate callable service can take advantage of a PKCS#11 private key object stored in the TKDS, but you should check if using “PKCS #11 Private Key Sign (CSFPPKS and CSFPPKS6)” on page 1379 aligns better with the overall design of your application.

PKCS-PSS details

Before specifying PKCS-PSS, see section A.2.3 of RFC 8017: PKCS #1: RSA Cryptography Specifications Version 2.2 (RFC 8017 (tools.ietf.org/html/rfc8017)). Section A.2.3 defines a parameter field for RSASSA-PSS that has the following four parameters:

hashAlgorithm
This parameter identifies the hash function. A hashing-method specification keyword of SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512 is required. There is no default.

maskGenAlgorithm
This parameter identifies the mask generation function. MGF1 is a mask generation function based on a hash function and is the only function currently defined by the standard. For CCA, the hash function on which MGF1 is based is always the same as hashAlgorithm. Although this is not required, CCA enforces the recommendation by the standard that the underlying hash function be the same as the one identified by hashAlgorithm.

saltLength
This is the length of the salt, which is a randomly generated value. Use the data variable of the verb to prepend a 4-byte saltLength field in big endian format to the hash digest to be signed or the message to be hashed and signed.

trailerField
This is the trailer field number. This is not an input to the verb because it is always set to the value X'BC'. Other trailer field numbers are not supported by the standard.

Access control points

For PKA private keys, the **Digital Signature Generate** access control point controls the function of this service.

The length of the hash for ZERO-PAD is restricted to 36 bytes. If the **DSG ZERO-PAD unrestricted hash length** access control point is enabled in the domain role, the length of the hash is not restricted. This access control is disabled by default.

For the PKCS-PSS formatting method, ICSF requires the salt length specified in the data parameter to be zero or the length of the hash specified. When the **Digital Signature Generate – PKCS-PSS allow small salt** access control is enabled in the domain role, the salt length may be less than the length of the hash.

For secure PKCS #11 private keys, the Sign with private keys access control point controls the function of this service. For more information on the access control points of the Enterprise PKCS #11 coprocessor, see 'PKCS #11 Access Control Points' in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 452. Digital Signature Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	PKCS-PSS support requires the October 2016 or later licensed internal code (LIC). Keywords EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported. Compliant-tagged key tokens are not supported.
	Crypto Express5 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key. Compliant-tagged key tokens are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	PKCS-PSS support requires the October 2016 or later licensed internal code (LIC). Keywords EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported. Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Keywords EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported.
	Crypto Express5 Enterprise PKCS #11 coprocessor Crypto Express6 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key. Compliant-tagged key tokens are not supported.

Table 452. Digital Signature Generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	PKCS-PSS support requires the October 2016 or later licensed internal code (LIC). Keywords EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported. Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Keywords EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported.
	Crypto Express7 CCA Coprocessor	Keywords EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH require the June 2020 or later licensed internal code (LIC). ECC Koblitz curve secp256k1 requires the September 2020 or later licensed internal code (LIC). Keyword EC-SDSA requires the CCA release 7.4 or later licensed internal code (LIC). CRYSTALS-Dilithium (8,7) Round 2 or Round 3 and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported.
	Crypto Express5 Enterprise PKCS #11 coprocessor Crypto Express6 Enterprise PKCS #11 coprocessor Crypto Express7 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key. Compliant-tagged key tokens are not supported.
	CP Assist for Cryptographic Functions	Clear and secure ECC key requests for the Prime P-256, Prime P-384, Prime P-521, Ed25519, and Ed448 curves are supported. Secure key support requires a CEX7 CCA coprocessor with the June 2020 or later licensed internal code.

Table 452. Digital Signature Generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	Keywords EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported.
	Crypto Express7 CCA Coprocessor	CRYSTALS-Dilithium (8,7) Round 2 or Round 3 and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported.
	Crypto Express6 Enterprise PKCS #11 coprocessor Crypto Express7 Enterprise PKCS #11 coprocessor Crypto Express8 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key. Compliant-tagged key tokens are not supported.
	Crypto Express8 CCA Coprocessor	CRYSTALS-Dilithium (8,7) Round 2 or Round 3 and CRYSTALS-Dilithium (6,5) Round 3 require the CCA 8.0 or later licensed internal code (LIC).
	CP Assist for Cryptographic Functions	ECC requests for the Prime P-256, Prime P-384, Prime P-521 curves, Ed25519, and Ed448 are supported.

Digital Signature Verify (CSNDDSV and CSNFDSV)

Use the Digital Signature Verify callable service to verify a digital signature using a PKA public key.

- The Digital Signature Verify callable service can use the RSA, EC, or CRYSTALS-Dilithium public key, depending on the digital signature algorithm used to generate the signature.
- The Digital Signature Verify callable service can use the RSA public keys that are contained in trusted blocks regardless of whether the block also contains rules to govern its use when generating or exporting keys with the RKX service. If the TPK-ONLY keyword is used in the *rule_array* parameter, an error will occur if the *PKA_public_key_identifier* parameter does not contain a trusted block.
- The Digital Signature Verify callable service can use an X.509 certificate containing an RSA or ECC public key.

This service supports these hash formatting methods for RSA keys:

- ANSI X9.31
- ISO 9796
- RSA DSI PKCS #1 v2.0 and v2.2
- Padding on the left with zeros

This service supports the ANSI X9.62 ECDSA (Brainpool, NIST prime, and Koblitz), the Edwards-curve Digital Signature Algorithm (EDDSA), the ISO/IEC 14888 Schnorr Digital Signature Algorithm (EC-SDSA), and the CRYSTALS-Dilithium Digital Signature Algorithm (CRDL-DSA) algorithms.

This service accepts either the input message or a hash of the input message.

If keyword ECDSA is specified in the rule array, the Elliptic Curve Digital Signature Algorithm is used as the digital-signature hash formatting method. If keyword EDDSA is specified, the EDDSA algorithm

and hashing method appropriate for Edwards curves is used. If keyword CRDL-DSA is specified, the CRYSTALS-Dilithium algorithm and hashing method appropriate for CRYSTALS-Dilithium is used. Otherwise, specify the optional digital-signature hash formatting method keyword in the rule array for the method used to generate the RSA digital signature being verified.

The callable service name for AMODE(64) invocation is CSNFDSV.

Format

```
CALL CSNDDSV(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             PKA_public_key_identifier_length,
             PKA_public_key_identifier,
             data_length,
             data,
             signature_field_length,
             signature_field)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 0, 1, 2, 3, 4, 5, or 6.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 453. Keywords for Digital Signature Verify Control Information

Keyword	Meaning
Signature Algorithm (One, optional)	
CRDL-DSA	Specifies to verify a signature in the CRYSTALS-Dilithium digital signature scheme.
ECDSA	Specifies to generate a digital signature using the ECDSA algorithm.
EC-SDSA	Specifies to generate a digital signature using the EC-SDSA algorithm.
EDDSA	Specifies to generate a digital signature using the EDDSA algorithm.
RSA	Specifies to verify an RSA digital signature. This is the default.
Digital Signature Formatting Method (One, optional, RSA only)	
ISO-9796	Specifies to format the hash according to the ISO/IEC 9796-1 standard and generates the digital signature. Any hash method is allowed. This is the default.
PKCS-1.0	Specifies to format the digital signature on the string supplied in the hash variable as defined in the RSA PKCS #1 standard for block-type 00. The PKCS #1 v2.0 standard no longer defines this signature scheme.
PKCS-1.1	Specifies to format the digital signature on the string supplied in the hash variable as defined in the RSA PKCS #1 v2.0 standard for the RSASSA-PKCS1-v1_5 signature scheme. This was formerly known as block-type 01.
PKCS-PSS	Specifies to format the hash as defined in the RSA PKCS #1 v2.2 standard for the RSASSA-PSS signature scheme.
X9.31	Specifies to format the hash according to the ANSI X9.31 standard. The input text must have been previously hashed with any of the hash algorithms listed. The modulus length of a key used must be one of 1024, 1280, 1536, 1792, 2048, or 4096 bits.
ZERO-PAD	Specifies to format the hash by padding it on the left with binary zeros to the length of the RSA key modulus. Any hash method is allowed.
Data Input Type (One, optional)	
HASH	Specifies that the *data* parameter contains the hash that is to be signed. This is the default.

Table 453. Keywords for Digital Signature Verify Control Information (continued)

Keyword	Meaning
MESSAGE	Specifies that the *data* parameter contains the message that is to be hashed and signed. This keyword is required with EDDSA, EC-SDSA, and CRDL-DSA keywords.
Hash Method Specification (One required:) • ***When the MESSAGE keyword is specified,*** • ***When the HASH keyword is specified and the hash formatting method is PKCS-PSS or X9.31, or*** • ***When the signature algorithm keyword EDDSA or CRDL-DSA is specified.***	
ED-HASH	Process the text supplied in the data variable using the hash method for the Edwards curve of the key passed in the *PKA_private_key_identifier* parameter. For Ed25519, this is SHA-512. For Ed448, this is SHAKE-256. This keyword is required with the EDDSA keyword.
CRDLHASH	Process the text supplied in the message variable using the hash method appropriate for the CRYSTALS-Dilithium algorithm. This is SHAKE-256. Required for CRDL-DSA.
MD5	Process the text supplied in the *data* parameter using the MD5 hash method. Not valid with PKCS-PSS.
RPMD-160	Process the text supplied in the *data* parameter using the RIPEMD-160 hash method. Not valid with PKCS-PSS.
SHA-1	The input value supplied in the *data* parameter is generated using the SHA-1 hash method.
SHA-224	The input value supplied in the *data* parameter is generated using the SHA-224 hash method.
SHA-256	The input value supplied in the *data* parameter is generated using the SHA-256 hash method.
SHA-384	The input value supplied in the *data* parameter is generated using the SHA-384 hash method.
SHA-512	The input value supplied in the *data* parameter is generated using the SHA-512 hash method.
Signature checking rule (One optional). Valid only with the PKCS-PSS digital signature hash formatting method.	
EXMATCH	Specifies that the 4-byte salt length prepended to the data identified by the *data* parameter must be an exact match to the salt length derived from the signature. This is the default.
NEXMATCH	Specifies that the 4-byte salt length prepended to the data identified by the *data* parameter does not have to be an exact match to the salt length derived from the signature. **Note:** The derived salt length cannot be less than the prepended length.
Trusted public key restriction (One, optional). Not valid with ECDSA, EC-SDSA, EDDSA, or CRDL-DSA keywords. Valid only with trusted blocks.	

Table 453. Keywords for Digital Signature Verify Control Information (continued)	
Keyword	**Meaning**
TPK-ONLY	The *PKA_public_key_identifier* parameter must be a trusted block or the PKDS label of a trusted block that contains, at a minimum, two sections: • Trusted Block Information section 0x14, which is required for all trusted blocks and • Trusted Public Key section 0x11, which contains the trusted public key and usage rules that indicate whether the trusted public key can be used in digital signature operations.
Certificate validation method (One required when PKA_public_key_identifier is a certificate. Otherwise, must not be specified.)	
RFC-2459	Validate the certificate using the semantics of RFC-2459.
RFC-3280	Validate the certificate using the semantics of RFC-3280.
RFC-5280	Validate the certificate using the semantics of RFC-5280.
RFC-ANY	Attempt to validate the certificate by first using the semantics of RFC-2459, then the semantics of RFC-3280, and finally, the semantics of RFC-5280.
Public Key Infrastructure Usage (One optional when the input is an X.509 certificate. Otherwise, must not be specified.)	
PKI-CHK	Specifies that the X.509 certificate is to be validated against the trust chain of the PKI hosted in the adapter. This requires that the CA credentials have been installed into all coprocessors using the Trusted Key Entry workstation. This is the default.
PKI-NONE	Specifies that the X.509 certificate is not to be validated against the trust chain of the PKI hosted in the adapter. This is suitable if the certificate has been validated using host-based PKI services or if using a self-signed certificate.

PKA_public_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *public_key_identifier* parameter. If the *public_key_identifier* contains a label, the value must be 64. Otherwise, the value must be between the actual length of the token and 8000.

PKA_public_key_identifier

Direction	Type
Input	String

The identifier of the public key to verify the signature. The key identifier is an RSA, EC, or CRYSTALS-Dilithium public key token, an internal trusted block, an X.509 certificate, or the label of such a public key identifier in key storage.

When the TPK-ONLY keyword is specified, the identifier must be a trusted block.

When the PKCS-PSS keyword is specified, the identifier cannot be an X.509 certificate.

Certificates may be PEM-formatted EBCDIC text or DER-encoded. The certificate may either have no key usage attribute or it must have at least one of the following usages: digitalSignature, nonRepudiation, keyCertSign, or cRLSign.

For the EC-SDSA algorithm, the key is restricted to secp256r1 (P256) and secp521r1 (P521) curves. The hash method for P256 keys is SHA-256. The hash method for P521 keys is SHA-512.

data_length

Direction	Type
Input	Integer

The length of the *data* parameter in bytes. See Usage Notes for details of the formatting methods and the length requirements.

When the data input type keyword is MESSAGE, the value is the length of the message to be hashed. The maximum value is 2147483647 bytes. When the signature algorithm keyword is EDDSA or EC-SDSA, the maximum length is 8192. When CRDL-DSA is specified, the maximum *data_length* for the data passed with a CRYSTALS-Dilithium (6,5) private key is 5000 bytes on Crypto Express7 CCA Coprocessor. The maximum *data_length* for the data passed with a CRYSTALS-Dilithium private key is 15000 bytes on a Crypto Express8 and later CCA coprocessor. A hash method rule array keyword must be specified.

When the data input type keyword is HASH, the value must be the exact length of the hash to be signed. The maximum value is 516 bytes. The length of the hash for the supported hashing method is 16 for MD5, 20 for SHA-1 and RPMD-160, 28 for SHA-224, 32 for SHA-256, 48 for SHA-384, and 64 for SHA-512. You can use either the One-Way Hash Generate callable service or the MDC Generate callable service to generate the hash.

For the PKCS-PSS formatting rule, the first four bytes must be the salt length. The remaining bytes of the field must contain the hash or message. The value will be 4 + length of the hash or message.

data

Direction	Type
Input	String

The application-supplied text on which the supplied signature was generated. The data can be the message to be hashed and verified or the hash of the message. See Usage Notes for details of the formatting methods and the data requirements.

The data must be in the caller's primary address space.

For the PKCS-PSS formatting rule, the first four bytes must be the salt length. The remaining bytes of the parameter must contain the hash or message.

signature_field_length

Direction	Type
Input	Integer

The length in bytes of the *signature_field* parameter. The maximum size is 5000 bytes.

signature_field

Direction	Type
Input	String

This field contains the digital signature to verify. The digital signature is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the digital signature. This string is left-justified within the *signature_field*.

Restrictions

The exponent for RSA public keys must be odd.

RSA keys 512-bit to 2048-bit may have a public exponent of 3, 5, 17, 257, 65537, or random. Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC).

For 2049-bit to 4096-bit RSA keys:

- The public exponent may have a value of 3, 5, 17, 257, 65537, or random.
- Support for a random public exponent requires zEC12, zBC12, and later systems with a CEX4C or later coprocessor with September 2013 or later licensed internal code (LIC).
- Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Notes on formatting the message:

ISO-9796
The length of the hash must be less than or equal to one-half of the number of bytes required to contain the modulus of the RSA key.

PKCS-1.0 and PKCS-1.1
The length of the hash must be 11 bytes shorter than the number of bytes required to contain the modulus of the RSA key.

When the HASH keyword is specified, the hash must be BER encoded. See "Formatting hashes and keys in public-key cryptography" on page 1650 for a description of the formatting methods.

PKCS-PSS
The first four bytes of the data parameter will contain the salt length. The remaining bytes of the parameter contains the hash or message.

The hash algorithm and salt length should be provided to you with the signature. If not, the recommended salt length is 0 or the byte length of the hash algorithm.

For signature verification, the salt length derived from the signature must be an exact match (keyword EXMATCH, the default) for the salt length specified with the *data* parameter. When the **Digital Signature Verify – PKCS-PSS allow not exact salt length** access control is enabled in the domain role, keyword NEXMATCH can be specified to allow signature verification when the salt length derived from the signature is not an exact match for the salt length specified with the *data* parameter. Salt lengths derived from the signature are not allowed to be less than the value specified with the *data* parameter.

When the signature verification is done on an accelerator, there is no domain role and no access control points. The check of the salt length will be performed based on the signature check rule keywords.

X9.31
There are no restrictions for the hash length or message.

ZERO-PAD
The length of the hash must be less than or equal to the number of bytes required to contain the modulus of the RSA key.

ECDSA
There are no restrictions for the hash length or message.

EDDSA and EC-SDSA
The length of the message must be less than or equal to 8192 bytes.

CRDL-DSA
The length of the message must be less than or equal to 5000 bytes with a CRYSTALS-Dilithium (6,5) private key when on a Crypto Express7 CCA Coprocessor or less than or equal to 15000 bytes with a CRYSTALS-Dilithium private key when on a Crypto Express8 and later CCA coprocessor.

PKCS-PSS details

Before specifying PKCS-PSS, see section A.2.3 of RFC 8017: PKCS #1: RSA Cryptography Specifications Version 2.2 (RFC 8017 (tools.ietf.org/html/rfc8017)). Section A.2.3 defines a parameter field for RSASSA-PSS that has the following four parameters:

hashAlgorithm
This parameter identifies the hash function. A hashing-method specification keyword of SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512 is required. There is no default.

maskGenAlgorithm
This parameter identifies the mask generation function. MGF1 is a mask generation function based on a hash function and is the only function currently defined by the standard. For CCA, the hash function on which MGF1 is based is always the same as hashAlgorithm. Although this is not required, CCA enforces the recommendation by the standard that the underlying hash function be the same as the one identified by hashAlgorithm.

saltLength
This is the length of the salt, which is a randomly generated value. Use the data variable of the verb to prepend a 4-byte saltLength field in big endian format to the hash digest to be signed or the message to be hashed and signed.

trailerField
This is the trailer field number. This is not an input to the verb because it is always set to the value X'BC'. Other trailer field numbers are not supported by the standard.

Access control point

The **Digital Signature Verify** access control point controls the function of this service.

For the PKCS-PSS formatting method, the salt length derived from the signature must be an exact match for the salt length specified in the *data* parameter. When the **Digital Signature Verify – PKCS-PSS allow not exact salt length** access control is enabled in the domain role, the NEXMATCH keyword may be specified in the *rule_array* parameter. When the NEXMATCH keyword is specified, the salt length derived from the signature need not be an exact match for the salt length specified with the *data* parameter.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 454. Digital Signature Verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	PKCS-PSS support requires the October 2016 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, PKI-NONE, EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported.
	Crypto Express5 Accelerator	Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, PKI-NONE, EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC not supported. CRYSTALS-Dilithium not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported.

Table 454. Digital Signature Verify required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	PKCS-PSS support requires the October 2016 or later licensed internal code (LIC). Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, PKI-NONE, EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, and PKI-NONE require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Self-signed certificates require a CEX6C with the July 2019 or later licensed internal code (LIC). Keywords EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported.
	Crypto Express5 Accelerator Crypto Express6 Accelerator	Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, PKI-NONE, EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC not supported. CRYSTALS-Dilithium not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported.

Table 454. Digital Signature Verify required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, PKI-NONE, EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Keywords EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported.
	Crypto Express7 CCA Coprocessor	Keywords EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH require the June 2020 or later licensed internal code (LIC). ECC Koblitz curve secp256k1 requires the September 2020 or later licensed internal code (LIC). Keyword EC-SDSA requires the CCA release 7.4 or later licensed internal code (LIC). CRYSTALS-Dilithium (8,7) Round 2 or Round 3 and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported.
	Crypto Express5 Accelerator Crypto Express6 Accelerator Crypto Express7 Accelerator	Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, PKI-NONE, EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC not supported. CRYSTALS-Dilithium not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported.
	CP Assist for Cryptographic Functions	ECC requests for the Prime P-256, Prime P-384, Prime P-521 curves, Ed25519, and Ed448 are supported.

Table 454. Digital Signature Verify required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 Accelerator Crypto Express7 Accelerator Crypto Express8 Accelerator	Keywords RFC-2459, RFC-3280, RFC-5280, RFC-ANY, PKI-CHK, PKI-NONE, EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC not supported. CRYSTALS-Dilithium not supported. X.509 certificates are not supported. Compliant-tagged key tokens are not supported.
	Crypto Express6 CCA Coprocessor	Keywords EC-SDSA, EDDSA, ED-HASH, CRDL-DSA, and CRDLHASH are not supported. ECC Koblitz curve secp256k1 is not supported.
	Crypto Express7 CCA Coprocessor	CRYSTALS-Dilithium (8,7) Round 2 or Round 3 and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported.
	Crypto Express8 CCA Coprocessor	CRYSTALS-Dilithium (8,7) Round 2 or Round 3 and CRYSTALS-Dilithium (6,5) Round 3 require the CCA 8.0 or later licensed internal code (LIC).
	CP Assist for Cryptographic Functions	ECC requests for the Prime P-256, Prime P-384, Prime P-521 curves, Ed25519, and Ed448 are supported.

Chapter 13. Managing PKA cryptographic keys

This topic describes the callable services that generate and manage PKA keys.

- "PKA Key Generate (CSNDPKG and CSNFPKG)" on page 1131
- "PKA Key Import (CSNDPKI and CSNFPKI)" on page 1140
- "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 1147
- "PKA Key Token Change (CSNDKTC and CSNFKTC)" on page 1165
- "PKA Key Translate (CSNDPKT and CSNFPKT)" on page 1169
- "PKA Public Key Extract (CSNDPKX and CSNFPKX)" on page 1181
- "Public Infrastructure Certificate (CSNDPIC and CSNFPIC)" on page 1184
- "Retained Key Delete (CSNDRKD and CSNFRKD)" on page 1191
- "Retained Key List (CSNDRKL and CSNFRKL)" on page 1194

Note: PKA private key tokens require either the RSA or ECC master key to be active to use the key. Table 455 on page 1131 shows the CSNDPKB keywords and the required master key.

Table 455. CSNDPKB keywords and the required master key

CSNDPKB keywords	Object protection key algorithm	Required master key
RSA-CRT RSA-PRIV RSAMEVAR	DES	RSA
RSA-AESM RSA-AESC ECC-PAIR	AES	ECC

PKA Key Generate (CSNDPKG and CSNFPKG)

Use the PKA Key Generate callable service to generate RSA, ECC, CRYSTALS-Dilithium, or CRYSTALS-Kyber key pairs.

Input to the PKA Key Generate callable service is either a skeleton key token that has been built by the PKA key token build service or a valid internal RSA token. PKG will generate a key with the same modulus length and the same exponent. In the case of a valid internal ECC token, PKG will generate a key based on the curve type and size. For CRYSTALS-Dilithium and CRYSTALS-Kyber tokens, PKG will generate a key based on the algorithm identifier and algorithm parameter. Internal tokens with a X'09' section are not supported.

RSA key generation requires this information in the input skeleton token:

- Size of the modulus in bits. The modulus for modulus-exponent form keys is between 512 and 1024. The CRT modulus is between 512 and 4096. The modulus for the variable-length-modulus-exponent form is between 512 and 4096.

RSA key generation has these restrictions: For modulus-exponent, there are restrictions on modulus, public exponent, and private exponent. For CRT, there are restrictions on dp, dq, U, and public exponent. See the Key value structure in "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 1147 for a summary of restrictions.

ECC key generation requires this information in the skeleton token:

- The key type: ECC.

- The type of curve: NIST Prime, Brainpool, Edwards, or Koblitz.
- The size of P in bits:
  - 192, 224, 256, 384 or 521 for Prime curves.
  - 160, 192, 224, 256, 320, 384, or 512 for Brainpool curves.
  - 255 or 448 for Edwards curves.
  - 256 for Koblitz curves.
- Key usage information:
  - For Brainpool, Prime, and Koblitz curves, the key can be used for ECDSA and key agreement.
  - For Edwards curve, the key can be used for EdDSA only.
- Optionally, application associated data.

The generated ECC private key will be returned in one of the following forms:

- Clear key.
- Encrypted key enciphered under the ECC master key.
- Encrypted key enciphered by an AES transport key.

CRYSTALS-Dilithium key generation requires the following information in the skeleton token:

- The Algorithm identifier:
  - X'01' CRYSTALS-Dilithium Round 2.
  - X'03' CRYSTALS-Dilithium Round 3.
- The Algorithm parameter:
  - X'0605' CRYSTALS-Dilithium (6,5) Round 2 or Round 3.
  - X'0807' CRYSTALS-Dilithium (8,7) Round 2 or Round 3.
- PKA Key usage information:
  - For CRYSTALS-Dilithium, U-DIGSIG is required.

The generated CRYSTALS-Dilithium private key will be returned in one of the following forms:

- Clear key.
- Encrypted key enciphered under the ECC master key.

CRYSTALS-Kyber key generation requires the following information in the skeleton token:

- The Algorithm identifier:
  - X'02' CRYSTALS-Kyber Round 2.
- The Algorithm parameter:
  - X'1024' CRYSTALS-Kyber (1024) Round 2.
- PKA Key usage information:
  - For CRYSTALS-Kyber, at least one of U-KEYENC or U-DATENC is required.

The generated CRYSTALS-Kyber private key will be returned in one of the following forms:

- Clear key.
- Encrypted key enciphered under the ECC master key.

The callable service name for AMODE(64) invocation is CSNFPKG.

Format

```
CALL CSNDPKG(
             return_code,
             reason_code,
```

```
                exit_data_length,
                exit_data,
                rule_array_count,
                rule_array,
                regeneration_data_length,
                regeneration_data,
                skeleton_key_identifier_length,
                skeleton_key_identifier,
                transport_key_identifier,
                generated_key_token_length,
                generated_key_token)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. Value may be 1 or 2.

rule_array

Direction	Type
Input	String

A keyword that provides control information to the callable service. See Table 456 on page 1134 for a list. A keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 456. Keywords for PKA Key Generate Rule Array

Keyword	Meaning
Private Key Encryption (required)	
CLEAR	Return the private key in clear text. The private key in clear text is an external token.
MASTER	Encipher the private key under the master key. The keyword is not supported if a skeleton token with a X'09' section is provided.
RETAIN	Retain the private key within a cryptographic coprocessor for additional security. This is only valid for RSA signature keys. Because of this, the RETAIN keyword is not supported for: • A skeleton token with a X'09', X'30', or X'31' section provided. • An ECC token. • A CRYSTALS-Dilithium token. • A CRYSTALS-Kyber token.
XPORT	Encipher the private key under the *transport_key_identifier*. This keyword is not supported for CRYSTALS-Dilithium or CRYSTALS-Kyber tokens.
Options (optional)	
CLONE	Mark a generated and retained private key as usable in cryptographic engine cloning process. This keyword is supported only if RETAIN is also specified. Only valid for RSA keys. The keyword is not supported for: • A skeleton token with a X'09' section is provided. • An ECC token. • A CRYSTALS-Dilithium token. • A CRYSTALS-Kyber token.
Processing Controls (Optional when regeneration_data_length is non-zero)	
ITER-38	When *regeneration_data* is specified, this keyword will cause the service to generate key values that are FIPS and ANSI X9.31 compliant.
Transport Key Type (one optional)	
OKEK-DES	The transport key identifier identifies a DES KEK token. This is the default value.
OKEK-AES	The transport key identifier identifies an AES KEK token.

regeneration_data_length

Direction	Type
Input	Integer

The length of the *regeneration_data* in bytes. The value must be 0 for ECC, CRYSTALS-Dilithium, and CRYSTALS-Kyber tokens. For RSA tokens, the value may be zero or between 8 and 512 bytes inclusive.

regeneration_data

Direction	Type
Input	String

This field points to a string variable containing a string used as the basis for creating a particular public-private key pair in a repeatable manner.

skeleton_key_identifier_length

Direction	Type
Input	Integer

The length of the *skeleton_key_identifier* parameter in bytes. The maximum allowed value is 8000 bytes.

skeleton_key_identifier

Direction	Type
Input	String

The application-supplied skeleton key token generated by PKA key token build or label of the token that contains the required key attributes for PKA key pair generation. If RETAIN was specified and the *skeleton_key_identifier* is a label, the label must match the *private-key name* of the key.

For RSA keys, the *skeleton_key_identifier* parameter must contain a token which specifies a modulus length in the range 512 – 4096 bits.

To generate a compliant-tagged key token, a compliant-tagged skeleton token must be supplied.

transport_key_identifier

Direction	Type
Input	String

The identifier of the key-encrypting key to wrap the generated key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

When the XPORT rule is not specified, this parameter must be 64 bytes of binary zeros.

When the XPORT rule is specified, this parameter is a:

- 64-byte label of a CKDS record that contains the transport key.
- CCA 64-byte DES internal key token containing the transport key.
- CCA variable-length AES internal key token containing the transport key.
- X9.143 (TR-31) variable-length AES or DES internal key block containing the transport key (CCA release 8.1 or later).

When the identifier is a label of a CCA key token, the label must resolve uniquely to either an IMPORTER or EXPORTER key.

- When the generated key is an external ECC key, this must be an AES key-encrypting key in a CCA variable-length key token or X9.143 key block (key usage K0, algorithm A and mode of use E or D).
- When the generated key is an external RSA key with private section id X'30' or X'31', this must be an AES key-encrypting key in a CCA variable-length key token or X9.143 key block (key usage K0, algorithm A and mode of use E or D).
- When the generated key is an external RSA key with other private section ids, this must be a DES key-encrypting key in a CCA fixed-length key token or a variable-length X9.143 key block (key usage K0, algorithm T and mode of use E or D).

If the token or key block supplied was encrypted under the old master key, the token or key block will be returned encrypted under the current master key

generated_key_token_length

Direction	Type
Input/Output	Integer

The length of the generated key token or label for the generated key token. The field is checked to ensure it is at least equal to the token being returned. The maximum size is 8000 bytes. The length must be specified as 64 when a label is provided. On output, this field is updated with the actual token length.

generated_key_token

Direction	Type
Input/Output	String

The internal token or label of the generated key. The label can be that of a retained key for most RSA key tokens.

Checks are made to ensure that:

- An ECC token in the PKDS will only be overlaid if an ECC token is specified in the *skeleton_key_identifier*
- A CRYSTALS-Dilithium token in the PKDS will only be overlaid if a CRYSTALS-Dilithium token is specified in the *skeleton_key_identifier*.
- A CRYSTALS-Kyber token in the PKDS will only be overlaid if a CRYSTALS-Kyber token is specified in the *skeleton_key_identifier*.
- A retained key cannot be overlaid in PKDS. If the label is that of a retained key, the private name in the token must match the label name. If a label is specified in the *generated_key_token* field, the *generated_key_token_length* returned to the application will be the same as the input length. If RETAIN was specified, but the *generated_key_token* was not specified as a label, the generated key length returned to the application will be zero (the key was retained in the cryptographic coprocessor). If the record already exists in the PKDS with the same label as the one specified as the *generated_key_token*, the record will be overwritten with the newly generated key token (unless the PKDS record is an existing retained private key, in which case it cannot be overwritten). If there is no existing PKDS record with this label in the case of generating a retained key, a record will be created.
- For generation of a non-retained key, if a label is specified in the *generated_key_token* field, a record must already exist in the PKDS with this same label or the service will fail.

Restrictions

RSA keys 512-bit to 2048-bit may have a public exponent of 3, 5, 17, 257, 65537, or random. Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC).

For 2049-bit to 4096-bit RSA keys:

- The public exponent may have a value of 3, 5, 17, 257, 65537, or random.
- Support for a random public exponent requires zEC12, zBC12, and later systems with a CEX4C or later coprocessor with September 2013 or later licensed internal code (LIC).
- Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC).

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Access control points

The **PKA Key Generate** access control point controls the function of this service. Additional access control points control the use of rule array keys.

Table 457. Required access control points for PKA Key Generate rule array keys

Key algorithm	Rule array keyword	Access control point
RSA	CLEAR	PKA Key Generate – Clear RSA keys
ECC	CLEAR	PKA Key Generate – Clear ECC keys
RSA	CLONE	PKA Key Generate - Clone
CRYSTALS-Dilithium	CLEAR	PKA Key Generate – Clear CRYSTALS-Dilithium keys
CRYSTALS-Kyber	CLEAR	PKA Key Generate – Clear CRYSTALS-Kyber keys

To generate keys based on the value supplied in the *regeneration_data* variable, you must enable at least one of these access control points:

- When not using the RETAIN keyword, **PKA Key Generate - Permit Regeneration Data**
- When using the RETAIN keyword, **PKA Key Generate - Permit Regeneration Data Retain**

For ECC keys, when an transport key is specified, the **Prohibit weak wrapping - Transport keys** access control point can be enabled in the active role to prevent stronger keys from being wrapped by weaker keys.

The **Allow weak DES wrap of RSA** access control allows a weaker DES key-encrypting key to wrap an RSA private key token. The **Prohibit weak wrap – Transport keys** access control must be enabled and this access control will override the restriction.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 458. PKA Key Generate required hardware

Server	Required Cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC). CRYSTALS-Dilithium keys not supported. Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC). CRYSTALS-Dilithium keys not supported. Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. Compliant-tagged key tokens are not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). CRYSTALS-Dilithium keys not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported. X9.143 key blocks are not supported.

Table 458. PKA Key Generate required hardware (continued)

Server	Required Cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Dilithium keys not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	CRYSTALS-Dilithium keys not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	CRYSTALS-Dilithium keys require the June 2020 or later licensed internal code (LIC). Edwards curves Ed25519 and Ed448 require the June 2020 or later licensed internal code (LIC). ECC Koblitz curve secp256k1 requires the September 2020 or later licensed internal code (LIC). CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported. X9.143 key blocks are not supported.

Table 458. PKA Key Generate required hardware (continued)

Server	Required Cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	CRYSTALS-Dilithium and CRYSTALS-Kyber keys not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys are not supported. X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys require CCA release 8.0 or later licensed internal code (LIC). X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

PKA Key Import (CSNDPKI and CSNFPKI)

Use this service to import an RSA, ECC, or QSA public-private key pair. A private key must be accompanied by the associated public key. The source PKA private-key either can be in the clear or it can be enciphered.

To import a PKA public-private key pair:

- Identify the external RSA, ECC, or QSA key token to be imported using the *source_key_identifier* parameter. Use the PKA Key Generate service to obtain the token or, if the key originates in a non-CCA system, use the PKA Key Token Build service.
- Use the *transport_key_identifier* parameter to identify the key used to encipher the source key token:
  - For an enciphered ECC key token, provide an AES key-encrypting key in a variable-length symmetric key-token.
  - For an enciphered RSA key token, provide a key-encrypting key that can be used for import, either in a fixed-length DES key-token or a variable-length AES key token. The transport key must have its key usage set to allow IMPORT. A variable-length key must also allow wrapping of an RSA key.
  - For a clear key token, provide a null transport key token.

If an RSA source key token that does not have an AES-enciphered Object Protection Key (OPK) contains an enciphered private key or its OPK data, the target key is protected by the RSA master-key. When an RSA private-key token contains a private-key section with a section identifier of X'30' or X'31', its AES OPK is used to encipher the private key and the ECC master key is used, in turn, to AES-encipher the OPK and its data. Likewise, if an ECC key token contains an enciphered private key, its AES OPK is used to encipher the private key and the ECC master key is used to encipher the OPK and its data.

The result is returned in an internal PKA key-token identified by the *target_key_identifier* parameter. The verb also updates the *target_key_identifier_length* variable to the length of data returned in the *target_key_identifier* variable.

Special note for CRT keys: ICSF always generates an RSA CRT key with p > q. If you import a CRT key from another RSA implementation with q > p, the key is usable within the coprocessor or accelerator, but your application encounters a performance degradation with each use of the key.

This service can also be used to import an active external trusted block. A trusted block does not contain a private key. Instead, it contains an encrypted confounder and triple-length MAC key. The MAC key is used to calculate an ISO 16609 CBC-mode Triple-DES MAC of the trusted block contents. In an external trusted block, the MAC key is encrypted under a DES IMP-PKA key.

To import an external trusted block so that it can be used as input by the Remote_Key_Export verb:

- Identify the active external trusted block to be imported using the *source_key_identifer* parameter. Use the Trusted Block Create service with the ACTIVATE rule-array keyword to obtain the token.
- Use the *transport_key_identifier* parameter to identify the operational DES IMP-PKA key used to encipher the confounder and triple-length MAC key contained within the trusted block. This is an IMPORTER key-encrypting key with only its IMPORT key-usage bit on (CV bit 21 = B'1').

The confounder and MAC key are enciphered under the RSA master key and returned along with the updated MAC value in an internal trusted block identified by the *target_key_identifier* parameter. The service also updates the *target_key_identifier_length* variable to the length of the returned token or key label.

The callable service name for AMODE(64) invocation is CSNFPKI.

Format

```
CALL CSNDPKI(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              source_key_identifier_length,
              source_key_identifier,
              importer_key_identifier,
              target_key_identifier_length,
              target_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This may be 0, 1, or 2.

rule_array

Direction	Type
Input	Character String

The *rule_array* parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The rule_array keywords are:

Table 459. Keywords for PKA Key Import

Keyword	Meaning
Token Type (optional)	
RSA	Specifies that the key token contains an RSA key. This is the default.
ECC	Specifies that the key token contains an ECC key.
QSA	Specifies that the key token contains a QSA key. Only clear keys are supported.
Transport key type (optional)	
IKEK-AES	The *importer_key_identifier* is a AES key.
IKEK-DES	The *importer_key_identifier* is a DES key. This is the default.

source_key_identifier_length

Direction	Type
Input	Integer

The length of the *source_key_identifier* parameter in bytes. The maximum size is 8000 bytes.

source_key_identifier

Direction	Type
Input	String

The external key token of an ECC, RSA, or QSA private key or active external trusted block to be imported. The ECC, RSA, and QSA key token must contain both public-key and private key information. The private key can be in cleartext or it can be enciphered.

importer_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-encrypting key to unwrap the source key. The key identifier is a variable-length operational key token or key block or the label of an operational token or block in key storage.

For RSA keys, this is either a DES limited authority transport key (IMP-PKA) or an AES transport key. For trusted blocks, this must be a DES limited authority transport key (IMP-PKA). For ECC keys, this must be an AES transport key. For QSA keys, this must be a null token.

This parameter contains one of the following:

- 64-byte null token when the source key is a QSA key.
- 64-byte label of a CKDS record that contains the transport key.
- 64-byte DES internal key token containing the transport key.
- a variable-length AES internal key token containing the transport key.
- a variable-length X9.143 (TR-31) key block containing the transport key: key usage K0, algorithm A or T, and mode of use D.

This parameter is ignored for clear tokens.

If the token or key block supplied was encrypted under the old master key, the token or key block will be returned encrypted under the current master key.

target_key_identifier_length

Direction	Type
Input/Output	Integer

The length of the *target_key_identifier* parameter in bytes. The maximum value is 8000 bytes.

On input, this is the size of the buffer to receive the output key token. If the *target_key_identifier* is the label of an existing record in the PKDS, the value must be 64.

On output, and if the size is of sufficient length, the parameter is updated with the actual length of the returned key token.

target_key_identifier

Direction	Type
Input/Output	String

This parameter contains the internal token or label of the imported PKA private key or a Trusted Block. If a label is specified on input, a PKDS record with this label must exist. The PKDS record with this

label will be overwritten with imported key unless the existing record is a retained key. If the record is a retained key, the import will fail. A retained key record cannot be overwritten.

When the *importer_key_identifier* is compliant-tagged, the key is imported as a compliant-tagged key token.

Restrictions

This service imports RSA keys of up to 4096 bits. However, the hardware configuration sets the limits on the modulus size of keys for digital signatures and key management; thus, the key may be successfully imported but fail when used if the limits are exceeded.

The *importer_key_identifier* is a limited-authority key-encrypting key.

CRT form tokens with a private section ID of X'05' cannot be imported into ICSF.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

An RSA modulus-exponent form token imported results in a X'06' format.

This service imports keys of any modulus size up to 4096 bits. However, the hardware configuration sets the limits on the modulus size of keys for digital signatures and key management; thus, the key may be successfully imported but fail when used if the limits are exceeded.

Access control points

The **PKA Key Import** access control point controls the function of this service. If the *source_key_token* parameter points to a trusted block, the **PKA Key Import - Import an external trusted block** access control point must also be enabled.

To prevent the importing of key tokens which have the private key values in the clear, the **PKA Key Import – Disallow clear key import** must be enabled.

The **Allow weak DES wrap of RSA** access control allows a weaker DES key-encrypting key to wrap an RSA private key token. The **Prohibit weak wrap – Transport keys** access control must be enabled and this access control will override the restriction.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 460. PKA Key Import required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC). Triple-length DES keys require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. CRYSTALS-Dilithium keys are not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 Private keys are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC). Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens are not supported. CRYSTALS-Dilithium keys are not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 Private keys are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys require the December 2018 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). CRYSTALS-Dilithium keys are not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 Private keys are not supported. X9.143 key blocks are not supported.

Table 460. PKA Key Import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	Compliant-tagged key tokens are not supported. CRYSTALS-Dilithium keys are not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 Private keys are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	CRYSTALS-Dilithium keys are not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 Private keys are not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	CRYSTALS-Dilithium key support require the June 2020 or later licensed internal code (LIC). Edwards curves Ed25519 and Ed448 require the June 2020 or later licensed internal code (LIC). ECC Koblitz curve secp256k1 requires the September 2020 or later licensed internal code (LIC). CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 Private keys are not supported. X9.143 key blocks are not supported.

Table 460. PKA Key Import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	CRYSTALS-Dilithium keys are not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 Private keys are not supported. X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys require CCA release 8.0 or later licensed internal code (LIC). X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

PKA Key Token Build (CSNDPKB and CSNFPKB)

This callable service can be used to create PKA key tokens. Specifically, it can be used to:

- Build external PKA key tokens containing unencrypted private key for ECC or RSA keys. You can use this token as input to the PKA Key Import service to obtain an operational internal token containing an enciphered private key.
- Build external RSA key tokens with the private key for use with the PKA Key Translate service.
- Build a skeleton token for ECC, RSA, CRYSTALS-Dilithium, or CRYSTALS-Kyber keys that you can use as input to the PKA Key Generate service.
- Build a public key token containing a clear unencrypted public key for an ECC or RSA keys and return the public key in a token format that other PKA services can use directly.

ECC key generation requires this information in the skeleton token:

- The key type: ECC.
- The type of curve: NIST Prime, Brainpool, Edwards, or Koblitz.
- The size of P in bits:
  - 192, 224, 256, 384, or 521 for NIST Prime curves.
  - 160, 192, 224, 256, 320, 384, or 521 for Brainpool curves.
  - 255 or 448 for Edwards curves.
  - 256 for Koblitz curves.
- Key usage information:
  - For Edwards curves, SIG-ONLY is required. Edwards curve keys are used with EdDSA only.
- Optionally, application associated data.
- Optionally, a key-derivation section.

RSA key generation requires this following information in the skeleton token:

- In modulus-exponent form:

  - The length of the modulus n in bits (512-4096).
  - The length of the public exponent e (optional).
  - The length of the private exponent d (optional).
  - The public exponent e (optional).
- In Chinese Remainder Theorem form:
  - The length of the modulus n in bits (512-4096).
  - The length of the public exponent e (optional).
  - The public exponent e (optional).
  - Other optional lengths.
- There are restrictions on the value and length of the RSA public exponent:
  - RSA keys 512-bit to 2048-bit may have a public exponent of 3, 5, 17, 257, 65537, or random. Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC).
  - For 2049-bit to 4096-bit RSA keys:
    - The public exponent may have a value of 3, 5, 17, 257, 65537, or random.
    - Support for a random public exponent requires zEC12, zBC12, and later systems with a CEX4C or later coprocessor with September 2013 or later licensed internal code (LIC).
    - Support for RSA public exponents 5, 17, and 257 requires the October 2016 or later licensed internal code (LIC).

CRYSTALS-Dilithium key generation requires the following information in the skeleton token:

- The Algorithm identifier:
  - X'01' CRYSTALS-Dilithium Round 2.
  - X'03' CRYSTALS-Dilithium Round 3.
- The Algorithm parameter:
  - X'0605' CRYSTALS-Dilithium (6,5) Round 2 or Round 3.
  - X'0807' CRYSTALS-Dilithium (8,7) Round 2 or Round 3.
- PKA Key usage information:
  - For CRYSTALS-Dilithium, U-DIGSIG is required.

CRYSTALS-Kyber key generation requires the following information in the skeleton token:

- The Algorithm identifier:
  - X'02' CRYSTALS-Kyber Round 2.
- The Algorithm parameter:
  - X'1024' CRYSTALS-Kyber (1024) Round 2.
- PKA Key usage information:
  - For CRYSTALS-Kyber, at least one of U-KEYENC or U-DATENC is required.

The usage limits for a private key:

- If a private-key can be allowed to be used for symmetric key management, and the key can also be used to create digital signatures, include the KEY-MGMT keyword in the rule array.
- If a private key cannot be used in digital signature generation, include the KM-ONLY keyword in the rule array.
- If a private-key cannot be used for symmetric key management, you can include the SIG-ONLY keyword in the rule array. This is the default.

- A new class of keywords has been added which will allow the user to restrict the formatting method to one particular method. These restriction keywords are only valid with the RSA-AESM and RSA-AESC key tokens.

Special note for CRT keys: ICSF always generates a CRT key with p > q. If you import a CRT key from another RSA implementation with q > p, the key is usable within the coprocessor or accelerator, but your application encounters a performance degradation with each use of the key.

The callable service name for AMODE(64) invocation is CSNFPKB.

Format

```
CALL CSNDPKB(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_value_structure_length,
             key_value_structure,
             private_key_name_length,
             private_key_name,
             user_definable_associated_data_length,
             user_definable_associated_data,
             key_derivation_data_length,
             key_derivation_data,
             reserved_3_length,
             reserved_3,
             reserved_4_length,
             reserved_4,
             reserved_5_length,
             reserved_5,
             key_token_length,
             key_token)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. Value must be 1 - 12.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. Table 461 on page 1150 lists the keywords. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 461. Keywords for PKA Key Token Build Control Information

Keyword	Meaning
Key Type (required)	
RSA-AESC	This keyword is for creating a key token for an RSA public and private key in Chinese-Remainder Theorem format. The object protection key is an AES key.
RSA-AESM	This keyword is for creating a key token for an RSA public and private key in modulus-exponent format. The object protection key is an AES key.
RSA-CRT	This keyword indicates building a token containing both public and private RSA key information in the optimized Chinese Remainder Theorem (CRT) form. The parameter *key_value_structure* identifies the input key values.
RSA-PRIV	This keyword indicates building a token containing both public and private RSA key information. The parameter *key_value_structure* identifies the input key values.
RSA-PUBL	This keyword indicates building a token containing public RSA key information. The parameter *key_value_structure* identifies the input values.
RSAAESC2	This keyword is for creating a key token for an RSA public and private key in Chinese-Remainder Theorem format with a version X'05' private key associated data section. The object protection key is an AES key. Either RSAAESC2 or RSAAESM2 is required when using the COMP-TAG or the PKA Key Usage Control group.

Table 461. Keywords for PKA Key Token Build Control Information (continued)	
Keyword	**Meaning**
RSAAESM2	This keyword is for creating a key token for an RSA public and private key in modulus-exponent format with a version X'04' private key associated data section. The object protection key is an AES key. Either RSAAESC2 or RSAAESM2 is required when using the COMP-TAG or the PKA Key Usage Control group.
RSAMEVAR	This keyword is for creating a key token for an RSA public and private key pair in modulus-exponent form whose modulus is 512 bits or greater.
ECC-PAIR	This keyword indicates building a token containing both public and private ECC key information. The parameter *key_value_structure* identifies the input key values.
ECC-PUBL	This keyword indicates building a token containing public ECC key information. The parameter *key_value_structure* identifies the input values.
QSA-PAIR	This keyword indicates building a token containing both public and private QSA key information. The parameter *key_value_structure* identifies the input values. This includes CRYSTALS-Dilithium and CRYSTALS-Kyber key pairs.
QSA-PUBL	This keyword indicates building a token containing public QSA key information. The parameter *key_value_structure* identifies the input values. This includes CRYSTALS-Dilithium and CRYSTALS-Kyber public keys.
Compliance (Optional)	
COMP-TAG	Build a compliant-tagged key token. Only valid with RSAAESC2 or RSAAESM2.
NOCMPTAG	Do not build a compliant-tagged key token. This is the default.
Key Usage Control (One, optional) Not valid with type ECC-PUBL, RSAAESC2, RSAAESM2, RSA-PUBL, QSA-PAIR, or QSA-PUBL.	
KEY-MGMT	Indicates that a private key can be used in both the symmetric key distribution (RSA) and key agreement (EC) and the digital signature callable services.
KM-ONLY	Indicates that a private key can be used only in symmetric key distribution (RSA) and key agreement (EC).
SIG-ONLY	Indicates that a private key cannot be used in symmetric key distribution. This is the default. This keyword is required when building type ECC-PAIR token with an Edwards curve.
Format restriction (One, optional). Only valid with token type RSA-AESC, RSA-AESM, RSAAESC2, or RSAAESM2.	
FR-NONE	Specifies to not restrict the private key to be used to a particular method. The key is usable for any method. This is the default.
FR-I9796	Specifies to render the private key usable with the ISO-9796 digital-signature hash formatting method.

Table 461. Keywords for PKA Key Token Build Control Information (continued)	
Keyword	**Meaning**
FR-PK10	Specifies to render the private key usable with the PKCS-1.0 digital-signature hash formatting method.
FR-PK11	Specifies to render the private key usable with the PKCS-1.1 digital-signature hash formatting method.
FR-PSS	Specifies to render the private key usable with the PKCS-PSS digital-signature hash formatting method.
FR-X9.31	Specifies to render the private key usable with the X9.31 digital-signature hash formatting method.
FR-ZPAD	Specifies to render the private key usable with the ZERO-PAD digital-signature hash formatting method.
PKA Key Usage Control (Required with RSAAESC2, RSAAESM2, QSA-PAIR, or QSA-PUBL. Not valid otherwise.)	
U-DIGSIG	Digital Signature usage is allowed. When COMP-TAG is specified, cannot be combined with U-KEYENC, U-DATENC, or U-KEYAGR.
U-NONRPD	Non-Repudiation usage is allowed. When COMP-TAG is specified, cannot be combined with U-KEYENC, U-DATENC, or U-KEYAGR.
U-KCRTSN	keyCertSign usage is allowed. When COMP-TAG is specified, cannot be combined with U-KEYENC, U-DATENC, or U-KEYAGR.
U-CRLSN	Certificate Revocation List Sign usage is allowed. When COMP-TAG is specified, cannot be combined with U-KEYENC, U-DATENC, or U-KEYAGR.
U-KEYENC	Key Encipherment usage is allowed. When COMP-TAG is specified, cannot be combined with any other keyword from this group.
U-DATENC	Data Encipherment usage is allowed. When COMP-TAG is specified, cannot be combined with any other keyword from this group.
U-KEYAGR	Key agreement usage is allowed. When COMP-TAG is specified, cannot be combined with any other keyword from this group.
Key Agreement Control (One, Optional). Only valid with U-KEYAGR.	
U-ENCONL	Only encipher operations are allowed during key agreement key establishment protocols.
U-DECONL	Only decipher operations are allowed during key agreement key establishment protocols.
Translate Control (optional, only allowed with key types ECC-PAIR, RSA-AESC, RSA-AESM, RSAAESC2, RSAAESM2, RSAMEVAR, RSA-CRT, and RSA-PRIV and is valid with all key usage rules.)	
XLATE-OK	Specifies that the private key material can be translated.
NO-XLATE	Indicates key translation is not allowed. This is the default.
ECC token version (one, optional. Only valid with token type ECC-PAIR.)	
ECC-VER0	Creates an ECC public and private key-pair containing a version 0 private key section. This is the default.
ECC-VER1	Creates an ECC public and private key-pair containing a version 1 private key section.

Table 461. Keywords for PKA Key Token Build Control Information (continued)	
Keyword	**Meaning**
CPACF export control (one, optional). Valid with ECC-PAIR only. Not valid with ECC Koblitz curves.	
NOEXCPAC	Prohibit export to CPACF protected key format. This is the default.
XPRTCPAC	Allow export to CPACF protected key format. Valid for NIST P256, NIST P384, NIST P521, Ed25519, and Ed448 curves. Not valid for Koblitz curves.
AES export (one, optional). Only valid with ECC-PAIR or QSA-PAIR.	
AES1ECOK	Enable export of the ECC, CRYSTALS-Dilithium, or CRYSTALS-Kyber private key under an AES key encrypting key of similar strength or stronger.
NOAES1EC	Do not allow export of the ECC or QSA private key. This is the default.

key_value_structure_length

Direction	Type
Input	Integer

This is a segment of contiguous storage containing a variable number of input clear key values. The length depends on the key type parameter in the rule array and on the actual values input. The length is in bytes.

Table 462. Key Value Structure Length Maximum Values for Key Types	
Key Type	**Key Value Structure Maximum Value**
RSA-CRT	3500
RSAMEVAR	3500
RSA-AESC	3500
RSA-AESM	3500
RSAAESC2	3500
RSAAESM2	3500
RSA-PRIV	648
RSA-PUBL	1032
ECC-PAIR	207
ECC-PUBL	139
QSA-PAIR	7480
QSA-PUBL	2600

key_value_structure

Direction	Type
Input	String

This is a segment of contiguous storage containing a variable number of input clear key values and the lengths of these values in bits or bytes, as specified. The structure elements are ordered, of variable length, and the input key values must be right-justified within their respective structure elements and padded on the left with binary zeros. If the leading bits of the modulus are zero's, do not count them in the length.

When COMP-TAG is specified, the following restrictions apply:

- Modulus length must be at least 2048 bits.
- The only clear key value that may be specified is the public exponent.

Table 463 on page 1154 defines the structure and contents as a function of key type.

Table 463. Key Value Structure Elements for PKA Key Token Build

Offset	Length (bytes)	Description
Key Value Structure: Optimized RSA, Chinese Remainder Theorem form (RSA-CRT, RSA-AESC, RSAAESC2)		
000	002	Modulus length in bits (512 to 4096). This is required.
002	002	Modulus field length in bytes, "nnn." This value can be zero if the key token is used as a *skeleton_key_token* in the PKA key generate callable service. This value must not exceed 512.
004	002	Public exponent field length in bytes, "eee." This value can be zero if the key token is used as a *skeleton_key_token* in the PKA key generate callable service.
006	002	Reserved, binary zero.
008	002	Length of the prime number, p, in bytes, "ppp." This value can be zero if the key token is used as a *skeleton_key_token* in the PKA key generate callable service. Maximum size of p + q is 512 bytes.
010	002	Length of the prime number, q, in bytes, "qqq." This value can be zero if the key token is used as a *skeleton_key_token* in the PKA key generate callable service. Maximum size of p + q is 512 bytes.
012	002	Length of d_p, in bytes, "rrr." This value can be zero if the key token is used as a *skeleton_key_token* in the PKA key generate callable service. Maximum size of $d_p + d_q$ is 512 bytes.
014	002	Length of d_q, in bytes, "sss." This value can be zero if the key token is used as a *skeleton_key_token* in the PKA key generate callable service. Maximum size of $d_p + d_q$ is 512 bytes.
016	002	Length of U, in bytes, "uuu." This value can be zero if the key token is used as a *skeleton_key_token* in the PKA key generate callable service. Maximum size of U is 512 bytes.
018	nnn	Modulus, n.

Table 463. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
018 + nnn	eee	Public exponent, e. This is an integer such that 1<e<n. e must be odd. When you are building a *skeleton_key_token* to control the generation of an RSA key pair, the public key exponent can be one of these values: 3, 5, 17, 257, 65537 (2^{16} + 1), or 0 to indicate that a full random exponent should be generated. The exponent field can be a null-length field if the exponent value is 0.
018 + nnn + eee	ppp	Prime number, p.
018 + nnn + eee + ppp	qqq	Prime number, q.
018 + nnn + eee + ppp + qqq	rrr	d_p = d mod(p-1).
018 + nnn + eee + ppp + qqq + rrr	sss	d_q = d mod(q-1).
018 + nnn + eee + ppp + qqq + rrr + sss	uuu	U = q^{-1}mod(p).
Key Value Structure: RSA Modulus-Exponent form (RSA-PRIV, RSA-PUBL, RSAMEVAR, RSA-AESM, RSAAESM2)		
000	002	Modulus length in bits. This is required. When building a skeleton token, the modulus length in bits must be greater than or equal to 512 bits.
002	002	Modulus field length in bytes, "XXX". This value must not exceed 512 when the RSA-PUBL, RSA-AESM, or RSAMEVAR keyword is used, and must not exceed 128 when the RSA-PRIV keyword is used. This service can build a key token for a public RSA key with a 4096-bit modulus length, or it can build a key token for a 1024-bit modulus length private key.
004	002	Public exponent field length in bytes, "YYY". This value must not exceed 512 when either the RSA-PUBL, RSA-AESM, or RSAMEVAR keyword is used, and must not exceed 128 when the RSA-PRIV keyword is used. This value can be zero if you are using the key token as a skeleton token in the PKA key generate verb. In this case, a random exponent is generated. To obtain a fixed, predetermined public key exponent, you can supply this field and the public exponent as input to the PKA key generate verb.
006	002	Private exponent field length in bytes, "ZZZ". This field can be zero, indicating that private key information is not provided. This value must not exceed 128 bytes. This value can be zero if you are using the key token as a skeleton token in the PKA key generate verb.
008	XXX	Modulus, n. This is an integer such that 1 < n <2**2048.The n is the product of p and q for primes p and q.

Table 463. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
008 + XXX	YYY	RSA public exponent, e. This is an integer such that 1<e<n. e must be odd. When you are building a *skeleton_key_token* to control the generation of an RSA key pair, the public key exponent can be one of these values: 3, 5, 17, 257, 65537 (2^{16} + 1), or 0 to indicate that a full random exponent should be generated. The exponent field can be a null-length field if the exponent value is 0.
008 + XXX + YYY	ZZZ	RSA secret exponent d. This is an integer such that 1<d<n. The value of d is e^{-1} mod(p-1)(q-1). e**-1 mod(p-1)(q-1); the product of e and d is 1 mod(p-1)(q-1). This can be a null-length field if you are using the key token as a skeleton token in the PKA key generate verb.
Key Value Structure: ECC Private/public key pair form (ECC-PAIR)		
000	001	Curve type **X'00'** Prime Curve **X'01'** Brainpool Curve **X'02'** Edwards Curve **X'03'** Koblitz Curve
001	001	Reserved x'00'
002	002	Length of *p* in bits **X'00A0'** 160 (Brainpool) **X'00C0'** 192 (Brainpool, Prime) **X'00E0'** 224 (Brainpool, Prime) **X'00FF'** 255 (Edwards curve25519) **X'0100'** 256 (Brainpool, Prime, Koblitz) **X'0140'** 320 (Brainpool) **X'0180'** 384 (Brainpool, Prime) **X'01C0'** 448 (Edwards curve448) **X'0200'** 512 (Brainpool) **X'0209'** 521 (Prime)

Table 463. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
004	002	ddd, This field is the length of the private key d value in bytes, This value can be zero if the key token is used as a skeleton key token in the PKA Key Generate callable service. The maximum value could be up to 66 bytes.
006	002	xxx, This field is the length of the public key Q value in bytes. This value can be zero if the key token is used as a skeleton key token in the PKA Key Generate callable service. The maximum value could be up to 133 bytes.
008	ddd	Private key d
008 + ddd	xxx	Public Key value Q. For Prime, Brainpool, or Koblitz, this must be an uncompressed point (prefixed with 0x04). For Edwards, this is the 'public key A', as described in RFC 8032.
Key value Structure: ECC Public form (ECC-PUBL)		
000	001	Curve type: **X'00'** Prime curve **X'01'** Brainpool curve **X'02'** Edwards curve **X'03'** Koblitz curve
001	001	Reserved x'00'
002	002	Length of *p* in bits **X'00A0'** 160 (Brainpool) **X'00C0'** 192 (Brainpool, Prime) **X'00E0'** 224 (Brainpool, Prime) **X'00FF'** 255 (Edwards curve25519) **X'0100'** 256 (Brainpool, Prime, Koblitz) **X'0140'** 320 (Brainpool) **X'0180'** 384 (Brainpool, Prime) **X'01C0'** 448 (Edwards curve448) **X'0200'** 512 (Brainpool) **X'0209'** 521 (Prime)

Table 463. Key Value Structure Elements for PKA Key Token Build (continued)		
Offset	**Length (bytes)**	**Description**
004	002	xxx, This field is the length of the public key Q value in bytes. This value can be zero if the key token is used as a skeleton key token in the PKA Key Generate callable service. The maximum value could be up to 133 bytes.
006	xxx	Public key value Q. For Prime, Brainpool, or Koblitz, this must be an uncompressed point (prefixed with 0x04). For Edwards, this is the 'public key A', as described in RFC 8032.
Key Value Structure: QSA Private/public key pair form (QSA-PAIR) and QSA Public key form (QSA-PUBL)		
000	001	Algorithm identifier: **X'01'** CRYSTALS-Dilithium Round 2. **X'02'** CRYSTALS-Kyber Round 2. **X'03'** CRYSTALS-Dilithium Round 3.
001	001	Clear key format: **X'00'** No Clear key. **X'01'** Clear private and public key pair in 'KAT format' (Known Answer Test format). **X'02'** Clear private and public key pair in 'parameterized format'. **X'03'** Clear Public key only. Must be specified with QSA-PUBL.
002	002	Algorithm Parameters: When Algorithm identifier is X'01', allowed values are: **X'0605'** CRYSTALS-Dilithium (6,5). **X'0807'** CRYSTALS-Dilithium (8,7). When Algorithm identifier is X'02', allowed values are: **X'1024'** CRYSTALS-Kyber (1024). When Algorithm identifier is X'03', allowed values are: **X'0605'** CRYSTALS-Dilithium (6,5). **X'0807'** CRYSTALS-Dilithium (8,7).

Table 463. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
004	002	Clear key length: ***ppp*** This field is the length of the QSA key value in bytes. This value can be zero if the key token is used as a skeleton key token in the PKA Key Generate callable service.
006	002	Reserved X'00'.

Table 463. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
008	*ppp*	QSA private/public key pair or QSA public key value. When the clear key format is X'01', see: • Table 464 on page 1161 for the KAT format for CRYSTALS-Dilithium keys. **Note:** The format matches the format of the NIST Known Answer Test keys for CRYSTALS-Dilithium, with the private key 'sk' in binary form, followed by public key 'pk'. The 'rho' appears twice. • Table 465 on page 1161 for the KAT format for CRYSTALS-Kyber keys. **Note:** Kyber1024 secret key 'sk' is in binary form, which includes public key 'pk'. The normal encapsulated form of the 'sk' produced by the reference Kyber1024 includes 'pk' components mixed with 'sk' secrets, in the order shown. When the clear key format is X'02', see: • Table 466 on page 1161 for the parameterized format for CRYSTALS-Dilithium keys. **Note:** This format matches the format of the NIST Known Answer Test keys for CRYSTALS-Dilithium, with the private key 'sk' in binary form, followed by public key 'pk'. The 'rho' does not appear twice. • Table 467 on page 1162 for the parameterized format for CRYSTALS-Kyber keys. **Note:** Kyber1024 secret key 'sk' is in binary form, which includes public key 'pk'. The normal encapsulated form of the 'sk' produced by the reference Kyber1024 includes 'pk' components mixed with 'sk' secrets, in the order shown. When the clear key format is X'03', see: • Table 468 on page 1162 for the public key format for CRYSTALS-Dilithium keys. **Note:** This format matches the format of the NIST Known Answer Test keys for CRYSTALS-Dilithium, with the public key 'pk' in binary form. • Table 469 on page 1162 for the public key format for CRYSTALS-Kyber keys. **Note:** Kyber1024 secret key 'sk' in binary form, which includes public key 'pk'. The normal encapsulated form of the 'sk' produced by the reference Kyber1024 includes 'pk' components mixed with 'sk' secrets in the order shown.

Notes:

1. All length fields are in binary.

2. All binary fields (exponent, lengths, modulus, and so on) are stored with the high-order byte field first. This integer number is right-justified within the key structure element field.
3. You must supply all values in the structure to create a token containing an RSA private key for input to the PKA key import service.

Table 464. Clear key format X'01' CRYSTALS-Dilithium key object layout with sizes

	Object	**Dilithium (6,5) Round 2**	**Dilithium (8,7) Round 2**	**Dilithium (6,5) Round 3**	**Dilithium (8,7) Round 3**
sk (private key)	rho	32	32	32	32
	Key D, 'seed'	32	32	32	32
	tr T ('tr')	48	48	32	32
	s1	480	672	640	672
	s2	576	768	768	768
	t0	2688	3584	2496	3328
pk (public key)	rho	32	32	32	32
	t1	1728	2304	1920	2560
Clear key length at offset 004		X'15F0' (5616 bytes)	X'1D30' (7472 bytes)	X'1740' (5952 bytes)	X'1D20' (7456 bytes)

Table 465. Clear Key format X'01' CRYSTALS-Kyber key object layout with sizes

	Object	**Kyber (1024) Round 2**
sk	secret polynomial vector	1536
pk	public polynomial vector	1536
	public seed	32
sk	hash of 'pk'	32
	'z': random bytes	32
Clear key length at offset 004		X'0C60' (3168 bytes)

Table 466. Clear key format X'02' CRYSTALS-Dilithium key object layout with sizes

	Object	**Dilithium (6,5) Round 2**	**Dilithium (8,7) Round 2**	**Dilithium (6,5) Round 3**	**Dilithium (8,7) Round 3**
pk (public key)	rho	32	32	32	32
sk (private key)	Key D, 'seed'	32	32	32	32
	tr T ('tr')	48	48	32	32
	s1	480	672	640	672
	s2	576	768	768	768
	t0	2688	3584	2496	3328
pk (public key)	t1	1728	2304	1920	2560
Clear key length at offset 004		X'15D0' (5584 bytes)	X'1D10' (7440 bytes)	X'1720' (5920 bytes)	X'1D00' (7424 bytes)

Table 467. Clear Key format X'02' CRYSTALS-Kyber key object layout with sizes		
	Object	**Kyber (1024) Round 2**
sk	secret polynomial vector	1536
pk	public polynomial vector	1536
	public seed	32
sk	hash of 'pk'	32
	'z': random bytes	32
Clear key length at offset 004		X'0C60' (3168 bytes)

Table 468. Clear key format X'03' CRYSTALS-Dilithium key object layout with sizes					
	Object	**Dilithium (6,5) Round 2**	**Dilithium (8,7) Round 2**	**Dilithium (6,5) Round 3**	**Dilithium (8,7) Round 3**
pk (public key)	rho	32	32	32	32
	t1	1728	2304	1920	2560
Clear key length at offset 004		X'06E0' (1760 bytes)	X'0920' (2336 bytes)	X'07A0' (1952 bytes)	X'0A20' (2592 bytes)

Table 469. Clear Key format X'03' CRYSTALS-Kyber key object layout with sizes		
	Object	**Kyber (1024) Round 2**
pk	public polynomial vector	1536
	public seed	32
Clear key length at offset 004		X'0620' (1568 bytes)

private_key_name_length

Direction	**Type**
Input	Integer

The length can be 0 or 64.

private_key_name

Direction	**Type**
Input	EBCDIC character

This field contains the *private-key name* that will be stored in the token. The name must conform to ICSF label syntax rules. That is, allowed characters are alphanumeric, national (@,#,$) or period (.). The first character must be alphabetic or national. The name is folded to uppercase and converted to ASCII characters. ASCII is the permanent form of the name because the name should be independent of the platform. The name is then cryptographically coupled with clear private key data prior to its encryption of the private key. Because of this coupling, the name can never change when the key token is already imported. This parameter is ignored with key type RSA-PUBL, ECC-PUBL, or QSA-PUBL.

For RSA and QSA keys, the *private-key name* is stored in the PKA private-key name section of the token.

For ECC keys, the *private-key name* is stored in the associated data section of the token.

user_definable_associated_data_length

Direction	Type
Input	Integer

The length of the *user_definable_associated_data* parameter.

Valid for Rule Array Key Type of ECC-PAIR with a maximum value of 100 and must be set to 0 for all other Rule Array Key Types.

user_definable_associated_data

Direction	Type
Input	String

The *user_definable_associated_data* parameter is a pointer to a string variable containing the associated data that will be placed following the IBM associated data in the token. The associated data is data whose integrity but not confidentiality is protected by a key wrap mechanism. It can be used to bind usage control information.

Valid for Rule Array Key Type of ECC-PAIR and is ignored for all others.

key_derivation_data_length

Direction	Type
Input	Integer

The length of the *key_derivation_data* parameter in bytes. When the token version keyword ECC-VER1 is specified, the value must be 0 or 4. Otherwise, the value must be 0.

key_derivation_data

Direction	Type
Input	String

The 4-byte key derivation data structure describing the key to be derived. This data will be used to create the optional key derivation section of an ECC key-token. Table 470 on page 1163 shows the contents of this structure.

Table 470. PKA Key Token Build key-derivation-data contents, ECC keys

Offset	Length (bytes)	Description
ECC key-derivation data, ECC-PAIR ECC-VER1		
000	001	Algorithm of the key to be derived: **X'01'** DES **X'02'** AES

Table 470. PKA Key Token Build key-derivation-data contents, ECC keys (continued)		
Offset	**Length (bytes)**	**Description**
001	001	Key type of the key to be derived: **X'01'** DATA **X'02'** EXPORTER **X'03'** IMPORTER **X'04'** CIPHER **X'05'** DECIPHER **X'06'** ENCIPHER **X'07'** CIPHERXI **X'08'** CIPHERXL **X'09'** CIPHERXO
002	002	Key bit length: 64, 128, 192, 256

reserved_3_length

Direction	Type
Input	Integer

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_3

Direction	Type
Input	String

The *reserved_3* parameter identifies a string that is reserved. The service ignores it.

reserved_4_length

Direction	Type
Input	Integer

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_4

Direction	Type
Input	String

The *reserved_4* parameter identifies a string that is reserved. The service ignores it.

reserved_5_length

Direction	Type
Input	Integer

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_5

Direction	Type
Input	String

The *reserved_5* parameter identifies a string that is reserved. The service ignores it.

key_token_length

Direction	Type
Input/Output	Integer

Length of the returned key token. The service checks the field to ensure it is at least equal to the size of the token to return. On return from this service, this field is updated with the exact length of the *key_token* created. On input, a size of 8000 bytes is sufficient to contain the largest *key_token* created.

key_token

Direction	Type
Output	String

The returned key token containing an unenciphered private or public key. The private key is in an external form that can be exchanged with different Common Cryptographic Architecture (CCA) PKA systems. You can use the public key token directly in appropriate ICSF signature verification or key management services.

Usage notes

If you are building a skeleton for use in a PKA Key Generate request to generate a retained PKA private key, you must build a private key name section in the skeleton token.

A CEX6C with the July 2019 or later licensed internal code (LIC) is required to generate an internal key token from an RSAAESM2 or RSAAESC2 external key token.

Required hardware

No cryptographic hardware is required by this callable service.

PKA Key Token Change (CSNDKTC and CSNFKTC)

The PKA Key Token Change callable service changes PKA key tokens (RSA, ECC, CRYSTALS-Dilithium, and CRYSTALS-Kyber) or trusted block key tokens, from encipherment under the cryptographic coprocessor's old RSA master key or ECC master key to encipherment under the current cryptographic coprocessor's RSA master key or ECC master key.

- For RSA key tokens - Key tokens must be private internal PKA key tokens to be changed by this service.
- For trusted block key tokens - Trusted block key tokens must be internal.
- For ECC key tokens - Key tokens must be private internal ECC key tokens encrypted under the ECC master key.
- For CRYSTALS - Dilithium key tokens - Key tokens must be private internal CRYSTALS-Dilithium key tokens encrypted under the ECC master key.

- For CRYSTALS-Kyber key tokens - Key tokens must be private internal CRYSTALS-Kyber key tokens encrypted under the ECC master key.

The callable service name for AMODE(64) invocation is CSNFKTC.

Format

```
CALL CSNDKTC(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_identifier_length,
             key_identifier )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 1 or 2.

rule_array

Direction	Type
Input	String

The process rules for the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justitfied in its 8-byte location and padded on the right with blanks.

Table 471. Rule Array Keywords for PKA Key Token Change

Keyword	Meaning
Algorithm (optional)	
RSA	Specifies that the key token is for an RSA or DSS key or trusted block token. This is the default.
ECC	Specifies that the key token is for an ECC key.
QSA	Specifies that the key being changed is in a QSA key token such as CRYSTALS-Dilithium or CRYSTALS-Kyber keys.
Reencipherment method (required)	
RTCMK	If the *key_identifier* is an RSA key token, the service will change an RSA private key from encipherment with the old RSA master key to encipherment with the current RSA master key. If the *key_identifier* is a trusted block token, the service will change the trusted block's embedded MAC key from encipherment with the old RSA master key to encipherment with the current RSA master key. If the *key_identifier* is an ECC, CRYSTALS-Dilithium, or CRYSTALS-Kyber key token, the service will change the private key from encipherment with the old ECC master key to encipherment with the current ECC master key.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter. The maximum size is 8000 bytes.

key_identifier

Direction	Type
Input/Output	String

Contains an internal key token of an internal RSA, ECC, CRYSTALS-Dilithium, CRYSTALS-Kyber, or trusted block key.

If the key token is an RSA key token, the private key within the token is securely reenciphered under the current RSA or ECC master key.

If the key token is a Trusted Block key token, the MAC key within the token is securely reenciphered under the current RSA master key.

If the key token is an ECC key token, the private key within the token is securely reenciphered under the current ECC master key.

If the key token is a CRYSTALS-Dilithium or CRYSTALS-Kyber key token, the private key within the token is securely reenciphered under the current ECC master key.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

To use this service, PKA callable services must be enabled for all RSA and DSS token types. For systems with CCA Cryptographic coprocessors that are a CEX3C or later, there is no PKA callable services control. The RSA master key must be valid to use this service.

While DSS tokens can be processed by this service, they are not useable by any other callable services.

To use this service for ECC, CRYSTALS-Dilithium, and CRYSTALS-Kyber tokens, the ECC master key must be valid.

Access control points

The **PKA Key Token Change RTCMK** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 472. PKA Key Token Change required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	CRYSTALS-Dilithium keys not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 private keys are not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	CRYSTALS-Dilithium keys not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 private keys are not supported.

Table 472. PKA Key Token Change required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	CRYSTALS-Dilithium keys not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported. CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 private keys are not supported.
	Crypto Express7 CCA Coprocessor	CRYSTALS-Dilithium key support require the June 2020 or later licensed internal code (LIC). Edwards curves Ed25519 and Ed448 require the June 2020 or later licensed internal code (LIC). ECC Koblitz curve secp256k1 requires the September 2020 or later licensed internal code (LIC). CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 private keys are not supported.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor	CRYSTALS-Dilithium and CRYSTAL-Kyger keys not supported. Edwards curves Ed25519 and Ed448 are not supported. ECC Koblitz curve secp256k1 is not supported.
	Crypto Express7 CCA Coprocessor	CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 private keys are not supported.
	Crypto Express8 CCA Coprocessor	CRYSTALS-Kyber, CRYSTALS-Dilithium (8,7) Round 2 or Round 3, and CRYSTALS-Dilithium (6,5) Round 3 keys require CCA release 8.0 or later licensed internal code (LIC).

PKA Key Translate (CSNDPKT and CSNFPKT)

The PKA Key Translate callable service is used to do the following:

Translation

Translate a CCA RSA key token into an external key token. The format of the external key token is specified by the output format keyword of the *rule_array* parameter. Supported output formats are: smart card, EMV, and PKCS #11 object.

The source CCA RSA key token must be wrapped with a transport key-encrypting key (KEK). The XLATE bit must also be turned on in the key usage byte of the source token. The source token is unwrapped using the specified source transport KEK. The target key token will be wrapped with the specified target transport KEK. Existing information in the target token is overwritten. There are restrictions on which type key can be used for the source and target transport key tokens. These restrictions are enforced by access control points.

Conversion

- Convert the object protection key (OPK) in an CCA RSA private key token from a DES key to an AES key (EXTDWAKW, INTDWAKW).

  The service will convert an existing internal or external RSA private key token. The modulus-exponent and Chinese Remainder Theorem forms are supported. Private key section identifiers 0x06, 0x08, and 0x09 can be converted.

  If a format restriction keyword is specified, the output key token will have the format restriction flag in the associated data section of the token set to a requested value. The key token can only be used to create signatures in the format specified.
- Change the key usage attributes of an internal CCA AES OPK key token (RSAAESC2 or RSAAESM2 private key). (INTUSCHG).
- Convert an internal CCA key token to a compliant-tagged token. (COMP-TAG).

Compliance checking
Check that a CCA key token can have the compliant tag.

Export
Export an EC, CRYSTALS-Dilithium, or CRYSTALS-Kyber key from a CCA internal key token to the AESKW external format, wrapped by an AES key encrypting key. See "AESKW external format" on page 1584 for the layout of AESKW external format.

The callable service name for AMODE(64) invocation is CSNFPKT.

Format

```
CALL CSNDPKT(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              source_key_identifier_length,
              source_key_identifier,
              source_transport_key_identifier_length,
              source_transport_key_identifier,
              target_transport_key_identifier_length,
              target_transport_key_identifier,
              target_key_token_length,
              target_key_token)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. Value must be 1 to 8 inclusive.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. See Table 473 on page 1171 for a list. A keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 473. Keywords for PKA Key Translate Rule Array

Keyword	Meaning
Process rule (one required)	
Translation rules The source key token must be an external token and allow translation (TRANSLAT/XLATE key usage). The target transport key must allow translation in the key usage attributes.	
CKM-RAKW	Specifies the translation of an external CCA key token containing an RSA or EC private key into an external encrypted PKCS #11 object. The target transport key is an RSA CCA public key to wrap the ephemeral AES key that wraps the private key in the object.
EMVCRT	Specifies the translation of an external CCA key token containing an RSA CRT private key into the EMVCRT format and wrapped using TDES-ECB.
EMVDDA	Specifies the translation of an external CCA key token containing an RSA CRT private key into EMV dynamic data authentication (DDA) format and wrapped using TDES-CBC.

Table 473. Keywords for PKA Key Translate Rule Array (continued)	
Keyword	**Meaning**
EMVDDAE	Specifies the translation of an external CCA key token containing an RSA CRT private key into EMV DDAE format and wrapped using TDES-ECB.
SCCOMCRT	Specifies the translation of an external CCA key token containing an RSA CRT private key into the smart card Chinese Remainder Theorem format.
SCCOMME	Specifies the translation of an external CCA key token containing an RSA ME private key into the smart card Modulus-Exponent format.
SCVISA	Specifies the translation of an external CCA key token containing an RSA ME private key into the smart card Visa proprietary format.
Conversion rules	
COMP-TAG	Specifies to convert the internal RSA private key token into a compliant-tagged token. When the input key type is not RSAAESC2 or RSAAESM2, the following rules apply: • When key-usage KM-ONLY is enabled, the token will be converted to a compliant-tagged RSAAESC2 or RSAAESM2 with key-usage U-KEYENC enabled. • When key-usage SIG-ONLY is enabled, the token will be converted to a compliant-tagged RSAAESC2 or RSAAESM2 with key-usage U-DIGSIG enabled. • For all other key-usages, use INTUSCHG to set the desired usage prior to using COMP-TAG.
EXTDWAKW	Specifies to convert an external RSA private key token with a TDES wrapped OPK to an AESKW wrapped OPK (RSA-AESC, RSAAESC2, RSA-AESM, or RSAAESM2).
INTDWAKW	Specifies to convert an internal RSA private key token with a TDES wrapped OPK to an AESKW wrapped OPK (RSA-AESC, RSAAESC2, RSA-AESM, or RSAAESM2).
INTUSCHG	Specifies to change the usage attributes of an internal RSA key token. Requires keyword group PKA Key Usage Control. Not valid if the input key is compliant-tagged.
Compliant checking rules	
COMP-CHK	Checks if an internal RSA key token can have the compliant tag.
Export rules	
ECC-AES1	Export an ECC key wrapped by an AES key encrypting key.
QSA-AES1	Export a QSA key, such as a CRYSTALS-Dilithium or CRYSTALS-Kyber key, wrapped by an AES key encrypting key.
EMV DDA encrypted key part data format (one, optional). Only valid with output format keyword EMVDDA or EMVDDAE.	
EMV1	Original EMV DDA output format. This is the default.

Table 473. Keywords for PKA Key Translate Rule Array (continued)	
Keyword	**Meaning**
EMVLENBT	Modified EMV DDA output format, which includes a length byte that becomes part of the encrypted key part section. The length byte, which replaces a post-padding byte of X'00' that **EMV1** uses, is prepended to the clear key part. This length is valued to the number clear key-part bytes and does not include any pad bytes.
Format restriction (one, optional). Only valid with keywords INTDWAKW and EXTDWAKW.	
FR-NONE	Specifies to not restrict the private key to be used to a particular method. The key is usable for any method. This is the default.
FR-I9796	Specifies to render the private key usable with the ISO-9796 digital-signature hash formatting method.
FR-PK10	Specifies to render the private key usable with the PKCS-1.0 digital-signature hash formatting method.
FR-PK11	Specifies to render the private key usable with the PKCS-1.1 digital-signature hash formatting method.
FR-PSS	Specifies to render the private key usable with the PKCS-PSS digital-signature hash formatting method.
FR-X9.31	Specifies to render the private key usable with the X9.31 digital-signature hash formatting method.
FR-ZPAD	Specifies to render the private key usable with the ZERO-PAD digital-signature hash formatting method.
PKA Key Usage Control. Only valid with INTUSCHG. The keywords specified reflect the only usage attributes that will be enabled in the output key token. All other usage attributes will be disabled.	
U-DIGSIG	Digital Signature usage is allowed. When input key type is RSAAESC2 or RSAAESM2, requires that the U-DIGSIG flag is enabled in the input key token. When input key type is not RSAAESC2 or RSAAESM2, requires that the KEY-MGMT or SIG-ONLY flag is enabled in the input key token.
U-NONRPD	Non-Repudiation usage is allowed. When input key type is RSAAESC2 or RSAAESM2, requires that the U-NONRPD flag is enabled in the input key token. When input key type is not RSAAESC2 or RSAAESM2, requires that the KEY-MGMT or SIG-ONLY flag is enabled in the input key token.
U-KCRTSN	keyCertSign usage is allowed. When input key type is RSAAESC2 or RSAAESM2, requires that the U- KCRTSN flag is enabled in the input key token. When input key type is not RSAAESC2 or RSAAESM2, requires that the KEY-MGMT or SIG -ONLY flag is enabled in the input key token.

Table 473. Keywords for PKA Key Translate Rule Array (continued)	
Keyword	**Meaning**
U-CRLSN	Certificate Revocation List Sign usage is allowed. When input key type is RSAAESC2 or RSAAESM2, requires that the U-CRLSN flag is enabled in the input key token. When input key type is not RSAAESC2 or RSAAESM2, requires that the KEY-MGMT or SIG -ONLY flag is enabled in the input key token.
U-KEYENC	Key Encipherment usage is allowed. When input key type is RSAAESC2 or RSAAESM2, requires that the U- KEYENC flag is enabled in the input key token. When input key type is not RSAAESC2 or RSAAESM2, requires that the KEY-MGMT or KM -ONLY flag is enabled in the input key token.
U-DATENC	Data Encipherment usage is allowed. When input key type is RSAAESC2 or RSAAESM2, requires that the U- DATENC flag is enabled in the input key token. When input key type is not RSAAESC2 or RSAAESM2, requires that the KEY-MGMT or KM -ONLY flag is enabled in the input key token.
U-KEYAGR	Key agreement usage is allowed. When input key type is RSAAESC2 or RSAAESM2, requires that the U- KEYAGR flag is enabled in the input key token. When input key type is not RSAAESC2 or RSAAESM2, requires that the KEY-MGMT or KM -ONLY flag is enabled in the input key token.
Key Agreement Control (One, Optional). Only valid with U-KEYAGR.	
U-ENCONL	Only encipher operations are allowed during key agreement key establishment protocols. When input key type is RSAAESC2 or RSAAESM2, the U-DECONL must not be enabled in the input key token.
U-DECONL	Only decipher operations are allowed during key agreement key establishment protocols. When input key type is RSAAESC2 or RSAAESM2, the U-ENCONL must not be enabled in the input key token.

source_key_identifier_length

Direction	Type
Input	Integer

Length in bytes of the *source_key_identifier* parameter. If the *source_key_identifier* contains a label, the length must be 64. Otherwise, the value must be between the actual length of the token and 8000.

source_key_identifier

Direction	Type
Input	String

The key identifier of the RSA, ECC, CRYSTALS-Dilithium, or CRYSTALS-Kyber private key to be processed. For translation, the key is an external key token wrapped with an AES or DES key-encrypting key. For OPK conversion, the token may be internal or external. External tokens are wrapped with a DES key encrypting key. When an internal token is specified, the transport keys are not used. For export to AESKW external format, the token is an internal token.

When keyword COMP-CHK is specified, this must be an internal RSA private key token.

When keyword COMP-TAG or INTUSCHG is specified, this must be an internal RSA private key token with private key section X'08', X'30', or X'31'.

When keyword CKM-RAKW is specified, this must be an external RSA private key token with private key section X'08', X'30', or X'31' or an external EC private key token with private key section X'20'. Compliance tagged key tokens are not supported.

source_transport_key_identifier_length

Direction	Type
Input	Integer

Length in bytes of the *source_transport_key_identifier* parameter.

When the *source_transport_key_identifier*contains a label, the length must be 64.

When the processing rule is INTDWAKW, INTUSCHG, COMP-CHK, COMP-TAG, ECC-AES1, or QSA-AES1, the value must be zero.

Otherwise, the value must be between the actual length of the token and 9992.

source_transport_key_identifier

Direction	Type
Input/Output	String

The key identifier of the key to unwrap the source key. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

When the *source_transport_key_identifier_length* is zero, this parameter is ignored.

For EC and RSA RSA-AESC, RSAAESC2, RSA-AESM, or RSAAESM2 key tokens, this is an CCA AES EXPORTER or IMPORTER key toke with the TRANSLAT key usage attribute or an X9.143 (TR-31) AES key block with key usage K0, algorithm A, mode of use D or E.

For other RSA key tokens, this is a CCA DES EXPORTER or IMPORTER key token with the XLATE control vector attribute or an X9.143 (TR-31) TDES key block with key usage K0, algorithm T, mode of use D or E.

See “Access control points” on page 1177 for details on the type of transport key that can be used.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

target_transport_key_identifier_length

Direction	Type
Input	Integer

Length in bytes of the *target_transport_key_identifier* parameter.

If the *target_transport_key_identifier* contains a label, the length must be 64.

When the processing rule is INTDWAKW, INTUSCHG, COMP-CHK, or COMP-TAG, the value must be zero.

Otherwise, the value must be between the actual length of the token and 9992.

target_transport_key_identifier

Direction	Type
Input/Output	String

The key identifier of the key that will wrap the output key in the *target_key_token* parameter. The key identifier is an operational key token or key block or the key label of an operational token or block in key storage.

When the *target_transport_key_identifier_length* is zero, this parameter is ignored.

When the processing rule is EMVCRT, EMVDDA, EMVDDAE, SCCOMCRT, SCCOMME, or SCVISA, the key is a DES key-encrypting key with the TRANSLATE key usage attribute.

- CCA key token for a DES IMPORTER or EXPORTER with the XLATE control vector attribute.
- X9.143 (TR-31) key block for a TDES key-encrypting key with key usage K0, algorithm T, mode of use E or D.

When the processing rule is EXTDWAKW, the key is an AES key-encrypting key with the TRANSLAT key usage attribute.

- CCA key token for an IMPORTER or EXPORTER with the TRANSLAT key usage attribute.
- X9.143 (TR-31) key block for an AES key-encrypting key with key usage K0, algorithm A, mode of use E or D.

When the processing rule is ECC-AES1 or QSA-AES1, the key is an AES key-encrypting key.

- CCA key token for an EXPORTER with the WR-QSA or WR-ECC bit enabled in Key-usage field 3, high-order byte.
- X9.143 (TR-31) key block for an AES key-encrypting key with key usage K0, algorithm A, mode of use E or D.

When the processing rule is CKM-RAKW, the key is an RSA public key with a modulus bit length of 2048, 3072, or 4096. The key will wrap the ephemeral AES key that wraps the private key.

See “Access control points” on page 1177 for details on the type of transport key that can be used.

If the token or block supplied was encrypted under the old master key, the token or block is returned encrypted under the current master key.

target_key_token_length

Direction	Type
Input/Output	Integer

Length in bytes of the *target_key_token* parameter. On output, the value in this variable is updated to contain the actual length of the *target_key_token* produced by the callable service. The maximum length is 8000 bytes.

If the COMP-CHK keyword is specified, this parameter must be 0.

target_key_token

Direction	Type
Output	String

The processed key token.

When converting to an AES OPK format, the token is a CCA key token wrapped by an AES key-encrypting key (EXTDWAKW) or an internal token (INTDWAKW). When the INTUSCHG keyword is specified, the output will be an internal RSA private key token with private key section X'30' and associated data version X'04' (RSAAESM2) or an internal RSA private key token with private key section X'31' and associated data version X'05' (RSAAESC2). Internal tokens may be stored in the PKDS.

When translating to a non-CCA smart card format, the key token is wrapped with the key-encrypting key specified in the *target_transport_key_identifier* parameter. The key token is not a CCA token and cannot be stored in the PKDS.

When the processing rule is CKM_RAKW, the output is a structure containing the AES ephemeral key wrapped by the RSA public key specified in the *target_transport_key_identifier* parameter and the wrapped private key.

Restrictions

CCA RSA ME tokens will not be translated to the SCCOMCRT, EMV DDA, EMV DDAE, or the EMV CRT formats. CCA RSA CRT tokens will not be translated to the SCCOMME format. SCVISA only supports Modulus-Exponent (ME) keys.

The maximum modulus size of CCA RSA CRT tokens for the EMVDDA, EMVDDAE, or the EMVCRT formats is 2040 bits.

Only CCA RSA CRT tokens with a private section of X'08' are supported by the EMVDDA, EMVDDAE, or the EMVCRT rule array keywords.

Access control points

There are access control points that control use of the format rule array keywords and the type of transport keys that can be used.

Table 474. Required access control points for PKA Key Translate

Rule array keyword	Access control point
COMP-CHK	PKA Key Translate - Allow COMP-CHK
COMP-TAG	PKA Key Translate - Allow COMP-TAG
CKM-RAKW and the source key is an EC key	PKA Key Translate – From CCA ECC to CKM-RAKW format
CKM-RAKW and the source key is an RSA key	PKA Key Translate – From CCA RSA to CKM-RAKW format
EMVCRT	PKA Key Translate - from CCA RSA CRT to EMV CRT Format
EMVDDA	PKA Key Translate - from CCA RSA CRT to EMV DDA Format
EMVDDAE	PKA Key Translate - from CCA RSA CRT to EMV DDAE Format
EXTDWAKW	PKA Key Translate - Translate external key token
INTDWAKW	PKA Key Translate - Translate internal key token
INTUSCHG	PKA Key Translate - Allow INTUSCHG
SCCOMCRT	PKA Key Translate - from CCA RSA to SC CRT Format
SCCOMME	PKA Key Translate - from CCA RSA to SC ME Format
SCVISA	PKA Key Translate - from CCA RSA to SC Visa Format
ECC-AES1	PKA Key Translate - Allow ECC private key export

Table 474. Required access control points for PKA Key Translate (continued)	
Rule array keyword	**Access control point**
QSA-AES1	PKA Key Translate - Allow QSA private key export

These access control points control the key type combination shown in this table. One of these access control points must be enabled.

Table 475. Required access control points for source/target transport key combinations		
Source transport key type	**Target transport key type**	**Access control point**
EXPORTER	EXPORTER	PKA Key Translate - from source EXP KEK to target EXP KEK
IMPORTER	EXPORTER	PKA Key Translate - from source IMP KEK to target EXP KEK
IMPORTER	IMPORTER	PKA Key Translate - from source IMP KEK to target IMP KEK
EXPORTER	IMPORTER	(Not allowed)

When the **Disallow translation from AES wrapping to DES wrapping** access control point is enabled, this service will fail if the *source_transport_key_identifier* is an AES key and the *target_transport_key_identifier* is a DES key.

When the **Disallow translation from AES wrapping to weaker AES wrapping** access control point is enabled, this service will fail if the *source_transport_key_identifier* is an AES key that is stronger than the *target_transport_key_identifier.*

When the **Disallow translation from DES wrapping to weaker DES wrapping** access control point is enabled, this service will fail if the *source_transport_key_identifier* is a DES key that is stronger than the *target_transport_key_identifier*.

The **Allow weak DES wrap of RSA** access control allows a weaker DES key-encrypting key to wrap an RSA private key token. The **Prohibit weak wrap – Transport keys** access control must be enabled and this access control will override the restriction.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 476. PKA Key Translate required hardware

Server	Required Cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	Support for the format restriction *rule_array* keywords requires the October 2016 or later licensed internal code (LIC). Triple-length DES keys and *rule_array* keywords EMV1 and EMVLENBT require the July 2019 or later licensed internal code (LIC). Key types RSAAESM2 and RSAAESC2 are not supported. Keywords CKM-RAKW, COMP-CHK, COMP-TAG, INTUSCHG, U-DIGSIG, U-NONRPD, U-KCRTSN, U-CRLSN, U-KEYENC, U-DATENC, U-KEYAGR, U-ENCONL, and U-DECONL are not supported. Compliant-tagged key tokens are not supported. Keywords ECC-AES1 and QSA-AES1 are not supported. X9.143 key blocks are not supported.

Table 476. PKA Key Translate required hardware (continued)

Server	Required Cryptographic hardware	Restrictions
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	Support for the format restriction *rule_array* keywords requires the October 2016 or later licensed internal code (LIC). Triple-length DES keys and *rule_array* keywords EMV1 and EMVLENBT require the July 2019 or later licensed internal code (LIC). Key types RSAAESM2 and RSAAESC2 are not supported. Keywords CKM-RAKW, COMP-CHK, COMP-TAG, INTUSCHG, U-DIGSIG, U-NONRPD, U-KCRTSN, U-CRLSN, U-KEYENC, U-DATENC, U-KEYAGR, U-ENCONL, and U-DECONL are not supported. Compliant-tagged key tokens are not supported. Keywords ECC-AES1 and QSA-AES1 are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Triple-length DES keys and *rule_array* keywords EMV1 and EMVLENBT require the December 2018 or later licensed internal code (LIC). Key types RSAAESM2 and RSAAESC2 require the July 2019 or later licensed internal code (LIC). Keywords CKM-RAKW, COMP-CHK, COMP-TAG, INTUSCHG, U-DIGSIG, U-NONRPD, U-KCRTSN, U-CRLSN, U-KEYENC, U-DATENC, U-KEYAGR, U-ENCONL, and U-DECONL require the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC). Keywords ECC-AES1 and QSA-AES1 are not supported. X9.143 key blocks are not supported.

Table 476. PKA Key Translate required hardware (continued)

Server	Required Cryptographic hardware	Restrictions
IBM z15 IBM z15 T02	Crypto Express5 CCA Coprocessor	Key types RSAAESM2 and RSAAESC2 are not supported. Keywords CKM-RAKW, COMP-CHK, COMP-TAG, INTUSCHG, U-DIGSIG, U-NONRPD, U-KCRTSN, U-CRLSN, U-KEYENC, U-DATENC, U-KEYAGR, U-ENCONL, and U-DECONL are not supported. Compliant-tagged key tokens are not supported. Keywords ECC-AES1 and QSA-AES1 are not supported. X9.143 key blocks are not supported.
	Crypto Express6 CCA Coprocessor	Rule array keyword CKM-RAKW is not supported. Keywords ECC-AES1 and QSA-AES1 are not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Rule array keyword CKM-RAKW requires the CCA release 7.4 or later licensed internal code (LIC). Keywords ECC-AES1 and QSA-AES1 are not supported. X9.143 key blocks are not supported.
IBM z16 IBM z16 A02	Crypto Express6 CCA Coprocessor	Keywords CKM-RAKW, ECC-AES1 and QSA-AES1 are not supported. X9.143 key blocks are not supported.
	Crypto Express7 CCA Coprocessor	Keywords ECC-AES1 and QSA-AES1 are not supported. X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	Keywords ECC-AES1 and QSA-AES1 require CCA release 8.0 or later licensed internal code (LIC). X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

PKA Public Key Extract (CSNDPKX and CSNFPKX)

Use the PKA public key extract callable service to extract a PKA public key token from a supplied PKA internal or external private key token. This service performs no cryptographic verification of the PKA private token. You can verify the private token by using it in a service such as digital signature generate.

The callable service name for AMODE(64) invocation is CSNFPKX.

Format

```
CALL CSNDPKX(
             return_code,
             reason_code,
             exit_data_length,
```

```
                exit_data,
                rule_array_count,
                rule_array,
                source_key_identifier_length,
                source_key_identifier,
                target_public_key_token_length,
                target_public_key_token)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

Reserved field. This field is not used, but you must specify it.

source_key_identifier_length

Direction	Type
Input	Integer

The length of the *source_key_identifier* parameter. The maximum size is 8000 bytes. When the *source_key_identifier* parameter is a key label, this field must be 64 bytes.

source_key_identifier

Direction	Type
Input/Output	String

The internal or external token of a PKA private key or the label of a PKA private key. This can be the input or output from PKA key import or from PKA key generate.

This service supports:

- RSA private key token formats. If the *source_key_identifier* specifies a label for a private key that has been retained within a cryptographic coprocessor, this service extracts only the public key section of the token.
- ECC private key token formats.
- CRYSTALS-Dilithium private key token formats.
- CRYSTALS-Kyber private key token formats.

target_public_key_token_length

Direction	Type
Input/Output	Integer

The length of the *target_public_key_token* parameter. The maximum size is 8000 bytes. On output, this field will be updated with the actual byte length of the *target_public_key_token*.

target_public_key_token

Direction	Type
Output	String

This field contains the token of the extracted PKA public key.

Usage notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the PKDS.

This service extracts the public key from the internal or external form of a private key. However, it does not check the cryptographic validity of the private token.

Required hardware

No cryptographic hardware is required by this callable service.

Public Infrastructure Certificate (CSNDPIC and CSNFPIC)

Use the Public Infrastructure Certificate Service to create a self-signed PKCS #10 certificate signing request (CSR) based on an existing RSA or ECC private key/public key pair. The self-signed PKCS #10 request for the input public key is signed by the input private key.

For a PKCS #10 CSR, the input private key can be an internal RSA or ECC token or the label of private key or the label of an RSA retained key token. In addition to the input private key, the user must specify extra input parameters that specify the following:

- The subject's distinguished name.
- The key usage and constraints indicators.
- The signature algorithm and hashing-method.
- The certificate extensions.
- The output format of the CSR.

The callable service name for AMODE(64) invocation is CSNFPIC.

Format

```
CALL CSNDPIC(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             subject_private_key_identifier_length,
             subject_private_key_identifier,
             subject_name_length,
             subject_name,
             extensions_length,
             extensions,
             reserved1_length,
             reserved1,
             reserved2_length,
             reserved2,
             reserved3_length,
             reserved3,
             reserved4,
             reserved5,
             reserved6_length,
             reserved6_data,
             certificate_length,
             certificate )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Ignored	Integer

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the rule_array parameter. Value must be in the range of 6 -14.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. The *rule_array* keywords are:

Table 477. Keywords for Public Infrastructure Certificate

Keyword	Meaning
Requested action (One required).	
PK10SNRQ	Specifies to create a PKCS #10 CSR request from the input private key (which always includes a public key section).
Issuer modifier (One required, specifies how the issuer and issuer's distinguished name will be determined).	
SELFSIGN	Specifies that the CSR is for a self-signed certificate. The issuer's distinguished name is the value that is passed in the *subject_name* parameter.
Input subject name format indicator (One required, specifies the format of the input subject_name parameter).	
SDNDER	Specifies that the input *subject_name* is ASN.1 DER encoded.

Table 477. Keywords for Public Infrastructure Certificate (continued)

Keyword	Meaning
SDNCLEAR	Specifies that the input *subject_name* is specified as a series of X.509 attribute-value pairs that are separated by commas. For example: cn=Thomas Watson,o=Endicott,c=US,o=IBM The allowable attribute identifiers are: **Identifier** **Meaning** **C** countryName **O** organizationName **OU** organizationalUnitName **CN** commonName **SN** surname **L** localityName **ST** stateOrProvinceName **SP** stateOrProvinceName **S** stateOrProvinceName **T** title **PC** postalCode **EMAIL** emailAddress **E** emailAddress **EMAILADDRESS** emailAddress **STREET** streetAddress **DC** domainComponent **MAIL** mail **NAME** name **GIVENNAME** givenName

Table 477. Keywords for Public Infrastructure Certificate (continued)

Keyword	Meaning
SDNCLEAR	**Identifier** **Meaning** **INITIALS** initials **GENERATIONQUALIFIER** generationQualifier **DNQUALIFIER** dnQualifier **SERIALNUMBER** serialNumber To specify a comma within an attribute value, escape the comma with the '\' character. For example, an organization name of 'IBM,Poughkeepsie' would be specified as: OU=IBM\,Poughkeepsie.
Output format indicator (One required, specifies the format of the data returned in the certificate parameter).	
DER-FMT	Specifies that the output in the certificate parameter object are DER encoded according to the X.509 standard.
PEM-FMT	Specifies that the output in the certificate parameter are encoded using Base64 encoding according to RFC 7468. The encoded stream will consist of ASCII printable characters with one line feed (X'25') inserted after each group of 64 encoded characters and one line feed at the end of the encoded stream.
Key usage and constraint indicators. Specifies key usage indicators that are encoded as allowed in the Key Usage extension in the data that is returned in the output certificate parameter. 1-7 of these rules are required if the extensions parameter does not specify any allowed usages for Key Usage or Extended Key Usage. None of these rules are allowed if the extensions parameter is specified and does specify usages for Key Usage or Extended Key Usage. Also, U-DECONL and U-ENCONL requires U-KEYAGR to be specified. U-DECONL cannot be combined with U-ENCONL.	
U-CRLSN	Specifies that cRLSign is allowed.
U-DATENC	Specifies that data encipherment key usage is allowed.
U-DECONL	Specifies that decipherOnly is allowed.
U-DIGSIG	Specifies that digitalSignature is allowed.
U-ENCONL	Specifies that encipherOnly is allowed.
U-KCRTSN	Specifies that keyCertSign is allowed.
U-KEYAGR	Specifies that keyAgreement is allowed.
U-KEYENC	Specifies that key encipherment key usage is allowed.
U-NONRPD	Specifies that nonRepudiation is allowed.
Signature Algorithm specification (One required, specifies the signature algorithm is to be used in creating the data returned in the certificate parameter. A hash method rule must also be specified.	
RSA	Specifies to use the RSA signature algorithm. The hash method that is used must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.

Table 477. Keywords for Public Infrastructure Certificate (continued)	
Keyword	**Meaning**
ECDSA	Specifies to use the ECDSA signature algorithm. The hash method that is used must be SHA-224, SHA-256, SHA-384, or SHA-512.
Hashing-method specification (One required, specifies the hashing method is to be used in conjunction with the required signature algorithm in creating the data return in the certificate parameter.	
SHA-1	The hash method to use is SHA-1. Not allowed with the ECDSA Signature Algorithm.
SHA-224	The hash method to use is SHA-224.
SHA-256	The hash method to use is SHA-256.
SHA-384	The hash method to use is SHA-384.
SHA-512	The hash method to use is SHA-512.

subject_private_key_identifier_length

Direction	Type
Input	Integer

The length of the *subject_private_key_identifier* parameter in bytes. The maximum value is 3500.

subject_private_key_identifier

Direction	Type
Input	String

Contains an internal token or label of a private RSA or ECC key or the label of an RSA retained key token. The key usage attributes must not conflict with the key usage keywords.

subject_name_length

Direction	Type
Input	Integer

The length of the *subject_name* parameter in bytes. The maximum value is 400.

subject_name

Direction	Type
Input	String

The subject distinguished name (SDN) that is used for creating the output in the certificate parameter. When the SELFSIGN rule is specified, the *subject_name* is also used for the issuer distinguished name (IDN) in the output certificate parameter.

extensions_length

Direction	Type
Input	Integer

The length of the *extensions* parameter in bytes. The maximum value is 1000.

extensions

Direction	Type
Input	String

Specifies a DER encoded set of x.509 extensions. If any of the extensions specify key usage or extended key usage indicators, then no key usage or constraint indicator rules can also be specified.

reserved1_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved1

Direction	Type
Input	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved2

Direction	Type
Input	String

This field is ignored.

reserved3_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved3

Direction	Type
Input	String

This field is ignored.

reserved4

Direction	Type
Input	Integer

This parameter must be zero.

reserved5

Direction	Type
Input	Integer

This parameter must be zero.

reserved6_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved6_data

Direction	Type
Input	String

This field is ignored.

certificate_length

Direction	Type
Input/Output	Integer

On input, the length in bytes of the buffer for the certificate parameter. The maximum value is 3500.

On output, the length in bytes of the data returned in the certificate parameter.

certificate

Direction	Type
Input/Output	String

On input, specifies the buffer to be used for the output certificate.

On output, contains the data that is requested by the *Requested Action* rule.

Access control point

Table 478. Required access control points for Public Infrastructure Certificate

Access control point	Function control
Public Infrastructure Certificate	Allows the service to be used.
Public Infrastructure Certificate - PK10SNRQ	Allows the PK10SNRQ keyword to be used.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 479. Public Infrastructure Certificate required hardware		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z13 **IBM z13s**		This callable service is not supported.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	This callable service is not supported.
	Crypto Express6 CCA Coprocessor	Generating a CSR from a private key that does not allow digital signatures requires the July 2019 or later licensed internal code (LIC). Compliant-tagged key tokens require a CEX6C with the July 2019 or later licensed internal code (LIC).
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor	This service is not supported.
	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Retained Key Delete (CSNDRKD and CSNFRKD)

Use the Retained Key Delete callable service to delete a key that has been retained within cryptographic coprocessor. This service also deletes the record that contains the associated key token from the PKDS. It also allows the deletion of a retained key in the coprocessor even if there is not a PKDS record, or deletion of a PKDS record for a retained key even if the coprocessor holding the retained key is not online. Use the *rule_array* parameter specifying the FORCE keyword and serial number of the coprocessor that contains the retained key to be deleted. If a PKDS record exists for the same label, but the serial number does not match the serial number in rule_array, the service will fail. If any applications still need the public key, use public key extract to create a public key token prior to deletion of the retained key.

The callable service name for AMODE(64) invocation is CSNFRKD.

Format

```
CALL CSNDRKD(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_label)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords supplied in the *rule_array* parameter. The value may be 0 or 2.

rule_array

Direction	Type
Input	Character String

This parameter may be FORCE and the coprocessor serial number.

key_label

Direction	Type
Input	String

A 64-byte label of a key that has been retained in a coprocessor.

Usage notes

ICSF calls the Security Server (RACF) to check authorization to use the Retained Key Delete service and the label of the key specified in key_label.

Retained private keys are domain-specific. Only the LPAR domain that created a Retained private key can delete the key via the Retained Key Delete service.

When a Retained key is deleted using the Retained Key Delete service, ICSF records this event in a type 82 SMF record with a subtype of 15.

If the Retained key does not exist in the coprocessor and the PKDS record exists and the domain that created the retained key matches the domain of the requester, ICSF deletes the PKDS record. This situation may occur if the coprocessor has been zeroized through TKE or the service processor.

If a PKDS record containing the retained key exists but the coprocessor holding the retained key is not online, ICSF deletes the PKDS record if the FORCE keyword is specified. The serial number specified in the rule array must be the serial number of the coprocessor where the Retained key was created. The key token in the PKDS record contains this serial number, and the serial number is used to verify that the PKDS record can be deleted.

If the retained key exists on the coprocessor but there is no corresponding PKDS record, ICSF deletes the retained key from the coprocessor if the FORCE keyword is specified.

Access control point

The **Retained Key Delete** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 480. Retained Key Delete required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	

Table 480. Retained Key Delete required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Retained Key List (CSNDRKL and CSNFRKL)

Use the Retained Key List callable service to list the key labels of those keys that have been retained within all current active coprocessors.

The callable service name for AMODE(64) invocation is CSNFRKL.

Format

```
CALL CSNDRKL(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_label_mask,
             retained_keys_count,
             key_labels_count,
             key_labels)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords supplied in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	Character String

This parameter is ignored by ICSF.

key_label_mask

Direction	Type
Input	String

A 64-byte key label mask that is used to filter the list of key names returned by the verb. You can use a wild card (*) to identify multiple keys retained within the coprocessor.

Note: If an asterisk (*) is used, it must be the last character in key_label_mask. There can only be one *.

retained_keys_count

Direction	Type
Output	Integer

An integer variable to receive the number of retained keys stored within all active coprocessor.

key_labels_count

Direction	Type
Input/Output	Integer

On input this variable defines the maximum number of key labels to be returned. On output this variable defines the total number of key labels returned. The maximum value for this field is 100. The value returned in the *retained_keys_count* variable can be larger if you have not provided for the return of a sufficiently large number of key labels in the *key_labels_count* field.

key_labels

Direction	Type
Output	String

A string variable where the key label information will be returned. This field must be at least 64 times the key label count value. The key label information is a string of zero or more 64-byte entries. The first 64-byte entry contains a coprocessor serial number, and is followed by one or more 64-byte

entries that each contain a key label of a key retained within that coprocessor. The format of the first 64-byte entry is as follows:

```
 /nnnnnnnnbbbbb...bbb
where
 "/" is the character "/" (EBCDIC: X'61')
 "nnnnnnnn" is the 8-byte cryptographic coprocessor serial number
 "bbbbb...bbb" is 55 bytes of blank pad characters
   (EBCDIC: X'40')
```

This information (64-byte card serial number entry followed by one or more 64-byte label entries) is repeated for each active coprocessor that contains retained keys that match the *key_label_mask*. All data returned is EBCDIC characters. The number of bytes of information returned is governed by the value specified in the *key_labels_count* field. The *key_labels* field must be large enough to hold the number of 64-byte labels specified in the *key_labels_count* field plus one 64-byte entry for each active coprocessor (a maximum of 64 coprocessors).

Usage notes

Not all platforms support multiple coprocessors. In the case where only one card is supported, the *key_labels* field will contain one or more 64-byte entries that each contain a key label of a key retained within the coprocessor. There will be no 64-byte entry or entries containing a coprocessor serial number.

ICSF calls RACF to check authorization to use the Retained Key List service.

ICSF caller must be authorized to the *key_label_mask* name including the *.

Retained private keys are domain-specific. ICSF lists only those keys that were created by the LPAR domain that issues the Retained Key List request.

Access control points

The **Retained Key List** access control point controls the function of this service.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 481. Retained Key List required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor	
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	

Table 481. Retained Key List required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Chapter 14. Key data set management

ICSF provides key stores for symmetric and asymmetric operational key tokens. Symmetric key tokens (AES, DES and HMAC) are stored in the Cryptographic Key Data Set (CKDS). Asymmetric key tokens (RSA and ECC) and trusted blocks are stored in the Public Key Data Set (PKDS).

In order to store operational key tokens encrypted under a master key in the CKDS or PKDS, the corresponding master key must be loaded into a CCA coprocessor and the coprocessor must be active. Tokens with a key value in the clear do not require a coprocessor to be available or active to store the token in the CKDS or PKDS.

This topic describes the callable services that manage key tokens in the key stores.

- "CKDS Key Record Create (CSNBKRC and CSNEKRC)" on page 1202
- "CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2)" on page 1203
- "CKDS Key Record Delete (CSNBKRD and CSNEKRD)" on page 1205
- "CKDS Key Record Read (CSNBKRR and CSNEKRR)" on page 1207
- "CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)" on page 1208
- "CKDS Key Record Write (CSNBKRW and CSNEKRW)" on page 1213
- "CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)" on page 1215
- "Coordinated KDS Administration (CSFCRC and CSFCRC6)" on page 1217
- "ICSF Multi-Purpose Service (CSFMPS and CSFMPS6)" on page 1221
- "Key Data Set List (CSFKDSL and CSFKDSL6)" on page 1225
- "Key Data Set Metadata Read (CSFKDMR and CSFKDMR6)" on page 1239
- "Key Data Set Metadata Write (CSFKDMW and CSFKDMW6)" on page 1246
- "Key Data Set Record Retrieve (CSFRRT and CSFRRT6)" on page 1252
- "Key Data Set Update (CSFKDU and CSFKDU6)" on page 1254
- "PKDS Key Record Create (CSNDKRC and CSNFKRC)" on page 1257
- "PKDS Key Record Delete (CSNDKRD and CSNFKRD)" on page 1259
- "PKDS Key Record Read and PKDS Key Record Read2 (CSNDKRR or CSNDKRR2 and CSNFKRR or CSNFKRR2)" on page 1261
- "PKDS Key Record Write (CSNDKRW and CSNFKRW)" on page 1263

Metadata for key data set records

Key data sets in the KDSR format have metadata that can be used as search criteria, can be read, and can be added, changed or deleted. The Key Data Set List, Key Data Set Metadata Read, and Key Data Set Metadata Write callable services preform these functions. These services can be used to manage the life cycle of key material.

The following are the metadata that are available with any key data set in any format:

Record creation date
The date and time that the record was created in the KDS.

Record update date
The date and time of the last time that the key material or metadata of the record was changed.

The following are the additional metadata that are available with any key data set in the KDSR format:

Key material validity start date
The date that the key material become active. An SMF record is logged every time an inactive record is referenced.

Note: The earliest valid date is January 1, 1900, and the latest valid date is June 4, 2185.

Key material validity end date
The last date that the key material is active. An SMF record is logged every time an inactive record is referenced.

Note: The earliest valid date is January 1, 1900, and the latest valid date is June 4, 2185.

Last used reference date
The date that the key material was last referenced. The date is dependent on the setting of the KDSREFDAYS option.

Last used service name
The service or utility that referenced the record the last time that the last used reference date was updated.

Last used class reference date
The date that the key was last used by any service in a class of operations. This variable length metadata block has a tag of X'0008'.

Table 482. Format of the metadata block

Offset	Number of bytes	Format	Description
0	2	Binary	Length of this block in bytes.
2	2	Binary	Number of entries in the block. The entry is detailed below.
Class reference date entry.			
0	2	EBCDIC string	Class identifier. **Identifier** **Meaning** **DD** Data decryption operations. **DE** Data encryption operations.
2	8	EBCDIC string	Reference date in format YYYYMMDD.

Record archive flag
When enabled, the key material cannot be used when the record is referenced by an application. An administrative option allows the key material to be used when the record is archived. An SMF record is logged every time an archived record is referenced.

Record archive date
The date that the record archive flag was enabled by the KDS Metadata Write service.

Record recall date
The date that the record archive flag was disabled by the KDS Metadata Write service.

Record prohibit archive flag
When enabled, the record cannot be archived.

Variable-length metadata blocks
Installations can add their own metadata to the record. These metadata blocks can be used as a search criteria and changed or deleted. IBM metadata blocks can be used as search criteria and can be read. IBM metadata blocks cannot be changed by installation applications.

Installation user data
The data stored in the user data field in the old formats of the CKDS (CKDUDATA), PKDS (PKDUDATA), or TKDS (TKDUDATA).

Key fingerprint

A set of identifiers for the key. Different key values are likely to have different identifiers, but this is not guaranteed. This variable length metadata has a tag of X'0005'.

Table 483. Key fingerprint metadata format

Number of bytes	Format	Description
1	binary	Length of this key fingerprint block.
1	binary	Number (*n*) of key fingerprint values.
n type-length-value triplets follow. Each triplet is in the format:		
1	binary	Type of fingerprint: **X'01'** Ciphertext from encrypting a data block of binary zeros in ECB mode. Used for: • CCA fixed-length DES, fixed length AES. • PKCS#11 DES, AES, Blowfish key objects. **X'02'** SHA-1 hash of the public key. Used for all asymmetric keys, PKCS#11 generic secret keys. **X'03'** SHA-256 algorithm. See Appendix E, "Cryptographic algorithms and processes," on page 1631 for more information. Used for CCA non-compliant-tagged AES variable-length keys. **X'04'** SHAVP1 algorithm. See Appendix E, "Cryptographic algorithms and processes," on page 1631 for more information. Used for CCA HMAC variable-length keys. **X'05'** CMACZERO. See "Key Test2 (CSNBKYT2 and CSNEKYT2)" on page 256 for more information. Used for CCA compliant-tagged AES variable-length keys.
1	binary	Length of this type-length-value triplet.
	binary	Fingerprint value.

For example, X'07010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt a data block of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 - 2) and the value is X'010203'.

CKDS Key Record Create (CSNBKRC and CSNEKRC)

Use the CKDS key record create callable service to add a key record to the CKDS that will be used to store AES and DES tokens. The record contains a null key token and is identified by the label passed in the *key_label* parameter. This service updates both the DASD copy of the CKDS currently in use by ICSF and the in-storage copy of the CKDS.

The callable service name for AMODE(64) invocation is CSNEKRC).

Format

```
CALL CSNBKRC(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              key_label)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_label

Direction	Type
Input	Character String

The 64-byte label of a record in the CKDS that is the target of this service. The created record contains a key token set to binary zeros and has a key type of NULL.

Restrictions

The record must have a unique label. Therefore, there cannot be another record in the CKDS with the same label and a different key type.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage notes

The CKDS key record create callable service checks the syntax of the label provided in the *key_label* parameter to ensure that it follows the KGUP rules. To bypass label syntax checking, use a preprocessing exit to turn on the bypass parse bit in the Exit Parameter Control Block (EXPB). For more information about preprocessing exits and the EXPB, refer to the *z/OS Cryptographic Services ICSF System Programmer's Guide*.

You must use either the CKDS key record create callable service or KGUP to create an initial record in the CKDS prior to using the CKDS key record write service to update the record with a valid key token.

Required hardware

No cryptographic hardware is required by this callable service.

CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2)

Use this service to add a key record to the CKDS. The record will contain a null key token, a CCA internal fixed-length or variable-length key token or X9.143 (TR-31) key block of an AES, DES, or HMAC key supplied in the *key_token* parameter. The record is identified by the label passed in the *key_label* parameter. This service updates both the DASD copy of the CKDS currently in use by ICSF and the in-storage copy of the CKDS.

Note: The large common record format (KDSRL) CKDS is required to store X9.143 (TR-31) key blocks.

The callable service name for AMODE(64) is CSNEKRC2.

Format

```
CALL CSNBKRC2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_label,
              key_token_length,
              key_token )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

This parameter is ignored by ICSF.

key_label

Direction	Type
Input	String

The 64-byte label of a record in the CKDS to be created.

key_token_length

Direction	Type
Input	Integer

The length of the field containing the token to be written to the CKDS.

If zero is specified, a null token will be added to the CKDS.

The maximum value is 9992.

key_token

Direction	Type
Input/Output	String

A symmetric internal CCA key token or X9.143 (TR-31) key block to be written to the CKDS if *key_token_length* is non-zero. If the token supplied was encrypted under the old master key, it will be returned encrypted under the current master key.

Restrictions

The record must have a unique label. Therefore, there cannot be another record in the CKDS with the same label and a different key type.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Only secure AES operational keys can be stored in the CKDS with the exception of DATA keys, which can be clear key. Secure and clear HMAC operational keys can be stored in the CKDS.

Usage notes

The service checks the syntax of the label provided in the *key_label* parameter to ensure that it follows the ICSF rules. To bypass label syntax checking, use a preprocessing exit to turn on the bypass parse bit in the Exit Parameter Control Block (EXPB). For more information about preprocessing exits and the EXPB, refer to the *z/OS Cryptographic Services ICSF System Programmer's Guide*.

If ICSF is configured to audit the lifecycle of labels [AUDITKEYLIFECKDS(LABEL(YES),...) is specified] and a secure or variable-length token is being written to the CKDS, a request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the CKDS update.

Required hardware

This service does not use cryptographic hardware during processing, but the corresponding master key must be active to store the token when the key is encrypted under a master key.

CKDS Key Record Delete (CSNBKRD and CSNEKRD)

Use the CKDS key record delete callable service to delete a key record from both the DASD copy of the CKDS and the in-storage copy.

The callable service name for AMODE(64) invocation is CSNEKRD.

Format

```
CALL CSNBKRD(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_label)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords supplied in the rule_array parameter. This number must always be 1.

rule_array

Direction	Type
Input	Character String

The 8 byte keyword that defines the action to be performed. The keyword must be LABEL-DL.

key_label

Direction	Type
Input	Character String

The 64-byte label of a record in the CKDS that is the target of this service. The record pointed to by this label is deleted.

Restrictions

The record defined by the *key_label* must be unique. If more than one record per label is found, the service fails.

This callable service does not support version X'10' external DES key tokens (RKX key tokens) because these tokens are not stored in the CKDS.

Usage notes

Encrypted key tokens cannot be processed when the corresponding master key is not loaded. Clear tokens can be processed on a system without a cryptographic coprocessor or accelerator.

Required hardware

This service does not use cryptographic hardware during processing, but the corresponding master key must be active to store the token when the key is encrypted under a master key.

CKDS Key Record Read (CSNBKRR and CSNEKRR)

Use the CKDS key record read callable service to copy an internal fixed-length AES or DES key token from the in-storage CKDS to application storage. Other cryptographic services can then use the copied key token directly. The key token can also be used as input to the token copying functions of key generate, key import, or secure key import services to create additional keys.

The callable service name for AMODE(64) invocation is CSNEKRR.

Format

```
CALL CSNBKRR(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              key_label,
              key_token)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it indicating specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_label

Direction	Type
Input	Character String

The 64-byte label of a record containing an AES or DES token in the in-storage CKDS. The internal key token in this record is returned to the caller.

key_token

Direction	Type
Output	String

The 64-byte internal key token retrieved from the in-storage CKDS.

Restrictions

The record defined by the *key_label* parameter must be unique and must already exist in the CKDS.

If the internal key token is a clear key token, the token is not returned to the caller unless the caller is in supervisor state or system key.

This callable service does not support version X'10' external DES key tokens (RKX key tokens) because these tokens are not stored in the CKDS.

Required hardware

No cryptographic hardware is required by this callable service.

CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)

Use this callable service to:

- Copy an internal fixed-length or variable-length AES, DES, or HMAC key token from the in-storage CKDS to application storage. Other cryptographic services can then use the copied key token directly.
- Return the protected-key CPACF form of a fixed-length DES, AES, or variable-length AES CIPHER key token.

The callable service name for AMODE(64) is CSNEKRR2.

Format

```
CALL CSNBKRR2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_label,
             key_token_length,
             key_token )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be between 0 or 2, inclusive.

rule_array

Direction	Type
Input	String

The *rule_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 484. Keywords for CKDS Key Record Read2

Keyword	Meaning
Administrative rules (optional)	

Table 484. Keywords for CKDS Key Record Read2 (continued)	
Keyword	**Meaning**
ADMNREAD	The record is being read for administrative purposes. Reference date processing is bypassed. For more information, see the KDSREFDAYS option in *z/OS Cryptographic Services ICSF Administrator's Guide*. This keyword requires the application of the PTF for APAR OA49503 at ICSF FMID HCR77B1 and lower. This is the default.
PROTKEY	The protected-key CPACF form of the secure key token is to be returned. This keyword requires the application of the PTF for APAR OA50450 at ICSF FMID HCR77B1 and lower.
Security checking rules (optional, one keyword)	
BYPAUTH	ICSF will not perform an authorization check against CSFKEYS for the label because an authorized caller has already performed the relevant check. This rule is only allowed with PROTKEY. This keyword requires the application of the PTF for APAR OA50450 at ICSF FMID HCR77B1 and lower.
DSENC	ICSF performs an authorization check against the CSFKEYS class for the label with CRITERIA(SMS=DSENCRYPTION) specified. This check is performed regardless of CHECKAUTH. This rule is only allowed with PROTKEY.

key_label

Direction	Type
Direction	String

The 64-byte label of a record in the CKDS to be retrieved.

key_token_length

Direction	Type
Input/Output	Integer

The length of the buffer for the output key token, key block, or the protected-key CPACF form of the token.

On input, the length of the buffer. The minimum length is 8 bytes and the maximum length is 9992 bytes.

The length specified must be sufficient to contain the data returned.

On output, this parameter will be updated with the length of the token returned in the *key_token* parameter.

key_token

Direction	Type
Output	String

The buffer into which the return key token, key block, or protected-key CPACF form of the token is written.

Restrictions

If the internal key token is a clear key token, the token is not returned to the caller unless the caller is in supervisor state or system key.

Usage Notes

- To retrieve a DES or AES encrypted key from the CKDS in the protected-key CPACF form, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES) and SYMCPACFRET(YES) and the invoker must be an authorized caller (either system key or supervisor state). For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*. The buffer that is used for the key-token should be in storage that will not appear in any dump and is fetch-protected. One way that this can be accomplished is by using storage that is obtained by the IARV64 REQUEST=GETSTOR macro with the DUMP=NO and FPROT=YES options. The protected-key CPACF form of the secure key is sensitive data, and this prevents the key from appearing in system dumps.
- To retrieve a variable-length AES encrypted CIPHER key from the CKDS in the protected-key CPACF form, the following must be true:
  - The key token must allow encryption and decryption, any cipher mode, and export to CPACF protected key format. For information on creating key tokens with the specified attributes, see "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 297. See the CKDS KEYS utility Key Attributes panel for information on the current attributes of a CKDS key.
  - The ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFRET(YES). For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.
  - The invoker must be an authorized caller (either system key or supervisor state).
- The master keys must be loaded when using this service with the PROTKEY rule and specifying the label of an encrypted CCA key token.

Access control points

When the PROTKEY rule is specified with the label of an encrypted CCA key token, the appropriate access control point must be enabled.

Table 485. Required access control points for CKDS Key Record Read2

Key algorithm	**Access control point**
Crypto Express5 and earlier CCA coprocessors	
AES	High-performance secure AES keys
DES	High-performance secure DES keys
Crypto Express6 and later CCA coprocessors	
All	Authenticated Key Export - EXPTSK

For AES CIPHER keys (CEX6C and later), the following access controls must be enabled:

- **Authenticated Key Export - DRVTXKEY**
- **Authenticated Key Export - EXPTSK**
- **Authenticated Key Export - SETSNKEY**

These access controls should be enabled by default, but **Authenticated Key Export - EXPTSK** may not be enabled and will cause return codes that are misleading.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 486. CKDS Key Record Read2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	Crypto coprocessor is required when using the PROTKEY rule with encrypted keys. When using the PROTKEY rule: • DES and AES CIPHER keys and triple-length DES DATA keys with a non-zero control vector are not supported. • Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	When using the PROTKEY rule: • An active crypto coprocessor is required with encrypted keys. • DES and AES CIPHER keys and triple-length DES DATA keys with a non-zero control vector are not supported. • Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor	When using the PROTKEY rule: • An active crypto coprocessor is required with encrypted keys. • DES CIPHER key tokens require the P41458.002 or later MCL. • Triple-length DES DATA keys with a non-zero control vector require the November 2018 or later licensed internal code (LIC). • Compliant-tagged key tokens require the July 2019 or later licensed internal code (LIC). X9.143 key blocks are not supported.

Table 486. CKDS Key Record Read2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	When using the PROTKEY rule: • An active crypto coprocessor is required with encrypted keys. • DES and AES CIPHER keys and triple-length DES DATA keys with a non-zero control vector are not supported. • Compliant-tagged key tokens are not supported. X9.143 key blocks are not supported.
	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	When using the PROTKEY rule, an active crypto coprocessor is required with encrypted keys. X9.143 key blocks are not supported.
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	When using the PROTKEY rule, an active crypto coprocessor is required with encrypted keys. X9.143 key blocks are not supported.
	Crypto Express8 CCA Coprocessor	X9.143 key blocks support requires the CCA release 8.1 or later licensed internal code (LIC).

CKDS Key Record Write (CSNBKRW and CSNEKRW)

Use the CKDS key record write callable service to write an internal fixed-length AES or DES key token to the CKDS record specified by the *key_label* parameter. The key in the token may be in the clear or encrypted under a master key.

This service updates both the DASD copy of the CKDS currently in use by ICSF and the in-storage copy. The record you are updating must be unique and must already exist in both the DASD and in-storage copies of the CKDS.

The callable service name for AMODE(64) invocation is CSNEKRW.

Format

```
CALL CSNBKRW(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              key_token,
              key_label)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

key_token

Direction	Type
Input/Output	String

The 64-byte internal AES or DES key token that is written to the CKDS.

key_label

Direction	Type
Input	Character String

The 64-byte label of a record in the CKDS that is the target of this service. The record is updated with the AES or DES internal key token supplied in the *key_token* parameter.

Restrictions

The record defined by the *key_label* parameter must be unique and must already exist in the CKDS.

This callable service does not support version X'10' external DES key tokens (RKX key tokens) because these tokens are not stored in the CKDS.

Usage notes

Tokens can only be overwritten by a token of the same key algorithm (AES or DES). Clear key tokens can only be overwritten by clear key tokens and encrypted key tokens can only be overwritten by encrypted key tokens.

Encrypted key tokens cannot be processed when the corresponding master key is not loaded. Clear tokens can be processed on a system without a cryptographic coprocessor.

You may use this service with the CKDS key record create callable service to write an initial record to key storage. Use it following the key import and key generate callable services to write an operational key imported or generated by these services directly to the CKDS.

You may use the CKDS Key Record Create2 service to create a record and write a token in one call.

If ICSF is configured to audit the lifecycle of labels [AUDITKEYLIFECKDS(LABEL(YES),...) is specified] and a secure or variable-length token is being written to the CKDS, a request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the CKDS update.

Required hardware

This service does not use cryptographic hardware during processing, but the corresponding master key must be active to store the token when the key is encrypted under a master key.

CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)

Use the CKDS Key Record Write2 callable service to write an internal fixed-length or variable-length AES, DES, or HMAC CCA key token or X9.143 (TR-31) key block to the CKDS record specified by the *key_label* parameter. This service updates both the DASD copy of the CKDS currently in use by ICSF and the in-storage copy. The record you are updating must be unique and must already exist in both the DASD and in-storage copies of the CKDS.

Note: The large common record format (KDSRL) CKDS is required to store X9.143 (TR-31) key blocks.

The callable service name for AMODE(64) is CSNEKRW2.

Format

```
CALL CSNBKRW2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_token_length,
              key_token,
              key_label )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

This parameter is ignored by ICSF.

key_token_length

Direction	Type
Input	Integer

The length in bytes of the token to be written to the CKDS. The maximum value is 9992.

key_token

Direction	Type
Input/Output	String

An internal symmetric key token or key block to be written to the CKDS. If the token or block supplied was encrypted under the old master key, the token or block will be returned encrypted under the current master key.

key_label

Direction	Type
Input	String

The 64-byte label of a record in the CKDS to be overwritten.

Restrictions

The record defined by the *key_label* parameter must be unique and must already exist in the CKDS.

This callable service does not support version X'10' external DES key tokens (RKX key tokens) because these tokens are not stored in the CKDS.

Only secure AES operational keys can be stored in the CKDS with the exception of DATA keys, which can be clear key. Secure and clear HMAC operational keys can be stored in the CKDS.

Usage notes

Tokens can only be overwritten by a token of the same key algorithm (AES or DES). Clear key tokens can only be overwritten by clear key tokens and encrypted key tokens can only be overwritten by encrypted key tokens.

Encrypted key tokens cannot be processed when the corresponding master key is not loaded. Clear tokens can be processed on a system without a cryptographic coprocessor.

If ICSF is configured to audit the lifecycle of labels [AUDITKEYLIFECKDS(LABEL(YES),...) is specified] and a secure or variable-length token is being written to the CKDS, a request is made to the Crypto Express coprocessor to generate the key fingerprint to be used for auditing the CKDS update.

Required hardware

This service does not use cryptographic hardware during processing, but the corresponding master key must be active to store the token when the key is encrypted under a master key.

Coordinated KDS Administration (CSFCRC and CSFCRC6)

Use the Coordinated KDS Administration callable service to perform one of the following functions:

- A coordinated KDS refresh.
- A coordinated KDS master key change.
- A conversion to a KDS capable of reference date tracking (KDSR or KDSRL format).
- A conversion from KDSR to KDSRL format.

Coordinated KDS refresh and conversion from KDSR to KDSRL are only supported for the CKDS and PKDS. Coordinated KDS refresh is not supported for TKDS. There is no need to convert to KDSRL format for the TKDS as the two formats (KDSR and KDSRL) are synonymous.

The callable service name for AMODE(64) invocation is CSFCRC6.

Format

```
CALL CSFCRC (
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             function,
             new_data_set_name,
             data_set_type,
             backup_data_set_name,
             archive_data_set_name,
             feedback_length,
             feedback )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the `rule_array` parameter. The value must be 0.

rule_array

Direction	Type
Input	String

This parameter is ignored.

function

Direction	Type
Input	Integer

The function to be performed by this callable service. The value must be 1 for coordinated change master key or 2 for coordinated refresh or 3 for coordinated conversion. The coordinated refresh function is only available for the CKDS and PKDS. Coordinated refresh is not supported by the TKDS. Coordinated change master key and coordinated conversion are available for the CKDS, PKDS, and TKDS.

Note: Attempting to convert from KDSR format to KDSRL format with the TKDS will be ignored as there is no conversion to be done.

new_data_set_name

Direction	Type
Input	String

The name of the new data set to be used by the CRC callable service. For coordinated set master key this data set will be used to reencipher the active KDS data set, and will become the active KDS data set. For coordinated refresh this data set will become the active KDS dataset. For coordinated conversion this data set will be used to convert the active KDS data set to the new KDS format, and will become the active KDS data set. This data set name must be a 44 character string with the data set name left justified and padded with blanks.

Note: The installation options dataset must be updated with the `new_data_set_name` in order for ICSF to use it in case of a future restart.

data_set_type

Direction	Type
Input	Integer

The type of data set to be processed by the callable service. This value must be 1 for a CKDS, 2 for PKDS, or 3 for TKDS.

backup_data_set_name

Direction	Type
Input	String

The name of the backup data set to be used by this callable service when performing a coordinated change master key. This parameter is optional. If specified, a backup copy of the reenciphered or converted KDS will be stored in this data set. This data set name must be a 44-character string with the data set name left justified and padded with blanks.

archive_data_set_name

Direction	Type
Input	String

The name of the archive data set to be used by the CRC callable service. This parameter is optional. If specified, the active KDS will be renamed to this data set name after performing the coordinated change master key or coordinated refresh or coordinated conversion to a new data set. This data set name must be a 44-character string with the data set name left justified and padded with blanks. The CRC service will take the suffix (usually .D or .DATA /.I or .INDEX) from the active KDS and apply them to the archive data set name. If the data or index name contains no suffix, or if the suffix applied to the archive data set name exceeds 44 characters, the request will be rejected.

feedback_length

Direction	Type
Input	Integer

The length of the feedback field used by the callable service. The recommended length is 32 bytes.

feedback

Direction	Type
Output	String

A field provided by the caller for the callable service to return additional feedback in.

Usage notes

One or more ICSF instances sharing the same active KDS in a sysplex create a KDS sysplex cluster. All KDS sysplex cluster members must be IPLed and started in order to perform a coordinated refresh or coordinated change master key operation. The Coordinated KDS Administration functions will not be queued for processing on inactive sysplex cluster members.

SAF will be invoked to verify the caller is authorized to use this callable service. The CSFCRC resource in the CSFSERV class protects access to this callable service. To access this service, callers will be required to have a UACC of READ for the CSFCRC resource.

The coordinated refresh function is only available for the CKDS and PKDS. This function is not supported for the TKDS. The coordinated change master key function is supported for the CKDS, PKDS, and TKDS. Attempting to convert from KDSR format to KDSRL format with the TKDS will be ignored as there is no conversion to be done.

A coordinated refresh on the active KDS requires KDS updates to be suspended. A refresh of the active KDS is only required when a utility (such as KGUP) has altered the KDS VSAM dataset outside of ICSF. Updates must be suspended in this case to allow the in-storage cache of the KDS VSAM data set to be rebuilt and loaded by ICSF.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 487. Coordinated KDS Administration required hardware

Server	Required cryptographic hardware for Coordinated Change Master Key	Required cryptographic hardware for Coordinated Refresh and Conversion
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor or Crypto Express5 Enterprise PKCS #11 coprocessor	None.
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor, Crypto Express6 CCA Coprocessor, Crypto Express5 Enterprise PKCS #11 coprocessor, or Crypto Express6 Enterprise PKCS #11 coprocessor	None.

Table 487. Coordinated KDS Administration required hardware (continued)

Server	Required cryptographic hardware for Coordinated Change Master Key	Required cryptographic hardware for Coordinated Refresh and Conversion
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express5 Enterprise PKCS #11 coprocessor Crypto Express6 Enterprise PKCS #11 coprocessor Crypto Express7 Enterprise PKCS #11 coprocessor	None.
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express6 Enterprise PKCS #11 coprocessor Crypto Express7 CCA Coprocessor Crypto Express7 Enterprise PKCS #11 coprocessor Crypto Express8 CCA Coprocessor Crypto Express8 Enterprise PKCS #11 coprocessor	

ICSF Multi-Purpose Service (CSFMPS and CSFMPS6)

Use the ICSF Multi-Purpose callable service to:

- Validate the keys in the active CKDS or PKDS.
- Prior to a change master key operation, detect keys that may cause a change master key operation to fail.
- Refresh some options in the installation options data set.

The callable service name for AMODE(64) invocation is CSFMPS6.

Format

```
CALL CSFMPS (
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             data_set_name,
             reserved1_length,
             reserved1,
             reserved2_length,
             reserved2,
             reserved3_length,
             reserved3 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 1 or 2.

rule_array

Direction	Type
Input	Character

The *rule_array* parameter contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 488. Keywords for ICSF Multi-Purpose Service

Keyword	Meaning
Key data set (One required if operation is VALIDATE. Valid only with the VALIDATE keyword.)	
CKDS	Specifies that the active CKDS is to be used.
PKDS	Specifies that the active PKDS is to be used.

Table 488. Keywords for ICSF Multi-Purpose Service (continued)	
Keyword	**Meaning**
Operation (One required)	
REFRESH	Specifies that an installation options data set refresh operation is to be performed. The installation options data set specified in the ICSF startup procedure on the CSFPARM DD statement will be used. Option parameters supported are: • AUDITKEYLIFECKDS • AUDITKEYLIFEPKDS • AUDITKEYLIFETKDS • AUDITKEYUSGCKDS • AUDITKEYUSGPKDS • AUDITPKCS11USG • BEGIN • CHECKAUTH • DEFAULTWRAP • END • FIPSMODE • KEYARCHMSG • KDSREFDAYS • MASTERKCVLEN • MAXSESSOBJECTS • RNGCACHE • SERVICELIBS • SERVSCSFMOD0 • SERVSIEALNKE • SSM • USERPARM • WAITLIST
VALIDATE	Specifies that the KDS validate operation is to be performed. The VALIDATE operation performs an integrity check on the active KDS to detect keys that may cause a change master key operation to fail. Perform the VALIDATE function on the system running the highest level of licensed internal code (LIC).

data_set_name

Direction	Type
Input	String

This field is ignored.

reserved1_length

Direction	Type
Input	Integer

The value must be zero.

reserved1

Direction	Type
Output	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

The value must be zero.

reserved2

Direction	Type
Output	String

This field is ignored.

reserved3_length

Direction	Type
Input	Integer

The value must be zero.

reserved3

Direction	Type
Output	String

This field is ignored.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 489. ICSF Multi-Purpose Service required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor	The VALIDATE rule requires the Nov. 2014 or later licensed internal code (LIC).
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor	The VALIDATE rule requires the Nov. 2014 or later licensed internal code (LIC).
	Crypto Express6 CCA Coprocessor	

Table 489. ICSF Multi-Purpose Service required hardware (continued)		
Server	**Required cryptographic hardware**	**Restrictions**
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express8 CCA Coprocessor	

Key Data Set List (CSFKDSL and CSFKDSL6)

Use the key data set list callable service to generate a list or count of CKDS and PKDS labels or TKDS object handles. The list can be refined by search criteria for metadata or key algorithm. The list can display the key labels in addition to the metadata associated with each label.

The common record format (KDSR or KDSRL) of the key data sets contains metadata that can be used for the search criteria. The older key data set formats have only the record create and update dates available. When the search criteria contains metadata that is not supported by the format of key data set, the service returns a return code 4 and does not generate a list or count. See "Coordinated KDS Administration callable service (CSFCRC and CSFCRC6)" on page 58 to convert your key data sets to a common record format.

For the CKDS and PKDS, the label is a character string up to 64 bytes long. Wild cards are allowed in the filter for labels. See the Usage Notes for information. The search criteria is applied to records that match the label filter.

For the TKDS, the name is the 32-byte name of a token. Wild cards are not allowed. The search criteria is applied to the objects of the token specified. Only PKCS #11 objects have metadata. While a PKCS #11 token has a record in the TKDS, it does not have metadata.

Tokens in the TKDS cannot be listed by using this service. The token record list service is used to generate a list of tokens.

The callable service name for AMODE(64) is CSFKDSL6.

Format

```
CALL CSFKDSL (
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              label_filter_length,
              label_filter,
              search_criteria_length,
              search_criteria,
              label_count,
              output_list_length,
              output_list,
              reserved1_length,
              reserved1,
              reserved2_length,
```

```
                reserved2,
                continuation_area )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. The value can be 3 or 4.

rule_array

Direction	Type
Input	Character

The *rule_array* parameter contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left justified in its own 8-byte location and padded on the right with blanks.

Table 490. Keywords for KDS list control information

Keyword	Meaning
Key data set (Required)	

Table 490. Keywords for KDS list control information (continued)	
Keyword	**Meaning**
CKDS	Specifies that the active CKDS is to be searched.
PKDS	Specifies that the active PKDS is to be searched.
TKDS	Specifies that the active TKDS is to be searched.
Key algorithm (Optional)	
DES	Specifies that DES key algorithms are to be added to the search criteria. Only valid with CKDS.
AES	Specifies that AES key algorithms are to be added to the search criteria. Only valid with CKDS.
PKA	Specifies that PKA key algorithms are to be added to the search criteria. Only valid with PKDS. Labels with either RSA or ECC key algorithms are returned.
Output format (Required)	
LABELS	Specifies that a list of labels that meet the search criteria is to be returned.
COUNT	Specifies that a count of labels that meet the search criteria is to be returned.
DETAILED	Specifies that a list of labels that meet the search criteria is to be returned along with the metadata associated with each label.
REPORT-A	Specifies that the labels that meet the search criteria are to be returned in the format of REPORT-A. See the *output_list* parameter for the format.
State of record (Required)	
ACTIVE	Specifies that only records that are not archived and within the start/end dates if specified are checked. If the start and end dates have not been set for a record, the record is considered active.
INACTIVE	Specifies that only records that are not archived and not within the start/end dates are checked.
ARCHIVED	Specifies that only records that have been archived are checked.
ALL	Specifies that all records are checked.

label_filter_length

Direction	**Type**
Input	Integer

The *label_filter_length* parameter specifies the length in bytes of the *label_filter* parameter.

For the CKDS and PKDS, the value can be between zero and 80 inclusive. When the length is zero, no filtering is used.

For the TKDS, the value must be 32.

label_filter

Direction	**Type**
Input	Character

The *label_filter* parameter contains the information that is used to filter on the key data set records by label or handle.

For the CKDS and PKDS, the filter is for the 64-byte label. Wild cards are allowed. Blank characters are not allowed. See the Usage Notes for details.

For the TKDS, the filter must be the 32-byte name of a token. Wild cards are not allowed. Trailing blanks are allowed. The search criteria is applied to the objects of the specified token.

search_criteria_length

Direction	Type
Input	Integer

The *search_criteria_length* parameter is the length in bytes of the *search_criteria* parameter. The value can be zero if no search criterion is to be applied. The maximum value is 500.

search_criteria

Direction	Type
Input	String

The *search_criteria* parameter is a list of search criterion to be applied to the search. The *search_criteria* is a block of entries in contiguous storage.

Each list entry identifies a single search criterion and any additional data. All of the criterion in the list is applied to each record matching the label filter in the key data set. Each entry in the list uses the structure that is shown in Table 491 on page 1228.

Table 491. Search criteria entry

Offset (Decimal)	Number of bytes	Description
0	2	Search criterion: **Value** **Meaning** **X'0001'** Criterion is for a date. **X'0002'** Criterion is a metadata tag. **X'0003'** Criterion is a TKDS object type. **X'0004'** Criterion is a CKDS key type. **X'0005'** Criterion is a metadata flag. **X'0006'** Criterion is an unsupported CCA key. **X'0007'** Criterion is a weak CCA key.
2	2	Length of structure. The length includes the search criterion and this length field.

Table 491. Search criteria entry (continued)		
Offset (Decimal)	**Number of bytes**	**Description**
4		Additional information (Required): • For dates, see Table 492 on page 1229. • For metadata tag, see Table 493 on page 1230. • For TKDS object type, see Table 494 on page 1231. • For CKDS key type, see Table 495 on page 1231. • For metadata flags, see Table 496 on page 1233. • For unsupported CCA keys, see Table 497 on page 1234. • For weak CCA keys, see Table 498 on page 1235.

- To use a date as a search criterion, select the date type you want to use, a comparison operator, and a date in YYYYMMDD format or binary zero. Binary zero means that the date has not been set.
- When searching for dates equal to zero, the record matches if the date is zero or the metadata block is not present. The date the record is archived or recalled is stored in a metadata block.
- When searching for dates being less than, or less than or equal to a non-zero date, the record matches only if there is a non-zero date for that type in the record.
- Searching for dates being less than or equal to zero are treated in the same manner as a request for dates equal to zero.

Table 492. Search criteria with date tag			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
0	2	1	Search criterion is date.
2	2	14	Length of the entry.
4	1		Date type: **Value** **Date** **X'01'** Date the record was created. **X'02'** Date the record was last updated. **X'03'** Last date the record was referenced. **X'04'** Date the record was archived. **X'05'** Key material validity start date. **X'06'** Key material validity end date. **X'07'** Date the record was recalled.

Table 492. Search criteria with date tag (continued)

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
5	1		Date comparison: **Value** **Meaning** **X'01'** Dates that are less than the specified date. **X'02'** Dates that are greater than the specified date. **X'03'** Dates that are equal to the specified date. **X'04'** Dates that are less than or equal to the specified date. **X'05'** Dates that are greater than or equal to the specified date.
6	8		Date in YYYYMMDD format (EBCDIC numeric characters) or binary zero.

To use a metadata tag as a search criterion, select a valid tag.

Table 493. Search criteria with metadata tag

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
0	2	2	Search criterion is metadata tag.
2	2	6	Length of the entry.
4	2		Metadata tag: **Value** **Meaning** **X'0001'** Installation user data. **X'0002'** Service that referenced the record. **X'0003'** Record archive date. **X'0004'** Record recall date. **X'0005'** Key fingerprint. **X'0006'** Retained RSA key information. **X'0007'-X'7FFF'** Reserved for IBM use. **X'8000'-X'FFFF'** Installation metadata.

To use a TKDS object type as a search criterion, specify the type of objects required.

Table 494. Search criteria with TKDS object type

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
0	2	3	Search criterion is TKDS object type.
2	2	5	Length of the entry.
4	1		Object type: **Value (EBCDIC)** **Meaning** **'T'** The key material of the object is in the clear. **'Y'** The key material of the object is wrapped by the EP11 master key.

To use a CKDS key type as a search criterion, specify the type of keys required.

Table 495. Search criteria with CKDS key type

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
0	2	4	Search criterion is CKDS key type.
2	2	12	Length of the entry.

Table 495. Search criteria with CKDS key type (continued)

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
4	8		CKDS key type in the label (bytes 65-72): **ACIPHER** DES ANSI X9.17 CIPHER keys (deprecated). **ADATA** DES ANSI X9.17 DATA keys (deprecated). **AKEK** DES ANSI X9.17 key-encrypting keys (deprecated). **AMAC** DES ANSI X9.17 MAC keys (deprecated). **CIPHER** AES and DES data-encrypting keys. **CIPHERXI** DES ciphertext translation keys (inbound). **CIPHERXL** DES ciphertext translation keys. **CIPHERXO** DES ciphertext translation keys (outbound). **CV** DES keys with key type listed as CV in the CKDS. Key types include: CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR, DATAC, DATAM, DATAMV, DECIPHER, DKYGENKY, ENCIPHER, IKEYXLAT, KEYGENKY, OKEYXLAT, SECMSG. **CVARDEC** DES Crypto-variable decrypting keys. **CVARENC** DES Crypto-variable encrypting keys. **CVARPINE** DES Crypto-variable PIN encrypting keys. **CVARXCVL** DES Crypto-variable translate keys (CV left). **CVARXCVR** DES Crypto-variable translate keys (CV right). **DATA** AES and DES data-encrypting keys (encrypted and clear). **DATAC** DES data-encrypting keys (double-length with CV). **DATAM** DES MAC keys (double-length with CV). **DATAMV** DES MACVER keys (double-length with CV). **DATAXLAT** DES data-translating keys (deprecated). **DECIPHER** DES data-encrypting keys (decrypt only). **DKYGENKY** AES and DES diversified key-generating keys.

Table 495. Search criteria with CKDS key type (continued)			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
4	8		**ENCIPHER** DES data-encrypting keys (encrypt only). **EXPORTER** AES and DES exporter key-encrypting keys. **IKEYXLAT** DES key-translation keys (inbound). **IMP-PKA** DES limited authority importer key-encrypting keys. **IMPORTER** AES and DES importer key-encrypting keys. **IPINENC** DES PIN encrypting keys (inbound). **KDKGENKY** AES key diversification keys. **KEYGENKY** DES key generating keys. **MAC** AES, DES, and HMAC MAC keys. **MACD** DES double-length MAC key (DATAM). **MACVER** DES and HMAC MAC verification keys. **NULL** Records with no key material. **OKEYXLAT** DES key-translation keys (outbound). **OPINENC** DES output PIN encrypting keys (outbound). **PINCALC** AES PIN calculation keys. **PINGEN** DES PIN generation keys. **PINPROT** AES PIN protection keys. **PINPRW** AES PIN reference value keys. **PINVER** DES PIN verification keys. **SECMSG** AES and DES secure messaging keys. See 'Mappings of TR-31 key usage and mode of use to CCA key types' in *z/OS Cryptographic Services ICSF Administrator's Guide* for a mapping of TR-31 key usages and modes of use to CKDS record types.

To use a metadata flag as a search criterion, specify the flag and value to be used.

Table 496. Search criteria with a metadata flag			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
0	2	5	Search criterion is a metadata flag.
2	2	6	Length of the entry.

Table 496. Search criteria with a metadata flag (continued)			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
4	1	1	Flag type: **Value** **Date** **X'01'** Prohibit archive flag.
5	1		**Value (Decimal)** **State of the flag** **1** Enabled. **0** Disabled.

To use an unsupported CCA key as a search criterion, specify the type of keys required. Any number of bits can be set to 1.

- When bit 0 is set to 1 for the CKDS, all DES ANSI X9.17 keys are listed.
- When bit 1 is set to 1 for the CKDS, all DES DATAXLAT keys are listed.
- When bit 2 is set to 1 for the CKDS, all system keys will be listed.
- When bit 0 is set to 1 for the PKDS, all DSS keys are listed.

Table 497. Search criteria with an unsupported CCA key			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
0	2	6	Search criterion is an unsupported CCA key.
2	2	5	Length of the entry.
4	1		Unsupported keys: **Bit** **Meaning** **0** CKDS: ANSI X9.17 keys PKDS: DSS keys **1** CKDS: DATAXLAT keys PKDS: Reserved **2** CKDS: SYSTEM keys PKDS: Reserved **3-7** Reserved.

To use a weak CCA key as a search criterion, specify the type of keys required.

- When bit 0 is set to 1 for the PKDS, all RSA keys with a modulus size less than 1024 bits are listed.

Table 498. Search criteria with a weak CCA key			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
0	2	7	Search criterion is a weak CCA key.
2	2	5	Length of the entry.
4	1		Weak keys: **Bit** **Meaning** **0** PKDS: RSA keys with a modulus size less than 1024 bits. **1 - 7** Reserved.

label_count

Direction	Type
Output	Integer

The number of labels or handles that are found that match the search criteria. The number of entries in the list in the *output_list* parameter when the LABEL keyword is specified in the rule array. This is the total number of matches found in the KDS when the COUNT keyword is specified in the rule array.

output_list_length

Direction	Type
Input/Output	Integer

The *output_list_length* parameter is the length in bytes of the *output_list* parameter. On input, the value is the size of the buffer for the output. On output, the value is the actual length of the data that is returned in the *output_list* parameter.

This parameter is ignored when the COUNT keyword is specified in the rule array.

output_list

Direction	Type
Output	Character

The output area for the list of labels or handles meeting the search criteria.

When the output format rule array keyword is LABELS, the labels are returned in an array where each entry is a fixed length as follows:

- CKDS - 72 bytes
- PKDS - 64 bytes
- TKDS - 44 bytes

When the DETAILED rule is specified, the output area data is formatted as shown in Table 499 on page 1236 for both the CKDS and the PKDS.

Table 499. Output area data when DETAILED is specified

Offset (Decimal)	Number of bytes	Field name	Description
0	64	Key label	The label that is used to identify the key in the CKDS and PKDS.
64	8	Label type	Key type. Blank for the PKDS.
72	8	Key algorithm	Encrypting algorithm of the key. **Contained in the CKDS** DES/AES/HMAC. **Contained in the PKDS** RSA/ECC/DSS/TBLK. **When the key algorithm rule is not specified, DETAILED returns:** HMAC/DSS/TBLK. This field is left justified and padded with blanks after the value.
80	8	Creation date	Date of the key's creation, expressed as *YYYYMMDD*.
88	8	Creation time	Time of the key's creation, expressed as Coordinated Universal Time (UTC) in the format *hhmmssth*.
96	8	Update date	Date that this key was last updated, expressed as *YYYYMMDD*.
104	8	Update time	Time that this key was last updated, expressed as Coordinated Universal Time (UTC) in the format *hhmmssth*.

When the output format rule array keyword is REPORT-A, the labels are returned in an array where each entry has the format shown in Table 500 on page 1236. All data returned are EBCDIC characters. When the format of the key data set is not the KDSR, all records will be active.

Table 500. Output area data when REPORT-A is specified

Offset	Length	Description
0	1	Status of the record. **Value** **Meaning** **A** Active. **B** Archived. **C** Pre-active. **D** Deactivated.
1	CKDS - 72 bytes PKDS - 64 bytes TKDS - 44 bytes	Label.

The number of labels that are returned is determined by the *output_list_length* parameter and the length of the label by KDS. The *label_count* parameter contains the number of labels in this list.

If there is not enough room to fit all the labels, the return code is 4 and the reason code is 3303. The *continuation_area* is used to continue the list for subsequent calls.

This parameter is not returned when the COUNT keyword is specified in the rule array.

Note: The CKDS label consists of a 64-byte label and an 8-byte key type. The key type is used for key separation.

reserved1_length

Direction	Type
Input	Integer

This parameter is reserved. The value must be zero.

reserved1

Direction	Type
Input	String

This parameter is ignored.

reserved2_length

Direction	Type
Input	Integer

This parameter is reserved. The value must be zero.

reserved2

Direction	Type
Input	String

This parameter is ignored.

continuation_area

Direction	Type
Input/Output	String

This parameter is ignored when the COUNT keyword is specified in the rule array.

This is a 100-byte work area that the calling application must supply. The *continuation_area* is a work area that allows the service to be called multiple times to get the complete list of all labels that meet the search criteria.

For the first request, initialize this area to binary zero.

For subsequent requests, your application must not change the data in this string. The return code indicates whether the list is complete.

Usage Notes

SAF is invoked to verify that the caller is authorized to use this callable service. No checking is done of the CSFKEYS or CRYPTOZ profiles.

ICSF system keys in the CKDS will only be listed when specified by the unsupported CCA key search criteria. System keys are keys that were used internally on systems with the Cryptographic Coprocessor

Feature (CCF). These keys are no longer used except for the SYSTEM MAC key, which is used for record authentication for fixed-length format CKDS. System keys cannot be deleted by using CSNBKRD.

Specifying the label filter for the CKDS and PKDS

A label can consist of up to 64 characters. The first character must be alphabetic or a national character (#, $, @). The remaining characters can be alphanumeric, a national character (#, $, @), or a period (.).

The *label_filter* parameter is a character string that can contain the following:

- Character strings containing valid characters for labels.
- Wild cards (* (asterisk)):
  - A wild card means 0 or more characters are to be ignored in the filtering process.
  - The number of characters to be ignored can be specified as *(*nn*), where *nn* is the number (1 –63) of characters to be ignored in the filtering process.
  - You can specify from 0 to 7 wild cards in the filter. When you do not have a wild card in the string, you are checking for the existence of the label in the key data set.
- Blanks are not allowed anywhere in the filter.

Examples:

*****
All labels.

***ABC**
All labels ending with ABC.

DEF
All labels with DEF anywhere within the label.

GHI*
All labels starting with GHI.

JKL*MNO
All labels starting with JKL and ending with MNO.

(20)PQR
All labels with PQR at character 21.

STU*(6)VWX*
All labels starting with STU and with VWX at character 10.

***(15)YZ1*234**
All labels with YZ1 at character 16 and ending with 234.

$5*6*7
All labels that start with $5, have a 6 anywhere in the label, and end with 7.

.APPL*.AES*.*.KEK*.*.
All labels that have APPL followed by anything, AES followed by anything, and KEK followed by anything after an initial prefix before APPL.

Examples of search criteria:

- Dates:
  - Using binary zero as a value to compare against allows a search to find records where a date field has not been set. If the search date is zero:

    Record creation date
    Cannot be zero.

    Record update date
    The record has not been updated.

    Key material validity start date
    The date was not set.

Key material validity end date
The date was not set.

Last used reference date
The record has not been used in a cryptographic operation.

Record archive date
If the record has been archived, the date cannot be zero. If the record has not been archived, the date is zero.

Record recall date
If the record has been recalled, the date cannot be zero. If the record has not been recalled, the date is zero.

Required hardware

No cryptographic hardware is required by this callable service.

Key Data Set Metadata Read (CSFKDMR and CSFKDMR6)

Use the Key Data Set Metadata Read callable service to get metadata of a CKDS, PKDS, or TKDS record. The metadata requested may be one or more types of data.

Note that only PKCS #11 token objects have metadata. While the PKCS #11 token has a record in the TKDS, it does not have metadata.

The following are the metadata that are available with any key data set in any format:

Record creation date
The date and time that the record was created in the KDS.

Record update date
The date and time of the last time that the key material or metadata of the record was changed.

The following are the additional metadata that are available with any key data set in a common record format (KDSR or KDSRL):

Key material validity start date
The date that the key material become active.

Key material validity end date
The last date that the key material is active.

Last reference date
The date that the key material was last referenced. The date is dependent on the setting of the KDSREFDAYS option.

Record archive date
The date that the record archived flag was enabled by the KDS metadata write service.

Record recall date
The date that the record archived flag was disabled by the KDS metadata write service.

User data
The data stored in the user data field in the old formats of the CKDS (CKDUDATA), PKDS (PKDUDATA), or TKDS (TKDUDATA).

Record archived flag
When enabled, the key material cannot be used when the record is referenced by an application.

Record prohibit archive flag
When enabled, the record cannot be archived.

Key fingerprint
A set of identifiers for the key. Different key values are likely to have different identifiers, but that is not guaranteed. See Table 483 on page 1201 for the format of this metadata.

Variable-length metadata blocks
Both IBM and installation tags may be specified.

Note: See "Coordinated KDS Administration callable service (CSFCRC and CSFCRC6)" on page 58 to convert your key data sets to KDSR format.

The callable service name for AMODE(64) is CSFKDMR6.

Format

```
CALL CSFKDMR (
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              record_label,
              metadata_list_length,
              metadata_list,
              output_list_length,
              output_list,
              reserved1_length,
              reserved1,
              reserved2_length,
              reserved2 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1.

rule_array

Direction	Type
Input	Character

The *rule_array* parameter contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 501. Keywords for KDS metadata read control information

Keyword	Meaning
Key data set (Required)	
CKDS	Specifies that the label is in active CKDS.
PKDS	Specifies that the label is in active PKDS.
TKDS	Specifies that the label is in active TKDS.

record_label

Direction	Type
Input	Character

The label or handle of the record to be processed. The length of the label is the maximum label length for the KDS being accessed:

CKDS
72 bytes.

If the last 8 bytes are blank, the service first verifies that the 64 byte label is unique. If the label is not unique, the request fails.

PKDS
64 bytes.

TKDS
44 bytes.

Note: The CKDS label consists of a 64-byte label and an 8-byte key type. The key type is used for key separation.

metadata_list_length

Direction	Type
Input	Integer

The *metadata_list_length* parameter is the length in bytes of the *metadata_list* parameter. The maximum value is 250.

metadata_list

Direction	Type
Input	String

The *metadata_list* parameter is an list of entries identifying the metadata to be read. The *metadata_list* is a block of entries in contiguous storage.

Each entry in the list identifies a specific metadata to be read. Each entry uses the structure shown in Table 502 on page 1242.

Table 502. Metadata entry

Offset (Decimal)	Number of bytes	Description
0	2	Length of this structure. This value includes the length of this field. For variable metadata block requests, the value is 6. For all other requests, the value is 4.
2	2	Type of metadata to be read: **Value** **Meaning** **X'0001'** Variable metadata block. **X'0002'** Record create date. **X'0003'** Record update date. **X'0004'** Key material validity start date. **X'0005'** Key material validity end date. **X'0006'** Last reference date (YYYYMMDD). **X'0007'** Last reference date (first 8 bytes of the value returned by store clock extended instruction). **X'0008'** Record archive date. **X'0009'** Record archive flag. **X'000A'** Record prohibit archive flag. **X'000B'** Record recall date.
4	2	For variable metadata block requests only: The two byte tag of the metadata. For all other requests, this field is not used.

output_list_length

Direction	Type
Input/Output	Integer

The *output_list_length* parameter is the length in bytes of the *output_list* parameter. On input, the value of the size of the *output_list* area. On output, the value is the actual length of the data returned in the *output_list* parameter. The maximum value is 1000.

output_list

Direction	Type
Output	String

The output area for the metadata requested. The output is a list of metadata corresponding to the metadata list. The *output_list* is a block of the output structures in contiguous storage.

Table 503. Output structure for variable metadata block

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
0	2		Length of this block. This value includes the length of this field.
2	2	1	Type of metadata to be read: Metadata block.
4	2		Tag: **Value** **Meaning** **X'0001'** Installation user data. **X'0002'** Service for reference. **X'0003'** Record archive date. **X'0004'** Record recall date. **X'0005'** Key fingerprint. **X'0006'** Retained key information. **X'0008'** Last used class reference date. **X'8000'-X'FFFF'** Installation metadata.
6	2		Length of the metadata at offset 8 in this block. If the length is zero, the metadata tag was not found in the record.
8	Variable		Variable length metadata.

Table 504. Output structure for record create and update dates

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
0	2	20	Length of this block. This value includes the length of this field.
2	2		Type of metadata to be read: **Value** **Meaning** **X'0002'** Record create date. **X'0003'** Record update date.
4	16		Record date (YYYYMMDD HHMMSSTH) in EBCDIC. If the date is not set in the record, the date is binary zero.

Table 505. Output structure for key material validity, archive, recall, and last reference dates

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
0	2	12	Length of this block. This value includes the length of this field.
2	2		Type of metadata to be read: **Value** **Meaning** **X'0004'** Key material validity start date. **X'0005'** Key material validity end date. **X'0006'** Last used reference date (YYYYMMDD). **X'0007'** Last used reference date (STCKE). **X'0008'** Record archive date. **X'000B'** Record recall date.
4	8		For the key material validity, archive, recall and last used reference date, the date is in YYYYMMDD format in EBCDIC. For the last used reference date (STCKE), this is the first 8 bytes of the value returned by store clock extended instruction. If the date is not set in the record, the date is binary zero.

Table 506. Output structure for flags

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
0	2	5	Length of this block. This value includes the length of this field.
2	2		Type of metadata to be read: **Value** **Meaning** **X'0009'** Record archive flag. **X'000A'** Record prohibit archive flag.
4	1		State of the flag: **Value (Decimal)** **Meaning** **0** The flag is disabled. **1** The flag is enabled.

reserved1_length

Direction	Type
Input	Integer

This parameter is reserved. The value must be zero.

reserved1

Direction	Type
Input	String

This parameter is ignored.

reserved2_length

Direction	Type
Input	Integer

This parameter is reserved. The value must be zero.

reserved2

Direction	Type
Input	String

This parameter is ignored.

Usage notes

SAF is invoked to verify that the caller is authorized to use this callable service.

The CSFKEYS profile for the CKDS or PKDS record being processed is not SAF checked. The CRYPTOZ profile for the PKCS #11 token being processed is not SAF checked.

Required hardware

No cryptographic hardware is required by this callable service.

Key Data Set Metadata Write (CSFKDMW and CSFKDMW6)

Use the Key Data Set Metadata Write callable service to add, delete, or modify metadata of a set of records in the active CKDS, PKDS, or TKDS. One or several metadata fields may be processed in one call. Note that only PKCS #11 objects have metadata. While the PKCS #11 token has a record in the TKDS, it does not have metadata.

The results of processing for each label is found in the *results_list*. There will be a return code and reason code for each label in the *label_list*. The *return_code* and *reason_code* parameters return the overall results of processing.

Note: The key data set must be in a common record format (KDSR or KDSRL). See "Coordinated KDS Administration callable service (CSFCRC and CSFCRC6)" on page 58 to convert your key data sets to KDSR format.

The following metadata may be specified for this service:

Key material validity start date
The date that the key material become active.

Key material validity end date
The last date that the key material is active.

Last used reference date
The last date the key material was referenced in a cryptographic operation. Changing this date may change how key life cycle management tools manage the record. It is recommended that due consideration be given to the effects of changing this date.

Record archive flag
When enabled, the key material cannot be used when the record is referenced by an application.

Record prohibit archive flag
When enabled, the record cannot be archived.

Variable-length metadata block
Only installation tags may be used and X'0001' for user data.

Note: The maximum length of all installation metadata is 500 bytes. This includes the tags and the length fields.

The callable service name for AMODE(64) is CSFKDMW6.

Format

```
CALL CSFKDMW (
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              label_count,
              label_list,
              metadata_list_length,
              metadata_list,
              results_list,
              reserved1_length,
              reserved1,
              reserved2_length,
              reserved2 )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

When a list of record labels is processed and all records were processed successfully, the value is zero. If any record fails, the value is 4.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1.

rule_array

Direction	Type
Input	Character

The *rule_array* parameter contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 507. Keywords for KDS metadata write control information

Keyword	Meaning
Key data set (Required)	

Table 507. Keywords for KDS metadata write control information (continued)	
Keyword	**Meaning**
CKDS	Specifies that the active CKDS is to be updated.
PKDS	Specifies that the active PKDS is to be updated.
TKDS	Specifies that the active TKDS is to be updated.

label_count

Direction	Type
Input	Integer

The *label_count* parameter is the number of labels or object handled in the *label_list* parameter and the number entries in *results_list*. If the *label_count* is zero, the *metadata_list* is validated and the return code indicates whether the *metadata_list* is syntactically correct.

label_list

Direction	Type
Input	Character

The list of labels or handles of the records to be processed. The length of the label is the maximum label length for the KDS being accessed:

CKDS
72 bytes.

If the last 8 bytes are blank, the service first verifies that the 64 byte label is unique. If the label is not unique, the request for that record fails.

PKDS
64 bytes.

TKDS
44 bytes.

Note: The CKDS label consists of a 64-byte label and an 8-byte key type. The key type is used for key separation.

metadata_list_length

Direction	Type
Input	Integer

The *metadata_list_length* parameter is the length in bytes of the *metadata_list* parameter. The maximum value is 250.

metadata_list

Direction	Type
Input	String

The *metadata_list* parameter is an list of entries identifying the metadata to be changed. The *metadata_list* is a block of entries in contiguous storage.

Each list entry identifies a specific metadata to be changed. All of the metadata changes are applied to each record identified by a label or handle in the *label_list*. Each entry uses the structure shown in Table 508 on page 1249.

Table 508. Metadata entry		
Offset (Decimal)	**Number of bytes**	**Description**
0	2	Length of this block. This value includes the length of this field.
2	2	Type of metadata to be written: **Value** **Meaning** **X'0001'** Metadata block. **X'0004'** Key material validity start date. **X'0005'** Key material validity end date. **X'0006'** Last used reference date. **X'0009'** Record archive flag. **X'000A'** Record prohibit archive flag.
4	Variable	Data for request.

- For information about the structure for a variable metadata block, see Table 509 on page 1249.
- For information about the structure for key material validity and last reference date, see Table 510 on page 1250.
- For information about the structure for a flag, see Table 511 on page 1250.
- To add an installation variable metadata block, specify a unique installation tag, the length of the metadata, and the metadata. The data can be any form. The data is not checked.
- To change the metadata in a block, specify the installation tag, the length of the new metadata, and the new metadata. The existing block is updated with the new metadata.
- To delete a metadata block, specify the installation tag and a length of zero.

Note: The maximum length of all installation metadata is 500 bytes. This includes the tags and the length fields.

Table 509. Structure for variable metadata block			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
0	2		Length of this block. This value includes the length of this field.
2	2	X'0001'	Type of metadata: Variable metadata block.
4	2		Tag: **Value** **Meaning** **X'0001'** Installation user data. **X'8000'-X'FFFF'** Installation metadata.

Table 509. Structure for variable metadata block (continued)			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
6	2		Length of the metadata at offset 8. If the length is zero and the metadata block exists in the record, metadata tag is deleted.
8	Variable		Metadata to insert in the record.

- To add or change a date, provide a valid date in YYYYMMDD format and EBCDIC numeric characters.
  - The key material validity end date may not be set to a date in the past. Today's date is acceptable.
  - The key material validity start date cannot be after the end date.
  - The earliest start or end date is January 1, 1900, and the latest date is June 4, 2185.
  - The last reference date may not be set to a date in the future. Today's date is acceptable.
- To remove a date, specify a date of binary zero.

Note: If the last reference date is changed by the KDS metadata write service, the variable-length metadata block for the service referenced is changed by the KDS metadata write service.

Table 510. Structure for key material validity and last reference date			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
0	2	12	Length of this block. This value includes the length of this field.
2	2		Type of metadata to write: **Value** **Meaning** **X'0004'** Key material validity start date. **X'0005'** Key material validity end date. **X'0006'** Last reference date.
4	8		Date in YYYYMMDD format in EBCDIC or binary zero.

- To archive a record, set the record archive flag to 1.
- To recall an archived record, set the record archive flag to 0.
- To prevent a record from ever being archived, set the record prohibit archive flag to 1.
- To disable the record prohibit archive flag, set the record prohibit archive flag to 0.

Table 511. Structure for flag			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
0	2	5	Length of this block. This value includes the length of this field.

Table 511. Structure for flag (continued)			
Offset (Decimal)	**Number of bytes**	**Value (Decimal)**	**Description**
2	2		Type of metadata to write: **Value** **Meaning** **X'0009'** Record archive flag. **X'000A'** Record prohibit archive flag.
4	1		Value flag is to be set to 0 or 1 (decimal).

results_list

Direction	Type
Output	String

The output area for the results of processing each entry in the *label_list*. The output is an array of return and reason codes, 4 bytes for each code. The array returned will be 8 times the *label_count* bytes long.

reserved1_length

Direction	Type
Input	Integer

This parameter is reserved. The value must be zero.

reserved1

Direction	Type
Input	String

This parameter is ignored.

reserved2_length

Direction	Type
Input	Integer

This parameter is reserved. The value must be zero.

reserved2

Direction	Type
Input	String

This parameter is ignored.

Usage notes

SAF is invoked to verify that the caller is authorized to use this callable service.

The CSFKEYS profile for the CKDS or PKDS record being processed is not SAF checked. The CRYPTOZ profile for the PKCS #11 token being processed is not SAF checked.

CKDS system keys do not have metadata.

Required hardware

No cryptographic hardware is required by this callable service.

Key Data Set Record Retrieve (CSFRRT and CSFRRT6)

Use the Key Data Set Record Retrieve callable service to retrieve a KDSR format record from a CKDS, PKDS, or TKDS.

This service is intended for diagnostic purposes only.

The callable service name for AMODE(64) is CSFRRT6.

Format

```
CALL CSFRRT (
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              kds_type,
              record_label,
              record_length,
              record_buffer )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended that you specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

This field is ignored by ICSF.

kds_type

Direction	Type
Input	Integer

The type of KDS to which this request applies:

1
CKDS

2
PKDS

3
TKDS

record_label

Direction	Type
Input	String

The label or handle of the record to be read from in the KDS. The length is 72 bytes for CKDS, 64 bytes for PKDS, and 44 bytes for TKDS. It must be left-justified and padded on the right with blanks. The CKDS label consists of a 64-byte label and an 8-byte key type. The key type is used for key separation.

record_length

Direction	Type
Input/Output	Integer

On input, this is the amount of space provided in the *record_buffer*. On output, this is the size, in bytes, of the KDSR record returned in *record_buffer*. It is recommended to specify the maximum LRECL of 32756 on input.

record_buffer

Direction	Type
Output	String

The KDSR record for the requested label.

Usage Notes

1. It is recommended that the CSFSERV profile for CSFRRT be defined with UACC(NONE) and no users are permitted access as it is intended for diagnostic purposes only.
2. This service does not perform SAF authorization checks against key labels or handles (SAF classes CSFKEYS and CRYPTOZ). Therefore, any user ID that is permitted to use this service is able to retrieve any KDS record.

Required hardware

This service does not use cryptographic hardware during processing.

Key Data Set Update (CSFKDU and CSFKDU6)

The Key Data Set Update service is intended to be used in a Geographically Dispersed Parallel Sysplex (GDPS) environment to propagate an update to a CKDS, PKDS, or TKDS from one sysplex to another sysplex. The ICSF key data sets (KDS) will be defined to GDPS, which will cause VSAM to capture updates to the KDS into a log stream. The log stream is then processed on another sysplex where VSAM will call ICSF exit CSFMIAAX to do the update to the appropriate KDS. The ICSF exit calls this service to make the update.

The callable service name for AMODE(64) is CSFKDU6.

Format

```
CALL CSFKDU (
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              kds_name,
              orig_sys_id,
              record_before_len,
              record_before,
              record_after_len,
              record_after,
              func_req,
              opt_flags,
              exit_code,
              orig_return_code )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended that you specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

This field is ignored.

kds_name

Direction	Type
Input	String

The name of the ICSF CKDS, PKDS, or TKDS that is to be updated. This must match the name of the active CKDS, PKDS, or TKDS, or the request is rejected. The string must be 44 bytes in length, left-justified, and padded on the right with blanks.

orig_sys_id

Direction	Type
Input	String

This is diagnostic data that ICSF records in internal trace entries to assist in debugging. The string must be 8 bytes in length, left-justified, and padded on the right with blanks.

record_before_len

Direction	Type
Input	Integer

The length of the *record_before* parameter. For a create request, this must be 0. For any delete or update requests, this is the length of the KDSR record before the change.

record_before

Direction	Type
Input	String

The KDSR record before the update. For any create requests, this parameter is ignored.

record_after_len

Direction	Type
Input	Integer

The length of the *record_after* parameter. For a delete request, this must be 0.

record_after

Direction	Type
Input	String

The KDSR record after the change. For any delete requests, this parameter is ignored.

func_req

Direction	Type
Input	Integer

The function that is being requested:

Code
Meaning

1
Create

3
Update

5
Delete

Any other function codes are not supported and result in an error.

opt_flags

Direction	Type
Input	Bit string

Option flags. A 32-bit string.

Bit
Meaning when set

0-30
Reserved.

31
Apply update in adaptive mode. When not set, standard mode is used.

exit_code

Direction	Type
Output	Integer

For adaptive mode updates. See the APEX_EXIT_CODE field in the CECMAPEX macro for the mapping of this parameter. A set of flags indicates the detected environment.

orig_return_code

Direction	Type
Output	Integer

An original return code for diagnostic purposes. This is the return code to be used when looking up a return and reason code pair in Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441.

Usage Notes

1. The CSFSERV profile for the CSFKDU service should be defined with UACC(NONE). Extreme care should be taken regarding its use because damage to the KDS may occur if used improperly. It is recommended that only the userid under which the log stream is processed should be given READ access to the CSFSERV CSFKDU profile.
2. The CSFKDU service does not perform SAF authorization checks against key labels or handles (SAF classes CSFKEYS and CRYPTOZ). Therefore, any user ID that is permitted to use this service is able to update any KDS record.
3. When called in standard mode (adaptive option flag off):

   Update
   The before image of the record being updated must match the current image of the record that ICSF has for that label.

   Delete
   The before image of the record being deleted must match the current image of the record that ICSF has for that label.

   Create
   There must not be a record currently defined with the label of the input record.
4. When called in adaptive mode, all updates that worked in standard mode also work in adaptive mode. The following cases also result in success with additional information:

   Update
   When the current record for the input label matches the after image for the update, the request is considered already complete.

   Delete
   When there is no record present for the input label, the delete is considered already complete.

   Create
   When the current record for the input label matches the after image for the create, the request is considered already complete.
5. When ICSF performs the update to the ICSF KDS data sets via the CSFKDU service, the ICSF KDS updates are not captured to a log stream by VSAM. This is necessary to prevent the occurrence where the ICSF KDS updates are being captured from each sysplex and routed to the other. Otherwise, an update would trigger an infinite stream of I/O requests between the two sysplexes.

Required hardware

This service does not use cryptographic hardware during processing.

PKDS Key Record Create (CSNDKRC and CSNFKRC)

Use the PKDS Key Record Create callable service to add a key record to the PKDS that will be used to store ECC, RSA, or QSA tokens, and trusted blocks. The record is identified by the label passed in the *key_label* parameter. The record will contain a null key token or the token supplied in the *token* parameter. This service updates both the DASD copy of the PKDS currently in use by ICSF and the in-storage copy of the PKDS.

The callable service name for AMODE(64) invocation is CSNFKRC.

Format

```
CALL CSNDKRC(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             label,
             token_length,
             token)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. This parameter is ignored by ICSF.

rule_array

Direction	Type
Input	String

This parameter is ignored by ICSF.

label

Direction	Type
Input	String

The label of the record to be created. A 64 byte character string.

token_length

Direction	Type
Input	Integer

The length of the field containing the token to be written to the PKDS. If zero is specified, a null token will be added to the PKDS. The maximum value of *token_length* is the maximum length of a private QSA token.

token

Direction	Type
Input	String

Data to be written to the PKDS if *token_length* is non-zero. An RSA, ECC, or QSA private token in either external or internal format, an RSA, ECC, or QSA public token, or a trusted block.

Usage notes

Encrypted key tokens cannot be processed when the corresponding master key is not loaded. Clear tokens can be processed on a system without a cryptographic coprocessor.

Required hardware

This service does not use cryptographic hardware during processing, but the corresponding master key must be active to store the token when the key is encrypted under a master key.

PKDS Key Record Delete (CSNDKRD and CSNFKRD)

Use the PKDS Key Record Delete callable service to delete a key record containing a PKA key token or trusted block from both the DASD copy of the PKDS and the in-storage copy.

The callable service name for AMODE(64) invocation is CSNFKRD.

Format

```
CALL CSNDKRD(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             label)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. This value must be 0, or 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 512. Keywords for PKDS Key Record Delete

Keyword	Meaning
Deletion Mode (optional) specifies whether the record is to be deleted entirely or whether only its contents are to be erased.	
LABEL-DL	Specifies that the record will be deleted from the PKDS entirely. This is the default deletion mode.

Table 512. Keywords for PKDS Key Record Delete (continued)	
Keyword	**Meaning**
TOKEN-DL	Specifies that the only the contents of the record are to be deleted. The record will still exist in the PKDS, but will contain only binary zeroes.

label

Direction	Type
Input	String

The label of the record to be deleted. A 64 byte character string.

Restrictions

This service cannot delete the PKDS record for a retained key.

Required hardware

No cryptographic hardware is required by this callable service.

PKDS Key Record Read and PKDS Key Record Read2 (CSNDKRR or CSNDKRR2 and CSNFKRR or CSNFKRR2)

Use this callable service to copy a PKA key token or trusted block from the in-storage PKDS to application storage. Other cryptographic services can then use the copied key token directly.

Choosing between CSNDKRR and CSNDKRR2

CSNDKRR and CSNDKRR2 provide identical functions. When choosing which service to use, consider the following:

- CSNDKRR ignores the *rule_array_count* and *rule_array* parameters. The callable service name for AMODE(64) invocation is CSNFKRR.
- CSNDKRR2 processes the *rule_array_count* and *rule_array* parameters. The callable service name for AMODE(64) invocation is CSNFKRR2. This callable service requires the application of the PTF for APAR OA49503 at ICSF FMID HCR77B1 and lower.

Format

```
CALL CSNDKRR(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              label,
              token_length,
              token)
```

```
CALL CSNDKRR2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              label,
```

```
                token_length,
                token)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. This parameter is ignored for CSNDKRR/CSNFKRR. For CSNDKRR2/CSNFKRR2, the value can be 0 or 1.

rule_array

Direction	Type
Input	String

The rule_array contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. This parameter is ignored for CSNDKRR/CSNFKRR.

Table 513. Keywords for PKDS Key Record Read2 control information	
Keyword	**Meaning**
Administrative rules (optional)	
ADMNREAD	The record is being read for administrative purposes. Reference date processing is bypassed. For more information, see the KDSREFDAYS option in *z/OS Cryptographic Services ICSF Administrator's Guide*.

label

Direction	Type
Input	String

The label of the record to be read. A 64 byte character string.

token_length

Direction	Type
Input/Output	Integer

The length of the area to which the record is to be returned. On successful completion of this service, *token_length* will contain the actual length of the record returned.

token

Direction	Type
Output	String

Area into which the returned record will be written. The area should be at least as long as the length specified in the *token_length* parameter.

Required hardware

No cryptographic hardware is required by this callable service.

PKDS Key Record Write (CSNDKRW and CSNFKRW)

Use the PKDS Key Record Write callable service to write a PKA key token or trusted block to the PKDS record specified by the *label* parameter. The key in the token may be in the clear or encrypted under a master key.

This service updates both the DASD copy of the PKDS currently in use by ICSF and the in-storage copy. The record you are updating must be unique and must already exist in both the DASD and in-storage copies of the PKDS.

The callable service name for AMODE(64) invocation is CSNFKRW.

Format

```
CALL CSNDKRW(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             label,
```

```
                token_length,
                token)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in the *rule_array* parameter. Its value must be 0 or 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 514. Keywords for PKDS Key Record Write

Keyword	Meaning
Write Mode (optional) specifies the circumstances under which the record is to be written.	

Table 514. Keywords for PKDS Key Record Write (continued)	
Keyword	**Meaning**
CHECK	Specifies that the record will be written only if a record of type NULL with the same label exists in the PKDS. If such a record exists, ICSF overwrites it. This is the default condition.
OVERLAY	Specifies that the record will be overwritten regardless of the current content of the record. If a record with the same label exists in the PKDS, ICSF overwrites it.

label

Direction	Type
Input	String

The label of the record to be overwritten. A 64 byte character string.

token_length

Direction	Type
Input	Integer

The length of the field containing the token to be written to the PKDS.

token

Direction	Type
Input	String

The data to be written to the PKDS, which is an RSA, ECC, or QSA private token in either external or internal format, an RSA, ECC, or QSA public token, or a trusted block.

Restrictions

This service cannot update a PKDS record for a retained key.

Usage notes

When the CHECK rule array keyword is specified, the PKDS Key Record Write service will only overwrite null tokens and tokens of the same key algorithm. For example, an RSA token cannot overwrite an ECC token.

Encrypted key tokens cannot be processed when the corresponding master key is not loaded. Clear tokens can be processed on a system without a cryptographic coprocessor.

Required hardware

This service does not use cryptographic hardware during processing, but the corresponding master key must be active to store the token when the key is encrypted under a master key.

Chapter 15. Utilities

This topic describes these callable services:

- "Character/Nibble Conversion (CSNBXBC and CSNBXCB)" on page 1267
- "Code Conversion (CSNBXEA and CSNBXAE)" on page 1269
- "Cryptographic Usage Statistic (CSFSTAT and CSFSTAT6)" on page 1271
- "ICSF Query Algorithm (CSFIQA and CSFIQA6)" on page 1273
- "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 1278
- "ICSF Query Facility2 (CSFIQF2 and CSFIQF26)" on page 1316
- "SAF ACEE Selection (CSFACEE and CSFACEE6)" on page 1321
- "X9.9 Data Editing (CSNB9ED)" on page 1322

Note: These services are not dependent on the hardware. They will run on any server.

Character/Nibble Conversion (CSNBXBC and CSNBXCB)

Use these utilities to convert a binary string to a character string (CSNBXBC) or convert a character string to a binary string (CSNBXCB).

These utilities do not support invocation in AMODE(64).

Format

```
CALL CSNBXBC(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              text_length,
              source_text,
              target_text,
              code_table)
```

```
CALL CSNBXCB(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              text_length,
              source_text,
              target_text,
              code_table)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

text_length

Direction	Type
Input/Output	Integer

On input, the *text_length* contains an integer that is the length of the *source_text*. The length must be a positive nonzero value. On output, *text_length* is updated with an integer that is the length of the *target_text*.

source_text

Direction	Type
Input	String

This parameter contains the string to convert.

target_text

Direction	Type
Output	String

The converted text that the callable service returns.

code_table

Direction	Type
Input	String

A 16-byte conversion table. The code table for binary to EBCDIC conversion is X'F0F1F2F3F4F5F6F7F8F9C1C2C3C4C5C6'.

Usage notes

These services are structured differently from the other services. They run in the caller's address space in the caller's key and mode.

ICSF need not be active for you to run either of these services. No pre- or post-processing exits are enabled for these services, and no calls to RACF are issued when you run these services.

Required hardware

No cryptographic hardware is required by this callable service.

Code Conversion (CSNBXEA and CSNBXAE)

Use these utilities to convert ASCII data to EBCDIC data (CSNBXAE) or EBCDIC data to ASCII data (CSNBXEA).

This service uses a fixed table for the conversion. The IBM Unicode Service can be used to convert using any ASCII and EBCDIC code page.

These utilities do not support invocation in AMODE(64).

Format

```
CALL CSNBXAE(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             text_length,
             source_text,
             target_text,
             code_table)
```

```
CALL CSNBXEA(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             text_length,
             source_text,
             target_text,
             code_table)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

text_length

Direction	Type
Input	Integer

The *text_length* contains an integer that is the length of the *source_text*. The length must be a positive nonzero value.

source_text

Direction	Type
Input	String

This parameter contains the string to convert.

target_text

Direction	Type
Output	String

The converted text that the callable service returns.

code_table

Direction	Type
Input	String

A 256-byte conversion table. To use the default code table, you need to pass a full word of hexadecimal zero's. See Appendix F, “EBCDIC and ASCII default conversion tables,” on page 1657 for contents of the default table.

Note: The Transaction Security System code table has 2 additional 8-byte fields that are not used in the conversion process. ICSF accepts either a 256-byte or a 272-byte code table, but uses only the first 256 bytes in the conversion.

Usage notes

These services are structured differently than the other services. They run in the caller's address space in the caller's key and mode. ICSF need not be active for you to run either of these services. No pre- or post-processing exits are enabled for these services, and no calls to RACF are issued when you run these services.

Required hardware

No cryptographic hardware is required by this callable service.

Cryptographic Usage Statistic (CSFSTAT and CSFSTAT6)

Use this service to track cryptographic usage external to the ICSF address space. ICSF increments the cryptographic usage statistics by the indicated amount. For more information on cryptographic usage statistics monitoring, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

The callable service name for AMODE(64) invocation is CSFSTAT6.

Format

```
CALL CSFSTAT(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             stat_data,
             reserved_length,
             reserved)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 1.

rule_array

Direction	Type
Input	Character string

The *rule_array* parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The *rule_array* keywords are:

Table 515. Keywords for Cryptographic Usage update

Keyword	Meaning
Cryptographic Usage Statistic (One, required)	
ENG-CPCF	For use with the ENG cryptographic usage statistic. Indicates that *stat_data* contains a count of the number of CPACF invocations since the last time CSFSTAT was called.
ENG-SOFT	For use with the ENG cryptographic usage statistic. Indicates that *stat_data* contains a count of the number of software cryptographic invocations since the last time CSFSTAT was called.

stat_data

Direction	Type
Input	String

For ENG-CPCF, a 4-byte binary field that contains the count of CPACF cryptographic usage since the last invocation of CSFSTAT.

For ENG-SOFT, a 4-byte binary field that contains the count of software cryptographic usage since the last invocation of CSFSTAT.

reserved_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved

Direction	Type
Ignored	String

This parameter is ignored.

Usage notes

To reduce processor usage of calling ICSF to track cryptographic usage, you should externally collect usage counts and then call CSFSTAT periodically with the accumulated count. After the call to CSFSTAT, any externally collected usage count should be zeroed. Each CSFSTAT call is additive.

For CSFSTAT invocations, the start and end times in the SMF record might not represent the actual interval for which the cryptographic operations were executed. The start and end times only represent the interval in which the counts were reported to ICSF.

Required hardware

No cryptographic hardware is required by this callable service.

ICSF Query Algorithm (CSFIQA and CSFIQA6)

Use this utility to retrieve information about the cryptographic and hash algorithms available. You can control the amount of data that is returned by passing in *rule_array* keywords. Keyword values describe the cryptographic algorithm or hash algorithm you are interested in.

The service returns a table of information in the *returned_data* parameter. A row of data consists of the algorithm name, the algorithm size, whether or not clear or secure keys are supported and what method ICSF will use to satisfy a request - CPU instructions, a cryptographic accelerator, a cryptographic coprocessor, or software. The service updates the *returned_data_length* field with the actual length of the output *returned_data* field.

The callable service name for AMODE (64) invocation is CSFIQA6.

Format

```
CALL CSFIQA(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              returned_data_length,
              returned_data,
              reserved_data_length,
              reserved_data)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in *rule_array*. Value must be 0 or 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 516. Keywords for ICSF Query Algorithm

Keyword	Meaning
ALGORITHM (optional)	
AES	Advanced Encryption Standard - symmetric key algorithm.
DES	Data Encryption Standard - single length symmetric key algorithm.
ECC	Elliptic Curve Cryptography. All curve types.
ECC-BP	Elliptic Curve Cryptography using Brain Pool Curves.
ECC-ED	Elliptic Curve Cryptography using Edwards curves.
ECC-KOB	Elliptic Curve Cryptography using Koblitz curves.
ECC-PRIM	Elliptic Curve Cryptography using NIST approved PRIME curves.
HMAC	FIPS-198 keyed-hash message authentication code algorithm.
KYBER	CRYSTALS-Kyber algorithm.
LI2	CRYSTALS-Dilithium Digital Signature Algorithm.
MDC-2	Modification Detection Code 2 - MDC-2 specifies two encipherments per 8 bytes of input text.
MDC-4	Modification Detection Code 4 - MDC-4 specifies four encipherments per 8 bytes of input text.
MD5	Message Digest 5 - A one way hash algorithm.
RNGL	Random number generate long callable service.
RPMD-160	RIPE MD-160 - A one way hash algorithm.
RSA	Rivest-Shamir-Adleman - public key cryptography algorithm, all usage types.

Table 516. Keywords for ICSF Query Algorithm (continued)	
Keyword	**Meaning**
RSA-GEN	Rivest-Shamir-Adleman - public key cryptography algorithm, key generation.
RSA-KM	Rivest-Shamir-Adleman - public key cryptography algorithm, key management usage.
RSA-SIG	Rivest-Shamir-Adleman - public key cryptography algorithm, signature usage.
SHA-1	Secure Hash Algorithm 1 - A one way hash algorithm.
SHA-2	Secure Hash Algorithm 2 - A one way hash algorithm.
SHA-3	Secure Hash Algorithm 3 - A one way hash and extended-output-function (XOF) algorithm.
TDES	Data Encryption Standard - double and triple length symmetric key algorithm.

returned_data_length

Direction	Type
Input/Output	Integer

The length of the *returned_data* parameter in bytes. On input, this is the size of the storage pointed to by the *returned_data* parameter. On output, this parameter will contain the actual length of the data returned. The minimum value is 32 (one row of data) and the maximum value is 1600.

returned_data

Direction	Type
Output	String

This field will contain the table output from the service. Depending on the contents of rule_array, multiple rows may be returned. One row in the table contains:

Table 517. Output for ICSF Query Algorithm

Offset (hex)	Name	Description
0 (X'0')	Algorithm	An 8-byte EBCDIC character string containing the name of the cryptographic algorithm. The character string is padded on the right with blanks. Possible values are: AES DES (single length DES) ECC-BP (Brainpool) ECC-ED (Edwards) ECC-PRIM (NIST Prime) ECC-KOB (Koblitz) HMAC KYBER (CRYSTALS-Kyber) LI2 (CRYSTALS-Dilithium) MDC-2 MDC-4 MD5 RNGL RPMD-160 RSA-GEN RSA-KM RSA-SIG SHA-1 SHA-2 SHA-3 (including SHAKE) TDES (double and triple length DES)
8 (X'8')	Size	An 8-byte EBCDIC string representing the maximum key, modulus, p value, or hash size. The string is padded with blanks on the right. The size is in bits. This is true for all algorithms except RNGL, CRYSTALS-Dilithium, and CRYSTALS-Kyber. For RNGL, the size is in bytes. For CRYSTALS-Dilithium and CRYSTALS-Kyber, this is the algorithm parameter.
16 (X'10')	Key Security	An 8-byte EBCDIC character string containing the string CLEAR SECURE NA The string is padded on the right with blanks.
24(X'18')	Implementation	An 8-byte EBCDIC character string containing how the algorithm is implemented. The string is padded on the right with blanks. Possible choices are: ACC - Cryptographic Accelerator COP - Cryptographic Coprocessor CPU - CPACF SW - Software

The rows are sorted in the following order:

- Algorithm name - alphabetically A to Z
- Algorithm size - numerically highest to least

- Key security - alphabetically A to Z
- Implementation - alphabetically A to Z

reserved_data_length

Direction	Type
Input	Integer

The length of the *reserved_data* parameter. Currently, the value must be 0.

reserved_data

Direction	Type
Ignored	String

This field is currently not used.

Usage notes

The *rule_array* keyword allows the caller to select how much information is returned. The returned data can describe all cryptographic support on the base system or it can be filtered by an algorithm.

For example, a *rule_array_count* of 0 will return information about all algorithms and key security. A *rule_array_count* of 1 and a keyword of 'AES' will return information about the AES algorithm support, both clear and secure AES keys.

Only cryptographic coprocessors in the active state are queried.

A key security of SECURE implies that both SECURE and CLEAR key versions of the algorithm are supported by the processor or the cryptographic coprocessor.

A key security of SECURE and an implementation of CPU implies that a secure key can be sent to the CCA coprocessor and then converted securely to a protected key and the function is processed in CPACF.

This service lists an algorithm as being supported when the cryptographic coprocessor or accelerator is capable of performing the function. It does not reflect when an algorithm is unavailable because TKE was used to disable the function.

RNGL keyword refers to the Random Number Generate Long (CSFBRNGL) callable service. The following is returned for implementation:

COP
When RNGL is implemented using the RNGL verb in the CCA cryptographic coprocessor or the Random Number Generate function in the Enterprise PKCS #11 coprocessor.

CPU
When RNGL is implemented using CPACF.

SW
When there are no available coprocessors and CPACF support is not present.

When a row of the *returned_data* table contains a Key Security value of SECURE and an Implementation value of CPU, this indicates that the CSNBSYE and CSNBSYD callable services support the use of key labels for encrypted keys stored in the CKDS. In other words, the required functions in ICSF, CPACF and the cryptographic coprocessor are available.

Required hardware

No cryptographic hardware is required by this callable service.

ICSF Query Facility (CSFIQF and CSFIQF6)

Use this utility to retrieve information about ICSF, the cryptographic coprocessors and the CCA code in the coprocessors. This information includes:

- General information about ICSF
- General information about CCA code in a coprocessor
- Export control information from a coprocessor
- Diagnostic information from a coprocessor

Coprocessor information requests may be directed to a specific ONLINE or ACTIVE coprocessor or any ACTIVE coprocessor.

This service has an interface similar to the IBM 4765/4767 service CSUACFQ. Instead of the output being returned in the rule array, there is a separate output area. The format of the data returned remains the same. This service supports a subset of the keywords supported by CSUACFQ. For the same supported keywords, CSFIQF and CSUACFQ return the same coprocessor-specific information. The service returns information elements in the *returned_data* field and updates the *returned_data_length* with the actual length of the output *returned_data* field.

The callable service name for AMODE(64) invocation is CSFIQF6.

Format

```
CALL CSFIQF(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             returned_data_length,
             returned_data,
             reserved_data_length,
             reserved_data)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in *rule_array*. Value must be 1, 2 or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to callable services. The keywords are left-justified in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. Specify one or two of the values in Table 518 on page 1279.

Table 518. Keywords for ICSF Query Facility

Keyword	Meaning
Device (optional) - Parameter is ignored for ICSFOPTN, ICSFSTAT, ICSFST2, and ICSFSP11	
COPROCnn	Specifies the specific coprocessor to execute the request. xx may be 00 through 63 inclusive. This may be the processor number of any coprocessor. The processor number of any accelerator is not supported. If specified with rule STATP11, the processor number must be that of an Enterprise PKCS #11 coprocessor. For all other rules, it must be that of a CCA coprocessor.
ANY	Process request on any ACTIVE cryptographic coprocessor. This is the default.
nnnnnnnn	Specifies the 8-byte serial number of the coprocessor to execute the request. If specified with rule STATP11, the processor number must be that of an Enterprise PKCS #11 coprocessor. For all other rules, it must be that of a CCA coprocessor.
Information to return (required)	
GETCOMPD	Get CCA coprocessor-related compliance data. The output is up to 128 bytes long. The format of the output is shown in Table 519 on page 1281.
ICSFOPTN	Get information related to ICSF options. The output is up to 256 bytes long. The format of the output is shown in Table 520 on page 1283.
ICSFSP11	Get ICSF-related PKCS #11 status information. The output is up to 128 bytes long. The format of the output is shown Table 521 on page 1291.

Table 518. Keywords for ICSF Query Facility (continued)	
Keyword	**Meaning**
ICSFSTAT	Get ICSF-related status information. The output is up to 128 bytes long. The format of the output is shown in Table 522 on page 1291.
ICSFST2	Get additional ICSF-related status information. The output is up to 128 bytes long. The format of the output is shown in Table 524 on page 1295.
MKCVCMAC	Get the key check value for all master key registers where the value is calculated using the CMACZERO algorithm. The output is 192 bytes long. The format of the output is shown in Table 525 on page 1303.
MKCVENCZ	Get the key check value for the DES and RSA master key registers where the value is calculated using the ENC-ZERO algorithm. The output is 48 bytes long. The format of the output is shown in Table 526 on page 1303.
NUM-DECT	Get the number of bytes of storage required for the output of a STATDECT request. The output is a 4-byte binary integer.
SIZEWPIN	Get the number of bytes of storage required for the output of a STATWPIN request. The output is a 4-byte binary integer.
STATAES	Get status information on AES enablement and the AES master key registers. The output is up to 64 bytes long. The format of the output is shown in Table 527 on page 1304.
STATAPKA	Get status information on ECC enablement and the ECC master key registers. The output is up to 64 bytes long. The format of the output is shown in Table 528 on page 1304.
STATCARD	Get coprocessor-related basic status information. The output is up to 128 bytes long. The format of the output is shown in Table 529 on page 1305.
STATCCA	Get CCA-related status information. The output is up to 64 bytes long. The format of the output is shown in Table 530 on page 1306.
STATCCAE	Get CCA-related extended status information. The output is up to 128 bytes long. The format of the output is shown in Table 531 on page 1308.
STATDECT	Get the status of the PIN decimalization tables loaded. The length of the data is 20 bytes per decimalization table. The NUM-DECT option will return the storage required for this option. The maximum length of the data is 2000 bytes. The format of the output is shown in Table 532 on page 1309.
STATDIAG	Get coprocessor-related basic status information. The output is up to 256 bytes long. The format of the output is shown in Table 533 on page 1310.
STATEID	Get coprocessor-related basic status information. The output is up to 32 bytes long. The format of the output is shown in Table 534 on page 1312.
STATEXPT	Get coprocessor-related basic status information. The output is up to 64 bytes long. The format of the output is shown in Table 535 on page 1312.

Table 518. Keywords for ICSF Query Facility (continued)	
Keyword	**Meaning**
STATP11	Get Enterprise PKCS #11 coprocessor-related status information. The output is up to 128 bytes long. The format of the output is shown in Table 536 on page 1313.
STATWPIN	Get the weak PIN table loaded. The output is up to 460 bytes long. The format of the output is shown in Table 537 on page 1314.
WRAPMTHD	Get coprocessor-related default configuration setting for the wrapping method. The output is up to 32 bytes long. The format of the output is shown in Table 538 on page 1315.
Additional Master Key Information (optional) - Rule is only supported with STATCCA or STATCCAE	
MOREMKS	Return additional master key information

returned_data_length

Direction	Type
Input/Output	Integer

The length of the *returned_data* parameter. The value depends on the keyword specified. Allow additional space for future enhancements. On output, this field will contain the actual length of the data returned.

returned_data

Direction	Type
Output	String/Integer

This field will contain the output from the service. The format of the output depends on the *rule_array* keyword. The format of the data is defined in the following tables, which describe the output for each keyword.

When the format is 8-byte elements that contain numbers, those numbers are represented by numeric characters which are left-justified and padded on the right with space characters. For example, a *returned_data* element which contains the number two will contain the character string '2'.

For option NUM-DECT and SIZEWPIN, the output is a 4-byte integer.

The output *returned_data* for the GETCOMPD option is defined in Table 519 on page 1281. The format of the output is shown in the table.

Table 519. Output for option GETCOMPD			
Offset (dec)	**Length**	**Format**	**Description**
0	7	EBCDIC	Coprocessor's secure part number.
7	1	N/A	Reserved.
8	7	EBCDIC	Coprocessor's EC field.
15	1	N/A	Reserved.
16	12	EBCDIC	Coprocessor's serial number header and serial number concatenated.

Table 519. Output for option GETCOMPD (continued)

Offset (dec)	Length	Format	Description
28	16	EBCDIC	Current coprocessor clock time in the format YYYYMMDDHHMMSS . Note that there are two blank spaces at the end.
44	8	EBCDIC	CCA version number.
52	8	EBCDIC	UDX supplied version field (first).
60	8	EBCDIC	UDX supplied version field (second).
68	16	EBCDIC	Local date and time when firmware was built in the format YYYYMMDDHHMMSS . Note that there are two blank spaces at the end.
84	4	Binary	Coprocessor scope action flags.
88	4	Binary	Compliance issues. **Bit** **Meaning when set** **0** A UDX was detected. **1-31** Reserved.
92	4	Binary	Maximum secure log events for this domain.
96	2	Binary	Maximum size of one secure log event in bytes.
98	2	Binary	Domain key derivation function in use.
100	4	Binary	Domain scope action flags. **Bit** **Meaning when set** **0** Domain zeroize is in progress. **1** Domain is transitioning into imprint mode. **2** Domain is in imprint mode. **3** Domain is in a compliance mode. **4** Domain is transitioning out of a compliance mode. **5** Domain is in compliance migration mode. **6-15** Reserved **16** Domain secure logging is enabled. **17** Domain secure log cannot wrap. **18-31** Reserved

Table 519. Output for option GETCOMPD (continued)			
Offset (dec)	**Length**	**Format**	**Description**
104	4	Binary	Domain scope compliance flags. **Bit** **Meaning when set** **0** Domain is in PCI-HSM 2016 compliance mode. **1-31** Reserved.
108	4	Binary	Current count of secure log events for this domain.
112	2	Binary	Owner-ID field 2.
114	2	Binary	Owner-ID field 3.
116	4	Binary	Miniboot versions: • High-order 2 bytes for miniboot 0 version. • Low-order 2 bytes for miniboot 1 version.
120	4	Binary	Adapter type.

The output *returned_data* for the ICSFOPTN keyword is defined in Table 520 on page 1283. The format of the output is shown in the table. The data is in character format (EBCDIC).

Table 520. Output for option ICSFOPTN			
Offset	**Length**	**Description**	**Values**
0	1	Value for CHECKAUTH	**Y** CHECKAUTH(YES) **N** CHECKAUTH(NO)
1	1	Value for COMPAT	**Y** COMPAT(YES) **N** COMPAT(NO) **C** COMPAT(COEXIST)

Table 520. Output for option ICSFOPTN (continued)

Offset	Length	Description	Values
2	2	Value for DEFAULTWRAP	First Character: **O** DEFAULTWRAP(ORIGINAL,...) **E** DEFAULTWRAP(ENHANCED,...) **3** DEFAULTWRAP(WRAPENH3,...) Second Character: **O** DEFAULTWRAP(...,ORIGINAL) **E** DEFAULTWRAP(...,ENHANCED) **3** DEFAULTWRAP(WRAPENH3,...)
4	2	Value for FIPSMODE	First Character: **Y** FIPSMODE(YES,...) **C** FIPSMODE(COMPAT,...) **N** FIPSMODE(NO,...) Second Character: **Y** FIPSMODE(...,FAIL(YES)) **N** FIPSMODE(...,FAIL(NO))
6	1	Reserved	Blanks.
7	1	Value for KEYARCHMSG	**Y** KEYARCHMSG(YES) **N** KEYARCHMSG(NO)
8	1	Value for REASONCODES	**I** REASONCODES(ICSF) **T** REASONCODES(TSS)
9	1	Value for RNGCACHE	**Y** RNGCACHE(YES) **N** RNGCACHE(NO)

Table 520. Output for option ICSFOPTN (continued)

Offset	Length	Description	Values
10	1	Value for SSM	**Y** SSM(YES) **N** SSM(NO)
11	5	Reserved	Blanks.
16	2	Value for SYSPLEXCKDS	First Character: **Y** SYSPLEXCKDS(YES,...) **N** SYSPLEXCKDS(NO,...) Second Character: **Y** SYSPLEXCKDS(...,FAIL(YES)) **N** SYSPLEXCKDS(...,FAIL(NO))
18	2	Value for SYSPLEXPKDS	First Character: **Y** SYSPLEXPKDS(YES,...) **N** SYSPLEXPKDS(NO,...) Second Character: **Y** SYSPLEXPKDS(...,FAIL(YES)) **N** SYSPLEXPKDS(...,FAIL(NO))
20	2	Value for SYSPLEXTKDS	First Character: **Y** SYSPLEXTKDS(YES,...) **N** SYSPLEXTKDS(NO,...) Second Character: **Y** SYSPLEXTKDS(...,FAIL(YES)) **N** SYSPLEXTKDS(...,FAIL(NO))
22	2	Reserved	Blanks.
24	8	Value for CTRACE	CTRACE parmlib member or “TRACE CT”.
32	4	Value for DOMAIN	Printable decimal number – DOMAIN(*xx*). If blank, then no domain was detected.

Table 520. Output for option ICSFOPTN (continued)

Offset	Length	Description	Values
36	9	Value for RPSEC	Reference date update period. Format is *DDDhhmmss*, where: ***DDD*** Days ***hh*** Hours ***mm*** Minutes ***ss*** Seconds
45	9	Value for RISEC	Reference date update interval. Format is *DDDhhmmss*, where: ***DDD*** Days ***hh*** Hours ***mm*** Minutes ***ss*** Seconds
54	10	Value for MAXSESSOBJECTS	Printable decimal number – MAXSESSOBJECTS(*x*).
64	8	Value for USERPARM	String – USERPARM(*value*).
72	54	Value for WAITLIST	CICS WAITLIST dataset name if WAITLIST(*data_set_name*) was specified in the ICSF options dataset. Otherwise, it is blank.
126	3	Value for MASTERKCVLEN	Printable character string - MASTERKCVLEN(*xxx*).
129	2	Value for AUDITKEYLIFECKDS	First Character: **Y** AUDITKEYLIFECKDS(TOKEN(YES),...) **N** AUDITKEYLIFECKDS(TOKEN(NO),...) Second Character: **Y** AUDITKEYLIFECKDS(LABEL(YES),...) **N** AUDITKEYLIFECKDS(LABEL(NO),...)

Table 520. Output for option ICSFOPTN (continued)

Offset	Length	Description	Values
131	2	Value for AUDITKEYLIFEPKDS	First Character: **Y** AUDITKEYLIFEPKDS(TOKEN(YES),...) **N** AUDITKEYLIFEPKDS(TOKEN(NO),...) Second Character: **Y** AUDITKEYLIFEPKDS(LABEL(YES),...) **N** AUDITKEYLIFEPKDS(LABEL(NO),...)
133	2	Value for AUDITKEYLIFETKDS	First Character: **Y** AUDITKEYLIFETKDS(SESSOBJ(YES),...) **N** AUDITKEYLIFETKDS(SESSOBJ(NO),...) Second Character: **Y** AUDITKEYLIFETKDS(TOKENOBJ(YES),...) **N** AUDITKEYLIFETKDS(TOKENOBJ(NO),...)
135	11	Value for AUDITKEYUSGCKDS	First Character: **Y** AUDITKEYUSGCKDS(TOKEN(YES),...) **N** AUDITKEYUSGCKDS(TOKEN(NO),...) Second Character: **Y** AUDITKEYUSGCKDS(LABEL(YES),...) **N** AUDITKEYUSGCKDS(LABEL(NO),...) Characters 3 through 11: Usage interval. Format is *DDDhhmmss*, where: ***DDD*** Days ***hh*** Hours ***mm*** Minutes ***ss*** Seconds

Table 520. Output for option ICSFOPTN (continued)

Offset	Length	Description	Values
146	11	Value for AUDITKEYUSGPKDS	First Character: **Y** AUDITKEYUSGPKDS(TOKEN(YES),...) **N** AUDITKEYUSGPKDS(TOKEN(NO),...) Second Character: **Y** AUDITKEYUSGPKDS(LABEL(YES),...) **N** AUDITKEYUSGPKDS(LABEL(NO),...) Characters 3 through 11: Usage interval. Format is *DDDhhmmss*, where: ***DDD*** Days ***hh*** Hours ***mm*** Minutes ***ss*** Seconds

Table 520. Output for option ICSFOPTN (continued)

Offset	Length	Description	Values
157	12	Value for AUDITPKCS11USG	First Character: **Y** AUDITPKCS11USG(SESSIONOBJ(YES),...) **N** AUDITPKCS11USG(SESSIONOBJ(NO),...) Second Character: **Y** AUDITPKCS11USG(TOKENOBJ(YES),...) **N** AUDITPKCS11USG(TOKENOBJ(NO),...) Third Character: **Y** AUDITPKCS11USG(NOKEY(YES),...) **N** AUDITPKCS11USG(NOKEY(NO),...) Characters 4 through 12: Usage interval. Format is *DDDhhmmss*, where: ***DDD*** Days ***hh*** Hours ***mm*** Minutes ***ss*** Seconds
169	1	Value for CICSAUDIT	**Y** CICSAUDIT(YES) **N** CICSAUDIT(NO)
170	4	Value for COMPLIANCEWARN	First Character: **Y** COMPLIANCEWARN(PCIHSM2016(YES)) **N** COMPLIANCEWARN(PCIHSM2016(NO)) **S** COMPLIANCEWARN(PCIHSM2016(SAF)) Characters 2 through 4 are reserved.

Table 520. Output for option ICSFOPTN (continued)

Offset	Length	Description	Values
174	8	Value for STATS	First Character: **Byte** **Meaning** **0** **Y** Crypto Engine Usage Tracking is enabled (ENG). **N** Crypto Engine Usage Tracking is disabled (ENG). **1** **Y** Crypto Service Usage Tracking is enabled (SRV). **N** Crypto Service Usage Tracking is disabled (SRV). **2** **Y** Crypto Algorithm Usage Tracking is enabled (ALG). **N** Crypto Algorithm Usage Tracking is disabled (ALG). **3-7** Reserved.
182	16	Value for STATSFILTERS	First Character: **Byte** **Meaning** **0** **Y** Task level userid is excluded from the stats aggregation criteria (NOTKUSERID). **N** Task level userid is included in the stats aggregation criteria (NOTKUSERID). **1-15** Reserved.
198	1	Value for SERVICELIBS	**Y** SERVICELIBS(YES) **N** SERVICELIBS(NO)

Table 520. Output for option ICSFOPTN (continued)

Offset	Length	Description	Values
199	8	Value for TRACKCLASSUSAGE	List of the classes of operations enabled for tracking. The values are two-byte character strings. These values are supported: **DD** Data-decryption operations. **DE** Data-encryption operations.
207	48	Reserved	Blanks.

The output *returned_data* for the ICSFSP11 keyword is defined in Table 521 on page 1291. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 521. Output for option ICSFSP11

Element Number	Name	Description
1	P11-MK State	Status of the P11-MK **Number** **Meaning** **0** P11-MK not active **1** P11-MK active
2	FIPS Mode	ICSF PKCS #11 FIPS mode **Number** **Meaning** **0** FIPS no enforcement mode **1** FIPS compatibility mode **2** FIPS mode
3-12	Future use	Currently blanks

The output *returned_data* for the ICSFSTAT keyword is defined in Table 522 on page 1291. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 522. Output for option ICSFSTAT

Element Number	Name	Description
1	FMID	8-byte ICSF FMID.
2	ICSF Status Field 1	Status of ICSF. **Number** **Meaning** **1** ICSF initialized. **3** DES master key valid.

Table 522. Output for option ICSFSTAT (continued)

Element Number	Name	Description
3	ICSF Status Field 2	Status of ICSF. **Number** **Meaning** **1** 64-bit callers supported. **2** 64-bit callers supported and a TKDS has been specified for the storage of persistent PKCS #11 objects.
4	CPACF	CPACF availability. **Number** **Meaning** **1** SHA-1 available only **2** DES/TDES enabled **3** SHA-224 and SHA-256 are available **4** SHA-224 and SHA-256, DES and TDES are available **5** SHA-384 and SHA-512 are available **6** SHA-1, SHA-2, AES, DES, and TDES clear key encryption functions are available. **7** DES and AES secure key encryption functions available. **8** OFB, CFB, and GCM encryption functions are available. **9** DRNG function is available. **10** TRNG, SHA-3, and SHAKE functions are available. **11** Clear and secure key ECC digital signature functions are available.
5	AES	CCA AES availability for clear keys. **Number** **Meaning** **1** AES software only. **3** CPACF instructions.
6	DSA	CCA DSA algorithm availability. **Number** **Meaning** **0** DSA not available.
7	RSA Signature	CCA RSA Signature key length. **Number** **Meaning** **0** RSA not available. **3** RSA 4096 key size.

Table 522. Output for option ICSFSTAT (continued)

Element Number	Name	Description
8	RSA Key Management	CCA RSA Key Management key length. **Number** **Meaning** **0** RSA not available. **3** RSA 4096 key size.
9	RSA Key Generate	CCA RSA Key Generate. **Number** **Meaning** **0** Service not available **2** Service available - 4096 bit modulus.
10	Accelerators	Availability of clear RSA key accelerators. **Number** **Meaning** **0** Not available. **1** At least one available for application use.
11	Accelerator Key Size	Clear key size supported by Accelerators. There must be at least one Accelerator available for use for this field to contain valid information. **Number** **Meaning** **1** RSA-ME key size of 4096, CRT key size of 4096.

Table 522. Output for option ICSFSTAT (continued)

Element Number	Name	Description
12	ICSF Status Field 3	The first character in this string indicates the current Special Secure Mode (SSM) setting: **Number** **Meaning** **0** SSM not allowed. **1** SSM allowed. The second character in this string indicates the current RNG Cache (RNGCACHE) setting: **Number** **Meaning** **0** ICSF is not maintaining a random number cache. **1** ICSF maintains a random number cache. The third character in this string indicates the current CSFSERV prefixing setting: **Number** **Meaning** **0** CSFSERV prefixing is disabled. **1** CSFSERV prefixing is enabled. The fourth character in this string indicates the current CSFKEYS prefixing setting: **Number** **Meaning** **0** CSFKEYS prefixing is disabled. **1** CSFKEYS prefixing is enabled. The fifth character in this string indicates the current SAF conditional access control setting: **Number** **Meaning** **0** SAF conditional access control is disabled. **1** SAF conditional access control is enabled. The sixth character indicates the state of the KGUP CSFKEYS authority control: **Number** **Meaning** **0** The KGUP CSFKEYS authority control is disabled. **1** The KGUP CSFKEYS authority control is enabled. The seventh character indicates the state of the KGUP verb authority control: **Number** **Meaning** **0** The KGUP verb authority control is disabled. **1** The KGUP verb authority control is enabled.

Table 523. Output for option ICSFSTAT (con't)		
Element Number	**Name**	**Description**
12 (con't)	ICSF Status Field 3	The eighth character indicates the state of the WRAPENH3 override control: **Number** **Meaning** **0** The WRAPENH3 override control is disabled. **1** The WRAPENH3 override control is enabled.

The output *returned_data* for the ICSFST2 keyword is defined in Table 524 on page 1295. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 524. Output for option ICSFST2		
Element Number	**Name**	**Description**
1	Version	Version of the ICSFST2 *returned_data*. **Number** **Meaning** **1** Elements 1 through 12 have information. **2** Elements 1 through 13 have information. **3** Elements 1 through 14 have information.
2	FMID	8-byte ICSF FMID.
3	ICSF Status Field 1	Status of ICSF. **Number** **Meaning** **0** DES and AES master keys not valid. **1** DES and/or AES master keys valid.
4	ICSF Status Field 2	Status of ICSF. **Number** **Meaning** **0** PKCS #11 is not available. **1** PKCS #11 is available.
5	ICSF Status Field 3	Status of ICSF. **Number** **Meaning** **1** AES master key not valid. **2** AES master key valid.

Table 524. Output for option ICSFST2 (continued)

Element Number	Name	Description
6	ICSF Status Field 4	Status of ICSF. **Number** **Meaning** **0** CCA secure key AES is not available. **1** CCA secure key AES is available.
7	ICSF Status Field 5	Key Store Policy status. The first character in this string indicates if Key Token Authorization Checking controls have been enabled for the CKDS in either warning or fail mode, and, if so, if the Default Key Label Checking control has also been enabled. The numbers that can appear in the first character of this string are: **Number** **Meaning** **0** Key Token Authorization Checking is not enabled for the CKDS. **1** Key Token Authorization Checking for CKDS is enabled in FAIL mode. **Key Store Policy is active for CKDS.** Default Key Label Checking is not enabled. **2** Key Token Authorization Checking for CKDS is enabled in WARN mode. **Key Store Policy is active for CKDS.** Default Key Label Checking is not enabled. **3** Key Token Authorization Checking for CKDS is enabled in FAIL mode. **Key Store Policy is active for CKDS.** Default Key Label Checking is also enabled. **4** Key Token Authorization Checking for CKDS is enabled in WARN mode. **Key Store Policy is active for CKDS.** Default Key Label Checking is also enabled.

Table 524. Output for option ICSFST2 (continued)

Element Number	Name	Description
		The second character in this string indicates if Duplicate Key Token Checking controls have been enabled for the CKDS. The numbers that can appear in the second character of this string are: **Number** **Meaning** **0** Duplicate Key Token Checking is not enabled for the CKDS. **1** Duplicate Key Token Checking is enabled for the CKDS. **Key Store Policy is active for CKDS.** The third character in this string indicates if Key Token Authorization Checking controls have been enabled for the PKDS in either warning or fail mode, and, if so, if the Default Key Label Checking control has also been enabled. The numbers that can appear in the third character of this string are: **Number** **Meaning** **0** Key Token Authorization Checking is not enabled for the PKDS. **1** Key Token Authorization Checking for PKDS is enabled in FAIL mode. **Key Store Policy is active for PKDS.** Default Key Label Checking is not enabled. **2** Key Token Authorization Checking for PKDS is enabled in WARN mode. **Key Store Policy is active for PKDS.** Default Key Label Checking is not enabled. **3** Key Token Authorization Checking for PKDS is enabled in FAIL mode. **Key Store Policy is active for PKDS.** Default Key Label Checking is also enabled. **4** Key Token Authorization Checking for PKDS is enabled in WARN mode. **Key Store Policy is active for PKDS.** Default Key Label Checking is also enabled.

Table 524. Output for option ICSFST2 (continued)

Element Number	Name	Description
		The fourth character in this string indicates if Duplicate Key Token Checking controls have been enabled for the PKDS. The numbers that can appear in the fourth character of this string are: **Number** **Meaning** **0** Duplicate Key Token Checking is not enabled for the PKDS. **1** Duplicate Key Token Checking is enabled for the PKDS. **Key Store Policy is active for PKDS.** The fifth character in this string indicates if Granular Key Label Access controls have been enabled in WARN or FAIL mode. The numbers that can appear in the fifth character of this string are: **Number** **Meaning** **0** Granular Key Label Access controls are not enabled. **1** Granular Key Label Access control is enabled in FAIL mode **2** Granular Key Label Access control is enabled in WARN mode The sixth character in this string indicates if Symmetric Key Label Export controls have been enabled for AES and/or DES keys. The numbers that can appear in the sixth character of this string are: **Number** **Meaning** **0** Symmetric Key Label Export controls are not enabled. **1** Symmetric Key Label Export control is enabled for DES keys only. **2** Symmetric Key Label Export control is enabled for AES keys only. **3** Symmetric Key Label Export controls are enabled for both DES and AES keys.

Table 524. Output for option ICSFST2 (continued)

Element Number	Name	Description
		The seventh character in this string indicates if PKA Key Management Extensions have been enabled in either WARN or FAIL mode. When PKA Key Management Extensions have been enabled in either WARN or FAIL mode, the seventh character also indicates whether a SAF key ring or a PKCS #11 token is identified as the trusted certificate repository. (The trusted certificate repository is identified using the APPLDATA field of the CSF.PKAEXTNS.ENABLE profile. If no value is specified in the APPLDATA field, a PKCS #11 token is assumed.) The numbers that can appear in the seventh character of this string are: **Number** **Meaning** **0** PKA Key Management Extensions control is not enabled. **1** PKA Key Management Extensions control is enabled in FAIL mode. The trusted certificate repository is a SAF key ring. **2** PKA Key Management Extension control is enabled in FAIL mode. The trusted certificate repository is a PKCS #11 token. **3** PKA Key Management Extensions control is enabled in WARN mode. The trusted certificate repository is a SAF key ring. **4** PKA Key Management Extension control is enabled in WARN mode. The trusted certificate repository is a PKCS #11 token. The eighth character in this string indicates if the Allow Archived Key Use control have been enabled. The numbers that can appear in the eighth character of this string are: **Number** **Meaning** **0** Allow Archived Key Use control is disabled. **1** Allow Archived Key Use control is enabled. This status field continues in element 13.

Table 524. Output for option ICSFST2 (continued)

Element Number	Name	Description
8	ICSF Status Field 6	Status of ICSF. **Number** **Meaning** **1** ECC master key not valid. **3** ECC master key valid, internal and external keys supported.
9	ICSF Status Field 7	Status of ICSF. **Number** **Meaning** **1** RSA master key not valid. **2** RSA master key valid.
10	ICSF Status Field 8	Status of ICSF. **Number** **Meaning** **1** DES master key not valid. **2** DES master key valid.
11	Reserved	This field will always contain the number 1.
12	Reserved	Unpredictable
Version 2 fields		

Table 524. Output for option ICSFST2 (continued)

Element Number	Name	Description
13	ICSF Status Field 5 continued	Key Store Policy status continued from element 7. The first character in this string indicates if the Archived Key for Data Decryption Use control has been enabled. The numbers that can appear in the first character of this string are: **Number** **Meaning** **0** Archived Key for Data Decryption Use control is disabled. **1** Archived Key for Data Decryption Use control is enabled. The second character in this string indicates the state of the CSFKEYS PKA ECC private-key name checking (XFACILIT profile CSF.CSFKEYS.ECC.PRIVATEKEYNAME.ENABLE). **Number** **Meaning** **0** PKA ECC private-key name SAF checking is not enabled. **1** PKA ECC private-key name SAF checking is enabled.
Version 3 fields		

Table 524. Output for option ICSFST2 (continued)

Element Number	Name	Description
14	ICSF status 12 - TR-31 indications	Byte 1 reports which functions are available, based on hardware present: **Number** **Meaning** **0** Only non-HMAC external TR-31 blocks supported. **1** HMAC external TR-31 support is available. **2** Operational TR-31 support is available. Byte 2 reports DES-wrapped operational TR-31 support: **Number** **Meaning** **0** Not supported (byte 1 is '0' or '1'). **1** No CKDS is loaded. **2** The DES MK is not active (or CKDS is empty). **3** The DES CMACZERO VP is not in header. **4** DES-MK-wrapped TR-31 key blocks are supported, but the CKDS is not KDSRL so no label support. **5** DES-MK-wrapped TR-31 blocks are fully supported. Byte 3 reports AES-wrapped operational TR-31 support: **Number** **Meaning** **0** Not supported (byte 1 is '0' or '1'). **1** No CKDS is loaded. **2** The AES MK is not active (or CKDS is empty). **3** The AES CMACZERO VP is not in header. **4** AES-MK-wrapped TR-31 key blocks are supported, but the CKDS is not KDSRL so no label support. **5** AES-MK-wrapped TR-31 blocks are fully supported.

The output *returned_data* for the MKCVCMAC option is defined in Table 525 on page 1303. The format of the data is 8-byte key check value (KCV). The data is shown in 16 hexadecimal digits (EBCDIC)

The CMACZERO key check value is five bytes long. The last three bytes are always zero. When a master key register is empty, the key check value will be zero.

Table 525. Output for option MKCVCMAC

Offset	Length	Description
0	16	KCV of DES old master key
16	16	KCV of DES current master key
32	16	KCV of DES new master key
48	16	KCV of RSA old master key
64	16	KCV of RSA current master key
80	16	KCV of RSA new master key
96	16	KCV of AES old master key
112	16	KCV of AES current master key
128	16	KCV of AES new master key
144	16	KCV of ECC old master key
160	16	KCV of ECC current master key
176	16	KCV of ECC new master key

The output *returned_data* for the MKCVENCZ option is defined in Table 526 on page 1303. The format of the data is 4-byte key check value (KCV). The data is shown in eight hexadecimal digits (EBCDIC).

The ENC-ZERO key check value is three bytes long. The fourth byte is always zero. When a master key register is empty, the key check value will be zero.

Table 526. Output for option MKCVENCZ

Offset	Length	Description
0	8	KCV of DES old master key
8	8	KCV of DES current master key
16	8	KCV of DES new master key
24	8	KCV of RSA old master key
32	8	KCV of RSA current master key
40	8	KCV of RSA new master key

The output *returned_data* for the STATAES keyword is defined in Table 527 on page 1304. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 527. Output for option STATAES

Element Number	Name	Description
1	AES NMK Status	State of the AES new master key register: **Number** **Meaning** **1** Register is clear **2** Register contains a partially complete key **3** Register contains a complete key
2	AES CMK Status	State of the AES current master key register: **Number** **Meaning** **1** Register is clear **2** Register contains a key
3	AES OMK Status	State of the AES old master key register: **Number** **Meaning** **1** Register is clear **2** Register contains a key
4	AES key length enablement	The maximum AES key length that is enabled by the function control vector. The value is 0 (if no AES key length is enabled in the FCV), 128, 192, or 256.

The output *returned_data* for the STATAPKA keyword is defined in Table 528 on page 1304. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 528. Output for option STATAPKA

Element Number	Name	Description
1	ECC NMK status	The state of the ECC new master key register: **Number** **Meaning** **1** Register is clear. **2** Register contains a partially complete key. **3** Register contains a complete key.

Table 528. Output for option STATAPKA (continued)		
Element Number	**Name**	**Description**
2	ECC CMK status	The state of the ECC current master key register: **Number** **Meaning** **1** Register is clear. **2** Register contains a key.
3	ECC OMK status	The state of the ECC old master key register: **Number** **Meaning** **1** Register is clear. **2** Register contains a key.
4	ECC key length enablement	The maximum ECC curve size that is enabled by the function control vector. The value will be 0 (if no ECC keys are enabled in the FCV) and 521 for the maximum size.

The output *returned_data* for the STATCARD keyword is defined in Table 529 on page 1305. The format of the data is 8-byte elements. The format of the data is listed in the table.

Table 529. Output for option STATCARD			
Element Number	**Name**	**Format**	**Description**
1	Number of installed adapters	EBCDIC	The number of active cryptographic coprocessors installed in the machine. This only includes coprocessors that have CCA software loaded (including those with CCA UDX software).
2	DES hardware level	EBCDIC	Version of the DES hardware in the coprocessor.
3	RSA hardware level	EBCDIC	Version of the RSA hardware in the coprocessor.
4	POST Version	EBCDIC	Version of the Power-On Self Test (POST) firmware. The first four characters define the POST0 version and the last four characters define the POST1 version.
5	Coprocessor Operating System Name	EBCDIC	The name of the operating system firmware.
6	Coprocessor Operating System Version	EBCDIC	The version of the operating system firmware on the coprocessor.
7	Coprocessor Part Number	EBCDIC	The eight-character part number identifying the version of the coprocessor.
8	Coprocessor EC Level	EBCDIC	The eight-character EC (engineering change) level for this version of the coprocessor.

Table 529. Output for option STATCARD (continued)			
Element Number	**Name**	**Format**	**Description**
9	Miniboot Version	EBCDIC	The version of the coprocessor's miniboot firmware. This firmware controls the loading of programs into the coprocessor. The first four characters define the MiniBoot0 version and the last four characters define the MiniBoot1 version.
10	CPU Speed	EBCDIC	The operating speed of the microprocessor chip, in megahertz.
11	Adapter ID (Also see element number 15)	Binary	A unique identifier manufactured into the coprocessor. The coprocessor's Adapter ID is an eight-byte binary value.
12	Flash Memory Size	EBCDIC	The size of the flash EPROM memory on the coprocessor, in 64-kilobyte increments.
13	DRAM Memory Size	EBCDIC	The size of the dynamic RAM (DRAM) on the coprocessor, in kilobytes.
14	Battery-Backed Memory Size	EBCDIC	The size of the battery-backed RAM on the coprocessor, in kilobytes.
15	Serial Number	EBCDIC	The unique serial number of the coprocessor. The serial number is factory installed and is also reported by the CLU utility in a coprocessor signed status message.

The output *returned_data* for the STATCCA keyword is defined in Table 530 on page 1306. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 530. Output for option STATCCA		
Element Number	**Name**	**Description**
1	NMK Status	State of the DES New Master Key Register: **First character** **Meaning** **1** Register is clear **2** Register contains a partially complete key **3** Register contains a complete key **Last character** **Meaning (when MOREMKS keyword specified in rule array)** **blank** Register contains a 16-byte key **1** Register contains a 24-byte key

Table 530. Output for option STATCCA (continued)

Element Number	Name	Description
2	CMK Status	State of the DES Current Master Key Register: **First character** **Meaning** **1** Register is clear **2** Register contains a complete key **Last character** **Meaning (when MOREMKS keyword specified in rule array)** **blank** Register contains a 16-byte key **1** Register contains a 24-byte key
3	OMK Status	State of the DES Old Master Key Register: **First character** **Meaning** **1** Register is clear **2** Register contains a complete key **Last character** **Meaning (when MOREMKS keyword specified in rule array)** **blank** Register contains a 16-byte key **1** Register contains a 24-byte key
4	CCA Application Version	The version of the CCA application program that is running in the coprocessor.
5	CCA Application Build Date	The build date for the CCA application program that is running in the coprocessor.
6	User Role	The Role identifier which defines the host application user's current authority.

The output *returned_data* for the STATCCAE keyword is defined in Table 531 on page 1308. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 531. Output for option STATCCAE

Element Number	Name	Description
1	Symmetric NMK Status	State of the DES New Master Key Register: **First character** **Meaning** **1** Register is clear **2** Register contains a partially complete key **3** Register contains a complete key **Last character** **Meaning (when MOREMKS keyword specified in rule array)** **blank** Register contains a 16-byte key **1** Register contains a 24-byte key
2	Symmetric CMK Status	State of the DES Current Master Key Register: **First character** **Meaning** **1** Register is clear **2** Register contains a complete key **Last character** **Meaning (when MOREMKS keyword specified in rule array)** **blank** Register contains a 16-byte key **1** Register contains a 24-byte key
3	Symmetric OMK Status	State of the DES Old Master Key Register: **First character** **Meaning** **1** Register is clear **2** Register contains a complete key **Last character** **Meaning (when MOREMKS keyword specified in rule array)** **blank** Register contains a 16-byte key **1** Register contains a 24-byte key
4	CCA Application Version	The version of the CCA application program that is running in the coprocessor.

Table 531. Output for option STATCCAE (continued)		
Element Number	**Name**	**Description**
5	CCA Application Build Date	The build date for the CCA application program that is running in the coprocessor.
6	User Role	The Role identifier which defines the host application user's current authority.
7	RSA NMK Status	State of the RSA New Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a partially complete key **3** Register contains a complete key
8	RSA CMK Status	State of the RSA Current Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a key
9	RSA OMK Status	State of the RSA Old Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a key

For STATDECT, the output is a table of up to 100 PIN decimalization tables as shown in Table 532 on page 1309. The maximum size is 2000 bytes. The data is in character format (EBCDIC).

Table 532. Output for option STATDECT		
Offset	**Field**	**Description**
0	Number	Numeric character indicating the table number
3	State	Character indicating the state of the table **L** loaded **A** active
4	Table	16-byte decimalization table

The output *returned_data* for the STATDIAG keyword is defined in Table 533 on page 1310. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 533. Output for option STATDIAG

Element Number	Name	Description
1	Battery State	State of the battery on the coprocessor: **Number** **Meaning** **1** Battery is good. **2** Battery should be replaced.
2	Intrusion Latch State	State of the intrusion latch on the coprocessor: **Number** **Meaning** **1** Latch is cleared. **2** Latch is set.
3	Error Log Status	Status of the CCA error log in the coprocessor: **Number** **Meaning** **1** Error log is empty. **2** Error log contains data, but is not yet full. **3** Error log is full.
4	Mesh Intrusion detected	Status of the protective mesh that surrounds the secure module — indicating a probable attempt to physically penetrate the module: **Number** **Meaning** **1** No intrusion detected. **2** Intrusion attempt detected.
5	Low Voltage Detected	Status of the secure module. The value indicates whether the power supply voltage was under the minimum acceptable level. This may indicate an attempt to attack the security module: **Number** **Meaning** **1** Only acceptable voltages have been detected. **2** A voltage has been detected under the low-voltage tamper threshold.

Table 533. Output for option STATDIAG (continued)

Element Number	Name	Description
6	High Voltage Detected	Status of the secure module. The value indicates whether the power supply voltage was higher than the maximum acceptable level. This may indicate an attempt to attack the security module: **Number** **Meaning** **1** Only acceptable voltages have been detected. **2** A voltage has been detected that is higher than the high-voltage tamper threshold.
7	Temperature Range Exceeded	Status of the secure module. The value indicates whether the temperature in the secure module was outside of the acceptable limits. This may indicate an attempt to obtain information from the module: **Number** **Meaning** **1** Temperature is acceptable. **2** Detected temperature is outside an acceptable limit.
8	Radiation Detected	Status of the secure module. The value indicates whether radiation was detected inside the secure module. This may indicate an attempt to obtain information from the module: **Number** **Meaning** **1** No radiation has been detected. **2** Radiation has been detected.
9, 11, 13, 15, 17	Last Five Commands Run	These five elements contain the last five commands that were executed by the coprocessor CCA application. They are in chronological order, with the most recent command in element 9. Each element contains the security API command code in the first four characters and the subcommand code in the last four characters.
10, 12, 14,16, 18	Last Five Return Codes	These five elements contain the return codes and reason codes corresponding to the five commands in elements 9, 11, 13, 15, and 17. l Each element contains the return code in the first four characters and the reason code in the last four characters.

The output *returned_data* for the STATEID keyword is defined in Table 534 on page 1312. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 534. Output for option STATEID		
Element Number	**Name**	**Description**
1	EID	During initialization, a value of zero is set in the coprocessor.

The output *returned_data* for the STATEXPT keyword is defined in Table 535 on page 1312. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 535. Output for option STATEXPT		
Element Number	**Name**	**Description**
1	Base CCA Services Availability	Availability of base CCA services: **Number** **Meaning** **0** Base CCA services are not available. **1** Base CCA services are available.
2	CDMF Availability	Availability of CDMF: **Number** **Meaning** **0** CDMF encryption is not available.
3	56-bit DES Availability	Availability 56-bit DES encryption: **Number** **Meaning** **0** 56-bit DES encryption is not available. **1** 56-bit DES encryption is available.
4	Triple-DES Availability	Availability triple-DES encryption: **Number** **Meaning** **0** Triple-DES encryption is not available. **1** Triple-DES encryption is available.
5	SET Services Availability	Availability of SET (Secure Electronic Transaction) services: **Number** **Meaning** **0** SET Services are not available. **1** SET Services are available.

Table 535. Output for option STATEXPT (continued)

Element Number	Name	Description
6	Maximum Modulus for Symmetric Key Encryption	Maximum modulus size that is enabled for the encryption of symmetric keys. This defines the longest public-key modulus that can be used for key management of symmetric-algorithm keys: **Number** **Meaning** **0** RSA not available. **1024** RSA 1024 key size. **2048** RSA 2048 key size. **4096** RSA 4096 key size.

The output *returned_data* for the STATP11 keyword is defined in Table 536 on page 1313. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 536. Output for option STATP11

Element Number	Name	Description
1	P11 NMK Status	State of the P11 new master key register: **Number** **Meaning** **1** Register is clear **2** Register contains an uncommitted key **3** Register contains a committed key
2	P11 CMK	State of the P11 current master key register: **Number** **Meaning** **1** Register is clear **2** Register contains a key

Table 536. Output for option STATP11 (continued)

Element Number	Name	Description
3	Compliance Mode	Current compliance mode for the coprocessor. An 8-digit hexadecimal number that is the sum of the active compliance modes: **Number** **Meaning** **n** An 8-digit hexadecimal number that is the sum of the active compliance modes: • 00000001 - FIPS 2009 • 00000002 - BSI 2009 • 00000004 - FIPS 2011 • 00000008 - BSI 2011 • 00000040 - BCI/CC 2017
4	Firmware version	Coprocessor PKCS #11 firmware version number as an 8-byte hexadecimal value.
5	Serial Number	A character string containing the unique serial number of the coprocessor. The serial number is factory installed.
6 - 7	PKCS #11 cryptographic coprocessor version information	14-byte character string containing the cryptographic coprocessor level and firmware version of the PKCS#11 coprocessor. The format is the same as D ICSF,CARDS command output (for example, 4.07.29/040E). The string is left-justified and padded with blanks. For a detailed explanation, see the D ICSF,CARDS command in *z/OS Cryptographic Services ICSF System Programmer's Guide*.
8 - 12	Future use	Currently blanks.

For STATWPIN, the output is a table of up to 20 weak PINs. Each entry in the table is formatted as shown in Table 537 on page 1314. The maximum size is 460 bytes. The data is in character format (EBCDIC).

Table 537. Output for option STATWPIN

Offset	Length	Description
0	1	Weak PIN structure type
1	3	Numeric character indicating the table number
4	1	Character indicating the state of the table: **Character** **Meaning** **A** Active **L** Loaded
5	2	PIN length
7	4 to 16	Weak PIN

The output *returned_data* for the WRAPMTHD keyword is defined in Table 538 on page 1315. The format of the data is 8-byte elements. The data is in character format (EBCDIC).

Table 538. Output for option WRAPMTHD

Element Number	Name	Description
1	Internal tokens	Default wrapping method for internal tokens. **Number** **Meaning** **0** Keys will be wrapped with the original method **1** Keys will be wrapped with the enhanced X9.24 method **3** Keys will be wrapped with the enhanced wrapping method version 3.
2	External tokens	Default wrapping method for external tokens. **Number** **Meaning** **0** Keys will be wrapped with the original method **1** Keys will be wrapped with the enhanced X9.24 method **3** Keys will be wrapped with the enhanced wrapping method version 3.

reserved_data_length

Direction	Type
Input	Integer

The length of the *reserved_data* parameter. The value must be 0.

reserved_data

Direction	Type
Input	String

This field is not used.

Usage notes

RACF will be invoked to check authorization to use this service.

PKA key generate available indicates that there is at least one ACTIVE coprocessor.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 539. ICSF Query Facility required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	None.	Keyword GETCOMPD is not supported.
IBM z14 **IBM z14 ZR1**	None.	Keyword GETCOMPD requires a Crypto Express6 CCA Coprocessor.
IBM z15 **IBM z15 T02**	None.	
IBM z16 **IBM z16 A02**	None.	

ICSF Query Facility2 (CSFIQF2 and CSFIQF26)

Use this utility to retrieve status information about the cryptographic environment as currently known to ICSF.

This callable service will:

- NOT be SAF protected.
- NOT make calls to any cryptographic processor
- Return information that can be collected from various ICSF control blocks

The callable service name for AMODE(64) invocation is CSFIQF26.

Format

```
CALL CSFIQF2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             returned_data_length,
             returned_data,
             reserved_data_length,
             reserved_data)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords in the *rule_array*. This field is currently reserved and must be 0.

rule_array

Direction	Type
Ignored	String

Keywords that provide control information to callable services. This field is currently ignored

returned_data_length

Direction	Type
Input/Output	Integer

The length of the *returned_data* parameter in bytes. A minimum value of 11 is required. Specify a length of 22 or greater to receive all the supported data.

returned_data

Direction	Type
Output	String/Integer

This field will contain the output from the service. The service will return only the amount of data specified by the *returned_data_length* field.

The format of the *returned_data* is defined in Table 540 on page 1317.

Table 540. Format of returned ICSF Query Facility 2 data

Bytes	Description
0-7	ICSF FMID.

Table 540. Format of returned ICSF Query Facility 2 data (continued)

Bytes	Description
8	**Bit** **Meaning when set on** **0** Crypto Accelerator available. **1** CCA Coprocessor available. **2** Public Key hardware available. **3** TKDS available. **4** SHA-1 available in CPACF. **5** SHA-224 available in CPACF. **6** SHA-256 available in CPACF. **7** SHA-384 available in CPACF.
9	**Bit** **Meaning when set on** **0** SHA-512 available in CPACF. **1** DES available in CPACF. **2** TDES available in CPACF. **3** AES 128-bit available in CPACF. **4** AES 192-bit available in CPACF. **5** AES 256-bit available in CPACF. **6** AES-GCM available in CPACF. **7** ECC Clear Key hardware available.

Table 540. Format of returned ICSF Query Facility 2 data (continued)

Bytes	Description
10	**Bit** **Meaning when set on** **0** ECC Secure Key hardware available. **1** PKCS #11 Secure Key available. **2** FIPS No Enforcement Mode. **3** FIPS Mode enabled. **4** FIPS Compatibility Mode enabled. **5** Reserved. **6** SHA-3 and SHAKE available. **7** ECC available in CPACF.
11	Reserved.
12-19	System compliance information. Byte 0 **Bit** **Meaning when set on** **0** Compliance mode is active. **1** Compliance migration mode is active. **2-7** Reserved. Bytes 1-6: Reserved. Byte 7 **Bit** **Meaning when set on** **0-6** Reserved. **7** PCI-HSM 2016 compliance mode is active. **Note:** These byte references only relate to the 8-byte structure contained in bytes 12-19.

Table 540. Format of returned ICSF Query Facility 2 data (continued)	
Bytes	**Description**
20	Crypto usage statistics flags. **Bit** **Meaning when set on** **0** Cryptographic engine usage tracking is enabled (ENG). **1** Cryptographic service usage tracking is enabled (SRV). **2** Cryptographic algorithm usage tracking is enabled (ALG). **3** Reserved. **4** Reserved. **5** Reserved. **6** Reserved. **7** Reserved.
21	Supported elliptic curves. **Value (hex)** **Supported curves** **01** secp192r1, secp224r1, secp256r1, secp384r1, secp521r1, brainpoolP160r1, brainpoolP192r1, brainpoolP224r1, brainpoolP256r1, brainpoolP320r1, brainpoolP384r1, brainpoolP512r1. **02** All of the above curves plus: Ed25519, X25519, Ed448, X448, secp256k1. **03** All of the above curves plus: Koblitz secp256k1. Note that some curves require hardware.

reserved_data_length

Direction	Type
Input	Integer

The length of the *reserved_data* parameter. This field is reserved and must be 0.

reserved_data

Direction	Type
Ignored	String

This field is currently not used.

Required hardware

No cryptographic hardware is required by this callable service.

SAF ACEE Selection (CSFACEE and CSFACEE6)

This callable service allows an authorized caller (either system key or supervisor state) to provide an ENVR to use in place of the default ACEE selected for SAF checking.

ICSF invokes RACROUTE to verify access to resources. When an ICSF callable service is invoked directly (not through this service), ICSF allows the ACEE selection to default. The default for RACROUTE is to use the TASK ACEE (TCBSENV) pointer in the TCB.

When there is no TCB (which is the case in SRB mode), or when the TASK ACEE pointer is zero, RACROUTE uses the main ACEE for the address space.

This service affects ACEE selection for all four ICSF classes: CSFSERV, CSFKEYS, XCSFKEY, and CRYPTOZ. It does not change the behavior of installation exits.

The callable service name for AMODE(64) is CSFACEE6.

Format

```
CALL CSFACEE(
             envr,
             service_name,,
             parameters... )
```

Parameters

envr

Direction	Type
Input	String

The ENVR data structure that holds the information used to describe a security environment. This was extracted from an ACEE using **RACROUTE REQUEST=EXTRACT,TYPE=ENVRXTR**.

The calling application is responsible for the integrity and currency of the information contained in the ENVR data structure.

service_name

Direction	Type
Input	String

The name of the ICSF callable service of the form CSFzzz or CSNyzzz. See "ICSF callable services naming conventions" on page 3 for details. The keyword is 8 bytes in length, left justified, and padded on the right with space characters.

This is the name of the entry point you would invoke directly. See "SAF ACEE Selection (CSFACEE and CSFACEE6)" on page 1321 for examples.

All services documented in this documentation are supported with the exception of this service itself.

parameters...

Direction	Type
not applicable	not applicable

The parameters for the callable service specified just as they would normally appear when invoking the service directly.

Usage notes

The parameters specified should match the normal invocation. For example, if the direct call was:

```
CALL CSNEXYZ(parm1, parm2, parm3, parm4);
```

The invocation via this service would be:

```
CALL CSFACEE6(envr, "CSNEXYZ ", parm1, parm2, parm3, parm4);
```

Note: Since the original call (CSNEXYZ) is AMODE(64), the CALL (CSFACEE6) must be as well.

Similarly, if the direct call was:

```
CALL CSFZYX(parm1, parm2, parm3);
```

The invocation via this service would be:

```
CALL CSFACEE(envr, "CSFZYX ", parm1, parm2, parm3);
```

Note: The callable service name can be either of the aliases (CSFzzz or CSNyzzz) for an invocation. If the original call was:

```
CALL CSFZYX(parm1, parm2, parm3);
```

The invocation via this service could be:

```
CALL CSFACEE(envr, "CSNBZYX", parm1, parm2, parm3);
```

Determination of whether a service is in the CICS Wait List is performed before the service name is resolved, so for the purposes of CICS Wait List checking, all calls through this service will be treated as CSFACEE, and not as the service that will eventually be executed. If this is a concern, CSFACEE could be added to the CICS Wait List.

Any environmental or parameters errors that would result in ICSF not invoking the requested service will cause a non-zero return code to be returned in register 15 and a non-zero reason code to be returned in register 0, with the rest of the parameters unchanged from input.

Required hardware

There is no required hardware for this service. See an individual service for specifics related to that service.

X9.9 Data Editing (CSNB9ED)

Use this utility to edit an ASCII text string according to the editing rules of ANSI X9.9-4. It edits the text that the *source_text* parameter supplies according to these rules. The rules are listed here in the order in which they are applied. It returns the result in the *target_text* parameter.

1. This service replaces each carriage-return (CR) character and each line-feed (LF) character with a single-space character.
2. It replaces each lowercase alphabetic character (a through z) with its equivalent uppercase character (A through Z).
3. It deletes all characters other than:
   - Alphabetics A...Z
   - Numerics 0...9
   - Space
   - Comma ,
   - Period .

- Dash -
- Solidus /
- Asterisk *
- Open parenthesis (
- Close parenthesis)

4. It deletes all leading space characters.
5. It replaces all sequences of two or more space characters with a single-space character.

This utility does not support invocation in AMODE(64).

Format

```
CALL CSNB9ED(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             text_length,
             source_text,
             target_text)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

text_length

Direction	Type
Input/Output	Integer

On input, the *text_length* contains an integer that is the length of the *source_text*. The length must be a positive, nonzero value. On output, *text_length* is updated with an integer that is the length of the edited text.

source_text

Direction	Type
Input	String

This parameter contains the string to edit.

target_text

Direction	Type
Output	String

The edited text that the callable service returns.

Usage notes

This service is structured differently from the other services. It runs in the caller's address space in the caller's key and mode.

ICSF need not be active for the service to run. There are no pre-processing or post-processing exits that are enabled for this service. While running, this service does not issue any calls to RACF.

Required hardware

No cryptographic hardware is required by this callable service.

Chapter 16. Trusted interfaces

This topic describes these callable services:

- "PCI Interface (CSFPCI and CSFPCI6)" on page 1327
- "Key Token Wrap (CSFWRP and CSFWRP6)" on page 1325

Key Token Wrap (CSFWRP and CSFWRP6)

The Key Token Wrap callable service is used to unwrap a fixed-length secure key token from the current master key and then rewrap in the system protect key. The protected key is typically used on CPACF operations. This service is intended for operating system code. In addition to requiring permission to the CSFSERV profile for CSFWRP, the caller must also be in supervisor state or system key 0-7.

The callable service name for AMODE(64) is CSFWRP6.

Format

```
CALL CSFWRP(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              input_key_identifier_length,
              input_key_identifier,
              output_key_identifier_length,
              output_key_identifier)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. The value must be 0.

rule_array

Direction	Type
Input	String

This parameter is ignored by ICSF.

input_key_identifier_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *input_key_identifier* parameter. The value must be 64.

input_key_identifier

Direction	Type
Input	String

A fixed length secure data key token to be rewrapped under the system protect key. Only version 4 AES key tokens are currently supported. If the *input_key_identifier* is enciphered under the old master key, you will see return code 0 and a reason code of 0 or 10001 (X'2711') depending on the level of hardware you are running with. ICSF does not return the key token enciphered under the current master key.

output_key_identifier_length

Direction	Type
Input/Output	Integer

On input, this is the length of the buffer to contain the *output_key_identifier*. On output, it contains the length of the *output_key_identifier* that is returned. The value must be at least 64.

output_key_identifier

Direction	Type
Output	String

The encrypted cryptographic key (protected by the corresponding CPACF wrapping key).

Usage notes

The CSFSERV profile for CSFWRP should be defined with UACC(NONE). Only user IDs that run authorized should be permitted to it. The buffer that is used for the key-token should be in storage that will not

appear in any dump and is fetch-protected. One way that this can be accomplished is by using storage that is obtained by the IARV64 REQUEST=GETSTOR macro with the DUMP=NO and FPROT=YES options. The protected-key CPACF form of the secure key is sensitive data, and this prevents the key from appearing in system dumps.

Access control points

This table lists the access control points in the domain role that control the function for this service.

Table 541. Key Token Wrap access control points

Cryptographic hardware	Access control point
Crypto Express5 and earlier CCA coprocessors	High-performance secure AES keys.
Crypto Express6 and later CCA coprocessors	Authenticated Key Export - EXPTSK.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, “CCA release levels,” on page 1723.

Table 542. Key Token Wrap required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	CP Assist for Cryptographic Functions and Crypto Express5 CCA Coprocessor	
IBM z14 **IBM z14 ZR1**	CP Assist for Cryptographic Functions and either a Crypto Express5 CCA Coprocessor or a Crypto Express6 CCA Coprocessor	
IBM z15 **IBM z15 T02**	CP Assist for Cryptographic Functions and either a Crypto Express5 CCA Coprocessor, Crypto Express6 CCA Coprocessor, or a Crypto Express7 CCA Coprocessor	
IBM z16 **IBM z16 A02**	CP Assist for Cryptographic Functions and either a Crypto Express6 CCA, Coprocessor Crypto Express7 CCA, or a Coprocessor Crypto Express8 CCA Coprocessor	

PCI Interface (CSFPCI and CSFPCI6)

Use this callable service to:

- Query the complete list of access control points which may be enabled or disabled.
- Send a request to a specific coprocessor and receive the corresponding response when complete.
- Perform audit uploads.

- Obtain information associated with the cryptographic coprocessor configuration of the system.

This service is synchronous. For the rules that involve sending requests to a specific coprocessor, the return and reason codes reflect the success or failure of sending the request rather than the success or failure of the request itself.

The callable service name for AMODE(64) invocation is CSFPCI6.

Format

```
CALL CSFPCI(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             target_pci_coprocessor,
             target_pci_coprocessor_serial_number,
             request_block_length,
             request_block,
             request_data_block_length,
             request_data_block,
             reply_block_length,
             reply_block,
             reply_data_block_length,
             reply_data_block,
             masks_length,
             masks_data)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. See Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441, for a list of return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. See Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 for a list of reason codes.

exit_data_length

Direction	Type
Input/Output	Integer

The length of the data that is passed to the installation exit. The data is identified in the *exit_data* parameter.

exit_data

Direction	Type
Input/Output	String

The data that is passed to the installation exit.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you are supplying in *rule_array*. The value must be 1 or 2.

rule_array

Direction	Type
Input	String

Keyword that provides control information to callable services. The keyword is left-justified in an 8-byte field and padded on the right with blanks. The keyword must be in contiguous storage. These keywords are mutually exclusive:

Table 543. Keywords for PCI interface callable service

Keyword	Meaning
Operation Requested (required)	
ACPOINTS	Queries the list of CCA access control points which may be enabled or disabled by a TKE user.
APNUM	Specifies the *target_pci_coprocessor* field is to be used to submit a CCA request.
PDECRYPT	This keyword is a request to decrypt a cryptogram received from TKE using the specified secret key, returning the clear value.
PENCRYPT	This keyword is a request to encrypt a cryptogram provided by TKE using the specified secret key, returning the enciphered value.
PKEYGEN	This keyword is a request to generate a symmetric encryption key (secret key) and return the key's value encrypted under the TKE audit upload fixed public key.
Q-APINFO	This keyword is a request to return information associated with the cryptographic coprocessor configuration of the system. Q-APINFO replaces the XCPMASK and QUERYDOM keywords and the returned data includes the information returned by those keywords.
QUERYDOM	This keyword is deprecated because Q-APINFO returns the same information as well as additional information. This keyword is a request to return a 256-bit mask indicating the controlled domain information from the AP facility. See the *masks_data* parameter description for more information.
SERIALNO	Specifies the *target_pci_coprocessor_serial_number* field is to be used to submit a CCA request
XCPMASK	This keyword is deprecated because Q-APINFO returns the same information as well as additional information. This keyword is a request to return both the 64-bit mask indicating which of the cryptographic coprocessors are online and the 64-bit mask indicating which of the cryptographic coprocessors are active. See the *masks_data* parameter description for more information.
XPNUM	Specifies the *target_pci_coprocessor* field is to be used to submit a PKCS #11 request.

Table 543. Keywords for PCI interface callable service (continued)	
Keyword	**Meaning**
XPPOINTS	Queries the list of PKCS #11 access control points which may be enabled or disabled by a TKE user.
Reason (optional)	
DISTREC	This keyword indicates that the operation is being issued in preparation for disaster recovery. ICSF performs limited error checking in the case.

Notes:

1. When the XCPMASK or QUERYDOM keyword is specified, the *request_block_length*, *request_block*, *reply_block_length*, *reply_block*, *request_data_block_length*, *request_data_block*, *reply_data_block_length*, and the *reply_data_block* parameters are ignored on input. The *reply_block_length* and *reply_data_block_length* parameters are set to zero on output.
2. When the Q-APINFO keyword is specified, the *request_block_length*, *request_block*, *reply_block_length*, *reply_block*, *request_data_block_length*, *request_data_block*, and the *reply_data_block* parameters are ignored on input. The *reply_block_length* parameter is set to zero on output.
3. When the PKEYGEN, PDECRYPT, or PENCRYPT keyword is specified, the *request_block_length*, *request_block*, *reply_block_length*, and *reply_block* parameters are ignored on input. The *reply_block_length* is set to zero on output.

target_pci_coprocessor

Direction	Type
Input	Integer

The index of the coprocessor card to which this request is directed. Valid values are between 0 and 63, inclusive.

target_pci_coprocessor_serial_number

Direction	Type
Input/Output	String

The serial number of the coprocessor to which the request is directed. This parameter may be used instead of the target_pci_coprocessor by specifying the SERIALNO rule. The length is 8 bytes. This parameter is updated with the serial number of the card if the request was successfully processed.

request_block_length

Direction	Type
Input	Integer

Length of CPRB and the request block in the *request_block* field. For the APNUM or SERIALNO rules, the maximum length allowed is 5,500 bytes. For the XPNUM rule, the maximum length allowed is 12,000 bytes.

request_block

Direction	Type
Input	String

The complete command or query request for the target coprocessor, including the CPRB.

request_data_block_length

Direction	Type
Input	Integer

Length of request data block in the *request_data_block* field. The maximum length allowed is 6,400 bytes. The length field must be a multiple of 4. For the XPNUM rule, the length must be zero.

request_data_block

Direction	Type
Input	String

The data that accompanies the *request_block* field.

reply_block_length

Direction	Type
Input/Output	Integer

Length of CPRB and the reply block in the *reply_block* field.

- For the APNUM or SERIALNO rules, the maximum length allowed is 5,500 bytes.
- For the XPNUM rule, the maximum length allowed is 12,000 bytes.
- For all other rules, this field is ignored on input and set to zero on output.

This field is updated on output with the actual length of the *reply_block* field.

reply_block

Direction	Type
Output	String

Reply from the target coprocessor. This is the CPRB and reply block that has been processed by the coprocessor.

reply_data_block_length

Direction	Type
Input/Output	Integer

Length of reply block in the *reply_data_block* field.

- For the APNUM or SERIALNO rules, the maximum length allowed is 6,400 bytes and must be a multiple of four.
- For the ACPOINTS rule, the minimum length is 26044 bytes.
- For the PDECRYPT rule, the minimum length is 92 bytes.
- For the PENCRYPT rule, the minimum length is 132 bytes.
- For the PKEYGEN rule, the minimum length is 308 bytes.
- For the Q-APINFO rule, the minimum length is 664 bytes.
- For the XPNUM rule, the length must be zero.
- For the XPPOINTS rule, the minimum length is 2274 bytes.
- For all other rules, this field is ignored on input and set to zero on output.

This field is updated on output with the actual length of the *reply_data_block* field.

reply_data_block

Direction	Type
Output	String

The data that accompanies the *reply_block* field.

For the ACPOINTS and XPPOINTS rules, the output is a series of rows of data (each being a group header or an individual ACP description). Each group header is followed by individual ACP descriptions that belong to that group. The first byte of each row determines which mapping to use (X'01' for group headers and X'02' for individual ACP descriptions).

Table 544. ACP group header format

Offset	Length in bytes	Type	Description
0	1	Binary	Group header identifier (X'01').
1	4	Integer	Group description length (*n*).
5	*n*	ASCII string	Group description.

Table 545. ACP description format

Offset	Length in bytes	Type	Description
0	1	Binary	ACP description identifier (X'02').
1	2	Integer	ACP index.
2	4	Integer	ACP description length (*n*).
6	*n*	ASCII string	ACP description.
6+*n*	4	Bit mask	Flag word. **Bit** **Meaning when set on** **0** ACP is required to be on and cannot be turned off. **1-31** Reserved.
10+*n*	4	Integer	ACP enablement list count (*r*).
14+*n*	2**r*	Integer array	List of ACPs that must be enabled before this ACP can be enabled. Each ACP is a two byte integer.

For the Q-APINFO rule, the output is a structure describing the cryptographic coprocessor configuration of the system at this point in time. Accelerators are not considered in the output.

Table 546. Current cryptographic coprocessor configuration

Offset (dec)	Length in bytes	Type	Description
0	4	ASCII string	Eyecatcher ('QINF').
4	2	Binary	Version (Currently, X'0001').
6	2	Integer	Length of the structure (664).

Table 546. Current cryptographic coprocessor configuration (continued)

Offset (dec)	Length in bytes	Type	Description
8	4	Integer	Maximum number of domains supported.
12	32	Bitmask	Mask of usage domains (0-based).
44	32	Bitmask	Mask of control domains (0-based).
76	4	Integer	Maximum number of cryptographic coprocessors supported.
80	4	Integer	Count of online coprocessors (Number of bits set to 1 in the mask of online coprocessors).
84	32	Bitmask	Mask of online coprocessors.
116	4	Integer	Count of active coprocessors (Number of bits set to 1 in the mask of active coprocessors).
120	32	Bitmask	Mask of active coprocessors.
152	2*256	Array	Array of coprocessor type information. 256 two-byte elements, each one representing a single coprocessor. Each bit set to 1 in the mask of online coprocessors will have non-zero data in this array. • The first byte of each element indicates the coprocessor type: – X'0B' (CEX5S) – X'0C' (CEX6S) – X'0D' (CEX7S) – X'0E' (CEX8S) • The second byte of each element indicates the coprocessor subtype: – X'01' (CCA coprocessor) – X'02' (EP11 coprocessor)

masks_length

Direction	Type
Input	Integer

Length of the reply data being returned in the masks_data field. The length must be 32 bytes for all requests.

masks_data

Direction	Type
Output	String

Masks data is returned only when the input *rule_array* keyword is XCPMASK or QUERYDOM. Both of these rules are deprecated. For all other *rule_array* keywords, binary zeroes are returned.

For the QUERYDOM rule, the returned data indicates a bit mask of the actual Crypto domains that may be controlled from this logical partition. For all other rules, the first 8 bytes indicate the count of the

cards online. The second 8 bytes indicate a bit mask of the actual cards brought online. The third 8 bytes indicate the count of the cards active. The fourth 8 bytes indicate a bit mask of the actual cards that are active.

Usage notes

The *target_pci_coprocessor*, the *target_pci_coprocessor_serial_number*, the *request_block*, the *reply_block*, the *request_data_block*, and the *reply_data_block*, are recorded in SMF Record Type 82, subtype 16.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service. The CCA releases used in the table are described in Appendix L, "CCA release levels," on page 1723.

Table 547. PCI Interface required hardware

Server	Required cryptographic hardware	Restrictions
IBM z13 **IBM z13s**	Crypto Express5 CCA Coprocessor Crypto Express5 Enterprise PKCS #11 Coprocessor	
IBM z14 **IBM z14 ZR1**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express5 Enterprise PKCS #11 Coprocessor Crypto Express6 Enterprise PKCS #11 Coprocessor	
IBM z15 **IBM z15 T02**	Crypto Express5 CCA Coprocessor Crypto Express6 CCA Coprocessor Crypto Express7 CCA Coprocessor Crypto Express5 Enterprise PKCS #11 Coprocessor Crypto Express6 Enterprise PKCS #11 Coprocessor Crypto Express7 Enterprise PKCS #11 Coprocessor	

Table 547. PCI Interface required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z16 **IBM z16 A02**	Crypto Express6 CCA Coprocessor Crypto Express6 Enterprise PKCS #11 Coprocessor Crypto Express7 CCA Coprocessor Crypto Express7 Enterprise PKCS #11 Coprocessor Crypto Express8 CCA Coprocessor Crypto Express8 Enterprise PKCS #11 Coprocessor	

Part 3. PKCS #11 callable services

Chapter 17. Using PKCS #11 tokens and objects

This topic describes the callable services for creating and maintaining PKCS #11 tokens and objects. ICSF provides a number of callable services to assist you in managing PKCS #11 tokens and maintaining the token data set (TKDS). Services are also provided for generating, using, and managing key objects.

The following callable services are described:

- "PKCS #11 Derive Multiple Keys (CSFPDMK and CSFPDMK6)" on page 1339
- "PKCS #11 Derive Key (CSFPDVK and CSFPDVK6)" on page 1347
- "PKCS #11 Get Attribute Value (CSFPGAV and CSFPGAV6)" on page 1355
- "PKCS #11 Generate Key Pair (CSFPGKP and CSFPGKP6)" on page 1357
- "PKCS #11 Generate Secret Key (CSFPGSK and CSFPGSK6)" on page 1360
- "PKCS #11 Generate Keyed MAC (CSFPHMG and CSFPHMG6)" on page 1364
- "PKCS #11 Verify Keyed MAC (CSFPHMV and CSFPHMV6)" on page 1368
- "PKCS #11 One-Way Hash, Sign, or Verify (CSFPOWH and CSFPOWH6)" on page 1372
- "PKCS #11 Private Key Sign (CSFPPKS and CSFPPKS6)" on page 1379
- "PKCS #11 Public Key Verify (CSFPPKV and CSFPPKV6)" on page 1382
- "PKCS #11 Pseudo-Random Function (CSFPPRF and CSFPPRF6)" on page 1386
- "PKCS #11 Set Attribute Value (CSFPSAV and CSFPSAV6)" on page 1389
- "PKCS #11 Secret Key Decrypt (CSFPSKD and CSFPSKD6)" on page 1391
- "PKCS #11 Secret Key Encrypt (CSFPSKE and CSFPSKE6)" on page 1396
- "PKCS #11 Secret Key Reencrypt (CSFPSKR and CSFPSKR6)" on page 1403
- "PKCS #11 Token Record Create (CSFPTRC and CSFPTRC6)" on page 1407
- "PKCS #11 Token Record Delete (CSFPTRD and CSFPTRD6)" on page 1411
- "PKCS #11 Token Record List (CSFPTRL and CSFPTRL6)" on page 1413
- "PKCS #11 Unwrap Key (CSFPUWK and CSFPUWK6)" on page 1417
- "PKCS #11 Wrap Key (CSFPWPK and CSFPWPK6)" on page 1422

The metadata of objects in the TKDS can be managed using the following services:

- "Key Data Set List (CSFKDSL and CSFKDSL6)" on page 1225
- "Key Data Set Metadata Read (CSFKDMR and CSFKDMR6)" on page 1239
- "Key Data Set Metadata Write (CSFKDMW and CSFKDMW6)" on page 1246

A TKDS is not required to use the PKCS #11 services. If ICSF is started without a TKDS, however, only the omnipresent token will be available. The omnipresent token supports session objects only. Session objects are objects that do not persist beyond the life of a PKCS #11 session. The handle for the omnipresent token is 'SYSTOK-SESSION-ONLY' without the quotes and left-justified in a 44-byte field padded to the right with blanks.

PKCS #11 Derive Multiple Keys (CSFPDMK and CSFPDMK6)

Use the PKCS #11 Derive Multiple Keys callable service to generate multiple secret key objects and protocol dependent keying material from an existing secret key object. This service does not support any recovery methods.

The key handle must be a handle of a PKCS #11 secret key object. The CKA_DERIVE attribute for the secret key object must be true. The mechanism keyword specified in the rule array indicates what

derivation protocol to use. The derive parms list provides additional input/output data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPDMK6.

Format

```
CALL CSFPDMK(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             attribute_list_length,
             attribute_list,
             base_key_handle,
             parms_list_length,
             parms_list)
```

Parameters

return_code

Direction	Type
Output	String

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	String

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 548. Keywords for derive multiple keys

Keyword	Meaning
Mechanism (required)	
SSL-KM	Use the SSL 3.0 Key and MAC derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_KEY_AND_MAC_DERIVE.
TLS-KM	Use the TLS 1.0/1.1 Key and MAC derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_KEY_AND_MAC_DERIVE.
IKE1PHA1	Use the IKEv1 phase 1 protocol to derive multiple keys using a previously derived IKE seed key as the base key and a previously derived secret key as an additional key. 3 keys are derived (one derivation, one authentication, and one encryption key). Using IKE terminology, this mechanism performs {*SKEYID_d* | *SKEYID_a* | *SKEYID_e*} = *prf*(*SKEYID*, *g^xy* | *CKY-I* | *CKY-R*) with key expansion for *SKEYID_e*, if required. (*SKEYID_d,a* are always the size of the prf output.) Where: • *CKY-I* | *CKY-R* - is the concatenated initiator/responder cookie string • *SKEYID* - is the base key • *g^xy* - is the additional key • *SKEYID_d,a,e* - are the to-be-derived derivation, authentication and encryption keys
IKE2PHA1	Use the IKEv2 phase 1 (SA) protocol to derive multiple keys using a previously derived IKE seed key as the base key. 7 keys are derived (one derivation, two authentication, two encryption, and two peer authentication keys). Using IKE terminology, this mechanism performs {*SK_d* | *SK_ai* | *SK_ar* | *SK_ei* | *SK_er* | *SK_pi* | *SK_pr* } = *prf+*(*SKEYSEED*, *Ni* | *Nr* | *SPIi* | *SPIr*). Where: • *Ni* | *Nr* | *SPIi* | *SPIr* - is the concatenated initiator/responder nonce and Security Parameter Index string • *SKEYSEED* - is the base key • *SK_d*,*ai*,*ar*,*ei*,*er*,*pi*,*pr* - are the to-be-derived derivation, initiator authentication, responder authentication, initiator encryption, responder encryption, initiator peer authentication, and responder peer authentication keys

Table 548. Keywords for derive multiple keys (continued)

Keyword	Meaning
IKE1PHA2	Use the IKEv1 phase 2 (CHILD SA) protocol to derive multiple keys and salt values using a previously derived IKE derivation key as the base key and a previously derived secret key as an additional key (optional). The derivation produces one of the following key sets: • One authentication key • One GMAC key plus salt value • One authentication key plus one encryption key • One GCM key plus a salt value Up to two such sets are produced, one for the sender and one for the receiver. Using IKE terminology, this mechanism performs *KEYMAT* = *prf*(*SKEYID_d*, [*g^xy* |] *protocol* | *SPI* | *Ni_b* | *Nr_b*), done in two passes – once for the sender and once for the receiver. Where: • *protocol* | *SPI* | *Ni_b* | *Nr_b* - is the concatenated Protocol, Security Parameter Index, and initiator/responder nonce string • *SKEYID_d* - is the base key • *g^xy* - is the optional additional key • *KEYMAT* - is the generated key material which is partitioned into the key set
IKE2PHA2	Use the IKEv2 phase 2 protocol to derive multiple keys and salt values using a previously derived IKE derivation key as the base key and a previously derived secret key as an additional key (optional). The derivation produces one of the following key sets: • One authentication key • One GMAC key plus salt value • One authentication key plus one encryption key • One GCM key plus a salt value Two such sets are produced, one for the initiator and one for the responder. Using IKE terminology, this mechanism performs *KEYMAT* = *prf*+(*SK_d*, [*g^ir* |] *Ni* | *Nr*). Where: • *Ni* | *Nr* - is the concatenated initiator/responder nonce string • *SK_d* - is the base key • *g^ir* - is the optional additional key • *KEYMAT* - is the generated key material which is partitioned into the key set

attribute_list_length

Direction	Type
Input	Integer

The length of the attributes supplied in the *attribute_list* parameter in bytes. The minimum value for this field is 2 and the maximum value for this field is 32752.

attribute_list

Direction	Type
Input	String

List of attributes for the derived secret key object. See "Attribute list" on page 111 for the format of an *attribute_list*.

base_key_handle

Direction	Type
Input	String

The 44-byte handle of the base key object. See "Handles" on page 111 for the format of a *key_handle*.

parms_list_length

Direction	Type
Input	Integer

The length of the parameters supplied in the *parms_list* parameter in bytes.

parms_list

Direction	Type
Input	String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 549. parms_list parameter format for SSL-KM and TLS-KM mechanisms

Offset	Length in bytes	Direction	Description
0	1	Input	Boolean indicating if "export" processing is required. Any value other than x'00' means yes
1	3	Not applicable	reserved
4	4	Input	length in bytes of the client's random data (x)), where 1 <= length <= 32
8	4	Input	length in bytes of the server's random data (y)), where 1 <= length <= 32
12	4	Input	size of MAC to be generated in bits, where 8 <= size <= 384, in multiples of 8
16	4	Input	strength of key to be generated in bits. Zero if no encryption keys are to be generated. This may be less than the size of the key to be generated if the "export" boolean is set true.
20	4	Input	size of IV to be generated in bits (v), where 0<= size <= 128, in multiples of 8. Must be zero if no encryption keys are to be generated.
24	44	Output	handle of client MAC secret object created
68	44	Output	handle of server MAC secret object created
112	44	Output	handle of client key object created

Table 549. parms_list parameter format for SSL-KM and TLS-KM mechanisms (continued)			
Offset	**Length in bytes**	**Direction**	**Description**
156	44	Output	handle of server key object created
200	x	Input	client's random data
200+x	y	Input	server's random data
200+x+y	v/8	Output	client's IV
200+x+y+v/8	v/8	Output	server's IV

Table 550. parms_list parameter format for IKE1PHA1 mechanism			
Offset	**Length in bytes**	**Direction**	**Description**
0	1	Input	IKE version code. Must be x'01'
1	1	Input	PRF function code x'01' = HMAC_MD5, x'02' = HMAC_SHA1, x'04' = HMAC_SHA256, x'05' = SHA384, and x'06' = SHA512
2	4	Input	reserved
6	2	Input	length of to-be-derived encryption key, SKEYID_e
8	44	Input	Key handle of additional key
52	16	Input	Concatenated cookie string
68	44	Output	SKEYID_d key handle
112	44	Output	SKEYID_a key handle
156	44	Output	SKEYID_e key handle

Table 551. parms_list parameter format for IKE2PHA1 mechanism			
Offset	**Length in bytes**	**Direction**	**Description**
0	1	Input	IKE version code. Must be x'02'
1	1	Input	PRF function code x'01' = HMAC_MD5, x'02' = HMAC_SHA1, x'04' = HMAC_SHA256, x'05' = SHA384, and x'06' = SHA512
2	2	Input	length of to-be-derived derivation key, SK_d
4	2	Input	length of a single to-be-derived authentication key, SK_a
6	2	Input	length of a single to-be-derived encryption key, SK_e
8	2	Input	length of a single to-be-derived peer authentication key, SK_p
10	2	Input	Concatenated nonce, SPI string length (n), where 24 <= n <= 520
12	44	Output	SKEYID_d key handle
56	44	Output	Initiator SKEYID_a key handle
100	44	Output	Responder SKEYID_a key handle

Table 551. parms_list parameter format for IKE2PHA1 mechanism (continued)			
Offset	**Length in bytes**	**Direction**	**Description**
144	44	Output	Initiator SKEYID_e key handle
188	44	Output	Responder SKEYID_e key handle
232	44	Output	Initiator SKEYID_p key handle
276	44	Output	Responder SKEYID_p key handle
320	n	Input	Concatenated nonce, SPI string

Table 552. parms_list parameter format for IKE1PHA2 and IKE2PHA2 mechanisms			
Offset	**Length in bytes**	**Direction**	**Description**
0	1	Input	IKE version code. Must be x'01' for IKE1PHA2, x'02' for IKE2PHA2
1	1	Input	PRF function code x'01' = HMAC_MD5, x'02' = HMAC_SHA1, x'04' = HMAC_SHA256, x'05' = SHA384, and x'06' = SHA512
2	2	Input	length of to-be-derived salts (s), where 0 <= s <= 4. Zero if salts are not to be derived
4	2	Input	length of to-be-derived authentication keys. Zero if authentication keys are not to be derived
6	2	Input	length of to-be-derived encryption, GMAC, or GCM keys. Zero if no such keys are to be derived
8	2	Input	First pass parameter string length (n) • For IKE1PHA2 – Receiver concatenated Protocol, Security Parameter Index, and initiator/responder nonce string length, where 25 <= n <= 525 • For IKE2PHA2 – Concatenated initiator/responder nonce string length, where 16 <= n <= 512.
10	2	Input	Second pass parameter string length (m) • For IKE1PHA2 – Sender concatenated Protocol, Security Parameter Index, and initiator/responder nonce string length, where 25 <= m <= 525. Zero if second pass is to be skipped • For IKE2PHA2 – Not used. Must be zero
12	44	Input	Key handle of additional key. Fill with binary zeros if n/a
56	44	Output	Initiator (sender) authentication key handle
100	44	Output	Responder (receiver) authentication key handle
144	44	Output	Initiator (sender) encryption, GMAC, or GCM key handle
188	44	Output	Responder (receiver) encryption, GMAC, or GCM key handle

Table 552. parms_list parameter format for IKE1PHA2 and IKE2PHA2 mechanisms (continued)			
Offset	**Length in bytes**	**Direction**	**Description**
232	n	Input	First pass parameter string
232+n	m	Input	Second pass parameter string
232+n+m	s	Output	Initiator (sender) salt
232+n+m+s	s	Output	Responder (receiver) salt

Authorization

There are multiple keys involved in this service — one or two base keys and the target keys (the new keys created from the base key).

- To use a base key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a base key that is a private object, the caller must have USER (READ) authority (user access).
- To derive a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.
- To derive a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

Usage Notes

The service does not support secure keys.

Key derivation functions are performed in software.

For the SSL-KM and TLS-KM mechanisms, an attribute list is required if encryption keys are to be generated.

For the IKE1PHA1, IKE2PHA1, IKE1PHA2, and IKE2PHA2 mechanisms, the following attribute rules apply to the derived keys:

- Derivation keys will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA_CLASS=CKO_SECRET_KEY
  - CKA_KEY_TYPE=CKK_GENERIC_SECRET
  - CKA_DERIVE=TRUE
  - CKA_VALUE_LEN=*as specified in the parms list*
- Authentication keys will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA_CLASS=CKO_SECRET_KEY
  - CKA_KEY_TYPE=CKK_GENERIC_SECRET
  - CKA_SIGN=TRUE=TRUE
  - CKA_VERIFY=TRUE=TRUE
  - CKA_VALUE_LEN= *as specified in the parms list*
- Encryption, GMAC, and GCM keys will be typed according to information found in the attribute list. However, they will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA_CLASS=CKO_SECRET_KEY
  - For key types other than CKK_DES, CKK_DES2, and CKK_DES3, CKA_VALUE_LEN= *as specified in the parms list*

- All key types will inherit the values of the CKA_SENSITIVE, CKA_ALWAYS_SENSITIVE, CKA_EXTRACTABLE, and CKA_NEVER_EXTRACTABLE attributes from the base key. These may not be overridden by other values in the attribute list. If an additional key is specified, its values will be applied after setting the base key values as follows:
  - If the additional key has CKA_SENSITIVE=TRUE, so will the derived key or keys.
  - If the additional key has CKA_EXTRACTABLE=FALSE, so will the derived key or keys.
  - If the additional key has CKA_ALWAYS_SENSITIVE=FALSE, so will the derived key or keys.
  - If the additional key has CKA_NEVER_EXTRACTABLE=FALSE, so will the derived key or keys.
- If encryption, GMAC, or GCM keys are to be derived, an attribute list is required for the key typing information. Otherwise, it is optional. For all keys, other applicable secret key attributes may be specified in the attribute list. Any attribute not specified will be assigned the default value normally assigned to a newly created secret key.

For the IKE1PHA1, IKE1PHA2, and IKE2PHA2 mechanisms, the additional key must be a secret key (CKA_CLASS=CKO_SECRET_KEY) capable of performing key derivation (CKA_DERIVE=TRUE). It must also be contained in the same PKCS #11 token as the base key.

The IKE1PHA1, IKE2PHA1, IKE1PHA2, and IKE2PHA2 mechanisms have the following limitations if the operation is FIPS 140 restricted:

- The MD5 PRF may not be specified.
- The length of the base key must be at least half the length of the output of the PRF function.

PKCS #11 Derive Key (CSFPDVK and CSFPDVK6)

Use the PKCS #11 Derive Key callable service to generate a new secret key object from an existing key object. This service does not support any recovery methods.

The deriving key handle must be a handle of an existing PKCS #11 key object. The CKA_DERIVE attribute for this object must be true. The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPDVK6.

Format

```
CALL CSFPDVK(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             attribute_list_length,
             attribute_list,
             base_key_handle,
             parms_list_length,
             parms_list,
             target_key_handle)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1 or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 553. Keywords for derive key

Keyword	Meaning
Mechanism (One required)	
PKCS-DH	Use the Diffie-Hellman PKCS derivation protocol as defined in the PKCS #11 standard as mechanism CKM_DH_PKCS_DERIVE.
SSL-MS	Use the SSL 3.0 Master Secret derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_MASTER_KEY_DERIVE. The SSL protocol version is also returned. The base key must have been generated according to the rules for SSL 3.0.
SSL-MSDH	Use the SSL 3.0 Master Secret for Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_MASTER_KEY_DERIVE_DH.
TLS-MS	Use the TLS Master Secret derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_MASTER_KEY_DERIVE. The base key must have been generated according to the rules for TLS 1.0 or TLS 1.1.

Table 553. Keywords for derive key (continued)	
Keyword	**Meaning**
TLS-MSDH	Use the TLS Master Secret for Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_MASTER_KEY_DERIVE_DH.
EC-DH	Use the Elliptic Curve Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_ECDH1_DERIVE.
IKESEED	Use the IKEv1 or IKEv2 initial seeding protocol to derive a seed key using a previously derived secret key as the base key. Using IKE terminology, this mechanism performs either *SKEYID* = *prf*(*Ni_b* | *Nr_b*, *g^xy*) for IKEv1 or *SKEYSEED* = *prf*(*Ni* | *Nr*, *g^ir*) for IKEv2. Where: • *Ni_b* | *Nr_b* or *Ni* | *Nr* - is the concatenated initiator/responder nonce string. • *g^xy* or *g^ir* - is the base key.
IKESHARE	Use the IKEv1 initial seeding protocol to derive a seed key using a pre-shared secret key as the base key. Using IKE terminology, this mechanism performs *SKEYID* = *prf*(*pre-shared-key*, *Ni_b* | *Nr_b*). Where: • *Ni_b* | *Nr_b* - is the concatenated initiator/responder nonce string. • *pre-shared-key* - is the base key.
IKEREKEY	Use the IKEv2 rekeying protocol to derive a new seed key using a previously derived IKE derivation key as the base key and a previously derived secret key as an additional key. Using IKE terminology, this mechanism performs *SKEYSEED* = *prf*(*SK_d*, *g^ir* | *Ni* | *Nr*). Where: • *Ni* | *Nr* - is the concatenated initiator/responder nonce string. • *SK_d* - is the base key. • *g^ir* - is the additional key.
KYBER	Perform the second step of a hybrid mode key exchange by using a generic secret key object generated with elliptic-curve Diffie-Hellman.
Key destination (Optional)	
OMNITOKN	Store the derived session key in the Omnipresent Token (label SYSTOK-SESSION-ONLY). The default action is to store the key in the same token as the base key or keys.

attribute_list_length

Direction	**Type**
Input	Integer

The length of the attributes supplied in the *attribute_list* parameter in bytes. The minimum value for this field is 2 and the maximum value for this field is 32752.

attribute_list

Direction	Type
Input	String

List of attributes for the derived secret key object. See “Attribute list” on page 111 for the format of an *attribute_list*.

base_key_handle

Direction	Type
Input	String

The 44-byte handle of the source key object. See “Handles” on page 111 for the format of a *base_key_handle*.

parms_list_length

Direction	Type
Input	Integer

The length of the parameters supplied in the *parms_list* parameter in bytes.

parms_list

Direction	Type
Input/Output	String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 554. parms_list parameter format for PKCS-DH mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Length in bytes of the other party's public value, where 1 <= length <= 256.
4	<=256	Input	Binary value representing the other party's public value.

Table 555. parms_list parameter format for SSL-MS, SSL-MSDH, TLS-MS, and TLS-MSDH mechanisms

Offset	Length in bytes	Direction	Description
0	2	Output	SSL protocol version returned for SSL-MS and TLS-MS only. For the other protocols, this field is left unchanged.
2	2	N/A	Reserved.
4	4	Input	Length in bytes of the client's random data (x), where 1 <= length <= 32.
8	4	Input	Length in bytes of the server's random data (y)), where 1 <= length <= 32.
12	x	Input	Client's random data.
12+x	y	Input	Server's random data.

Table 556. parms_list parameter format for EC-DH mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	KDF function code: **X'00000001'** CKD_NULL **X'00000002'** CKD_SHA1_KDF **X'00000005'** CKD_SHA224_KDF **X'00000006'** CKD_SHA256_KDF **X'00000007'** CKD_SHA384_KDF **X'00000008'** CKD_SHA512_KDF **X'80000001'** CKD_IBM_HYBRID_NULL **X'80000002'** CKD_IBM_HYBRID_SHA1_KDF **X'80000003'** CKD_IBM_HYBRID_SHA224_KDF **X'80000004'** CKD_IBM_HYBRID_SHA256_KDF **X'80000005'** CKD_IBM_HYBRID_SHA384_KDF **X'80000006'** CKD_IBM_HYBRID_SHA512_KDF **Note:** Previously, the CKD_*xxx* values were compressed to a single byte (the high-order byte of this field) to represent the KDF function code. This is tolerated for existing callers for function codes X'01', X'02', X'05', X'06', X'07', and X'08', but new callers should use the actual CKD_*xxx* values described above.
4	4	Input	Length in bytes of the optional data shared between the two parties. A zero length means no shared data. For the NULL KDF the length must be zero. Otherwise, • When deriving a clear key, the maximum shared data length is 2147483647. • When deriving a secure key, the maximum shared data length is 1024.
8	8	Input	64-bit address of the data shared between the two parties. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length is zero.

Table 556. parms_list parameter format for EC-DH mechanism (continued)

Offset	Length in bytes	Direction	Description
16	4	Input	Length in bytes of the other party's public value (x). This length is dependent on the curve type/size of the base key and on whether the value is DER encoded or not: secp192r1 - 49 (51 w/DER) secp224r1 - 57 (59 w/DER) secp256r1 - 65 (67 w/DER) secp384r1 - 97 (99 w/DER) secp521r1 - 133 (136 w/DER) brainpoolP160r1 - 41 (43 w/DER) brainpoolP192r1 - 49 (51 w/DER) brainpoolP224r1 - 57 (59 w/DER) brainpoolP256r1 - 65 (67 w/DER) brainpoolP320r1 - 81 (83 w/DER) brainpoolP384r1 - 97 (99 w/DER) brainpoolP512r1 - 129 (132 w/DER) X25519 - 32 (34 w/DER) X448 - 57 (59 w/DER).
20	x<=136	Input	Binary value representing the other party's public value with or without DER encoding.

Table 557. parms_list parameter format for IKESEED, IKESHARE, and IKEREKEY mechanisms

Offset	Length in bytes	Direction	Description
0	1	Input	IKE version code. Must be x'01' for IKESHARE, x'02' for IKEREKEY, x'01' or x'02' for IKESEED.
1	1	Input	PRF function code x'01' = HMAC_MD5, x'02' = HMAC_SHA1, x'04' = HMAC_SHA256, x'05' = SHA384, and x'06' = SHA512.
2	2	Input	Length of concatenated initiator/responder nonce string (n), where 16 <= n <= 512.
4	44	Input	Key handle of additional key - required for IKEREKEY. Ignored for the other mechanisms.
48	n	Input	Concatenated initiator/responder nonce string.

Table 558. parms_list parameter format for Kyber mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Version of this structure (must be X'00000000').
4	4	Input	Encapsulation/decapsulation mode: **X'00000001'** CK_IBM_KEM_ENCAPSULATE **X'00000002'** CK_IBM_KEM_DECAPSULATE

Table 558. parms_list parameter format for Kyber mechanism (continued)

Offset	Length in bytes	Direction	Description
8	4	Input	Key derivation function: **X'80000002'** CKD_IBM_HYBRID_SHA1_KDF **X'80000003'** CKD_IBM_HYBRID_SHA224_KDF **X'80000004'** CKD_IBM_HYBRID_SHA256_KDF **X'80000005'** CKD_IBM_HYBRID_SHA384_KDF **X'80000006'** CKD_IBM_HYBRID_SHA512_KDF
12	1	Input	Whether to prepend new key material to BLOB: **X'00'** CK_FALSE **X'01'** CK_TRUE
13	3	N/A	Reserved.
16	4	Input	Length in bytes (*sss*) of the optional data shared between the two parties. A zero length means no shared data. When deriving a secure key, the maximum shared data length is 1024.
20	4	Input/ Output	Length in bytes (*ccc*) of the cipher (Kyber-encapsulated random key material): • For CK_IBM_KEM_ENCAPSULATE, this is the size of the buffer allotted for the output cipher. This length will be updated on successful completion of the operation. It is recommended to provide 1600 bytes for output. • For CK_IBM_KEM_DECAPSULATE, this is the size of the input cipher provided by the other party. This field will not be updated.
24	44	Input	Handle of the generic secret key derived previously using the other party's ECC public-key and our ECC private key, using CKD_IBM_HYBRID_NULL.
68	*sss*	Input	Optional data shared between the two parties.
68+*sss*	*ccc*	Input/ Output	Cipher (Kyber-encapsulated random key material): • For CK_IBM_KEM_ENCAPSULATE, this buffer will be updated with the ciphertext on successful completion of the operation. • For CK_IBM_KEM_DECAPSULATE, this is the cipher provided by the other party. This field will not be updated.

target_key_handle

Direction	Type
Output	String

Upon successful completion, the 44-byte handle of the secret key object that was derived.

Authorization

There are multiple keys involved in this service — one or two base keys and the target key (the new key created from the base key).

- To use a base key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a base key that is a private object, the caller must have USER (READ) authority (user access).
- To derive a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.
- To derive a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

Usage Notes

Rule EC-DH may derive a clear or secure key. Rule KYBER may only derive a secure key. For all other rules, the service does not support the derivation of secure keys. For rules EC-DH, PKCS-DH, and KYBER only, the input base key may be a secure key.

Derivation of the EC-DH shared secret "Z" may be performed in hardware or software.

- Derivation using a secure key requires an EP11 coprocessor.
- On IBM z15 and IBM z15 T02 (or later hardware), CP Assist for Cryptographic Functions will be used for derivation using the following elliptic curves:
  - secp256r1.
  - secp384r1.
  - secp521r1.
  - Curve25519 (Montgomery form).
  - Curve448 (Montgomery form).
- Derivation of the PKCS-DH shared secret will be performed using the EP11 coprocessor for a secure base key or accelerator (if available) for a clear base key.
- All other key derivation operations are performed in software.

For the IKESEED, IKESHARE, and IKEREKEY mechanisms, the following attribute rules apply to the derived key:

- The key will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA_CLASS=CKO_SECRET_KEY
  - CKA_KEY_TYPE=CKK_GENERIC_SECRET
  - CKA_DERIVE=TRUE
  - CKA_VALUE_LEN=*length of the output of the PRF function*
- Other applicable secret key attributes may be specified in the attribute list. However, an attribute list is not required. Any attribute not specified will be assigned the default value normally assigned to a newly created secret key. In particular, CKA_SENSITIVE defaults to FALSE and CKA_EXTRACTABLE defaults to TRUE.
- CKA_ALWAYS_SENSITIVE is set to FALSE if the CKA_ALWAYS_SENSITIVE attribute from the base key is FALSE. Otherwise it is set equal to the value of the CKA_SENSITIVE attribute assigned to the derived key.
- CKA_NEVER_EXTRACTABLE is set to FALSE if the CKA_NEVER_EXTRACTABLE attribute from the base key is FALSE. Otherwise it is set opposite to the value of the CKA_EXTRACTABLE attribute assigned to the derived key.

For the IKEREKEY mechanism, the additional key must be a secret key (CKA_CLASS=CKO_SECRET_KEY) capable of performing key derivation (CKA_DERIVE=TRUE). It must also be contained in the same PKCS #11 token as the base key.

For the IKESEED, IKESHARE, and IKEREKEY mechanisms, the MD5 PRF may not be specified if the operation is FIPS 140 restricted.

For the IKESHARE and IKEREKEY mechanisms, the length of the base key must be at least half the length of the output of the PRF function if the operation is FIPS 140 restricted.

For the IKESEED mechanism, the length of the concatenated initiator/responder nonce value must be at least half the length of the output of the PRF function if the operation is FIPS 140 restricted.

The OMNITOKN rule cannot be specified in combination with CKA_TOKEN=TRUE in the *attribute_list*.

CCA access controls

If your system has a Crypto Express adapter configured as a CCA coprocessor, the service will use the coprocessor to process your request. In order for the request to complete successfully, the access controls for the ECC Diffie-Hellman (CSNDEDH and CSNFEDH) callable service must be enabled. See the access control points section for "ECC Diffie-Hellman (CSNDEDH and CSNFEDH)" on page 182 for more information.

PKCS #11 Get Attribute Value (CSFPGAV and CSFPGAV6)

Use the PKCS #11 Get Attribute Value callable service (CSFPGAV) to retrieve the attributes of an object.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPGAV6.

Format

```
CALL CSFPGAV(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               handle,
               rule_array_count,
               rule_array,
               attribute_list_length,
               attribute_list)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

handle

Direction	Type
Input	String

The 44-byte handle of the object. See “Handles” on page 111 for the format of a *handle*.

rule_array_count

Direction	Type
Input	Integer

The number of keywords supplied in the *rule_array* parameter. This value must be 0.

rule_array

Direction	Type
Ignored	String

This field is ignored.

attribute_list_length

Direction	Type
Input/Output	Integer

On input, the length of the *attribute_list* parameter in bytes.

On output, the length of the *attribute_list* parameter in bytes. If the length supplied on input is insufficient to hold all attributes, the length on output is set to the minimum length required.

The minimum value for this field is 2 and the maximum value for this field is 32752.

attribute_list

Direction	Type
Output	String

A list of object attributes.

See “Attribute list” on page 111 for the format of an *attribute_list*.

Authorization

The token authorization required and the amount of attribute information returned is dependent on the values of the attributes the object possesses.

The authority to retrieve the non-sensitive attributes is as follows:

- For a public object - any authority to the token (USER (READ) or SO (READ))
- For a private object - USER (READ) or SO (CONTROL)

If the caller is not authorized to retrieve the non-sensitive attributes, the service fails.

If the caller is authorized to retrieve the non-sensitive attributes and the object does not possess any sensitive attributes, the service returns all the object's attributes.

If the caller is authorized to retrieve the non-sensitive attributes and the object does possess sensitive attributes, processing is as defined in this table:

Table 559. Get attribute value processing for objects possessing sensitive attributes

Object	PKCS #11 role authority	CKA_SENSITIVE	CKA_EXTRACTABLE	Attributes returned
Public	USER (READ) or SO (READ)	True	True or False	Non-sensitive only
Private	USER (READ) or SO (CONTROL)	True	True or False	Non-sensitive only
Public	USER (READ) or SO (READ)	False	False	Non-sensitive only
Private	USER (READ) or SO (CONTROL)	False	False	Non-sensitive only
Public	USER (READ) or SO (READ)	False	True	Sensitive and non-sensitive
Private	SO (CONTROL)	False	True	Non-sensitive only
Private	USER (READ)	False	True	Sensitive and non-sensitive

Note:

- Session and token objects require the same authority.
- The sensitive attributes are as follows:
  - CKA_VALUE for a secret key, Elliptic Curve private key, DSA private key, or Diffie-Hellman private key object.
  - CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, and CKA_COEFFICIENT for a private key object.
- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles.

Usage Notes

If the object is marked sensitive or not extractable, the sensitive attributes are not returned.

If the caller is authorized to list the non-sensitive attributes of an object, but not the sensitive ones, the sensitive attributes are not returned.

If the caller is not authorized to list the non-sensitive attributes of the object, the service fails.

PKCS #11 Generate Key Pair (CSFPGKP and CSFPGKP6)

Use the PKCS #11 Generate Key Pair callable service to generate an RSA, DSA, Elliptic Curve, Diffie-Hellman, Dilithium (LI2), or Kyber key pair. New token or session objects are created to hold the key pair.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPGKP6.

Format

```
CALL CSFPGKP(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              token_handle,
              rule_array_count,
              rule_array,
              public_key_attribute_list_length,
              public_key_attribute_list,
              public_key_object_handle,
              private_key_attribute_list_length,
              private_key_attribute_list,
              private_key_object_handle)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

token_handle

Direction	Type
Input	String

The 44-byte handle of the token of the key objects. See “Handles” on page 111 for the format of a *token_handle*.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array_parameter*. This value must be 0.

rule_array

Direction	Type
Ignored	String

This field is ignored.

public_key_attribute_list_length

Direction	Type
Input	Integer

The length of the attributes supplied in the *public_key_attribute* list parameter in bytes.

public_key_attribute_list

Direction	Type
Input	String

List of attributes for the public key object. The minimum value for this field is 2 and the maximum value for this field is 32752. See "Attribute list" on page 111 for the format of a *public_key_attribute_list*.

public_key_object_handle

Direction	Type
Output	String

The 44-byte handle of the new public key object.

private_key_attribute_list_length

Direction	Type
Input	Integer

The length of the attributes supplied in the *private_key_attribute_list* parameter in bytes.

private_key_attribute_list

Direction	Type
Input	Integer

List of attributes for the private key object. The minimum value for this field is 2 and the maximum value for this field is 32752. See "Attribute list" on page 111 for the format of a *private_key_attribute_list*.

private_key_object_handle

Direction	Type
Output	String

The 44-byte handle of the new private key object.

Authorization

To generate a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.

To generate a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

Usage Notes

The type of key pair generated is determined by the key type attributes in the *public_key_attributes_list* and *private_key_attributes_list* parameters.

Key pair generation may be done in hardware or software.

- Secure key pair generation requires an EP11 coprocessor.
- Generating an RSA key pair when FIPS compliance is not required will use a CCA coprocessor, if available.
- On IBM z15 and IBM z15 T02 (or later hardware), CP Assist for Cryptographic Functions will be used for clear key pair generation for the following elliptic curves:
  - secp256r1.
  - secp384r1.
  - secp521r1.
  - Curve25519 (both the Montgomery and the twisted Edwards forms).
  - Curve448 (both the Montgomery and the twisted Edwards forms).
- In all other cases, software will be used.

Consider the compliance mode of your EP11 coprocessor when specifying attributes. For information on the key/modulus size to use for each ACP when in the different compliance modes, see 'PKCS #11 Access Control Points' in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

PKCS #11 Generate Secret Key (CSFPGSK and CSFPGSK6)

Use the generate secret key callable service to generate a secret key or set of domain parameters. A new token or session object is created to hold the information.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPGSK6.

Format

```
CALL CSFPGSK(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             handle,
             rule_array_count,
             rule_array,
             attribute_list_length,
             attribute_list,
             parms_list_length,
             parms_list )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

handle

Direction	Type
Input/Output	String

On input, the 44-byte handle of the token. On output, the 44-byte handle of the new secret key or domain parameters object. See "Handles" on page 111 for the format of a *handle*.

rule_array_count

The number of keywords you supplied in the *rule_array* parameter. This value must be 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service.

Table 560. Keywords for generate secret key

Keyword	Meaning
Mechanism (One of the following must be specified)	
KEY	Generate a secret key object according to the key type attribute in the *attribute_list* parameter (for CKM_AES_KEY_GEN, CKM_BLOWFISH_KEY_GEN, CKM_IBM_CHACHA20_KEY_GEN, CKM_DES_KEY_GEN, CKM_DES2_KEY_GEN, CKM_DES3_KEY_GEN, CKM_GENERIC_SECRET_KEY_GEN, and CKM_RC4_KEY_GEN)
PARMS	Generate a domain parameters object according to the key type attribute in the *attribute_list* parameter (for CKM_DSA_PARAMETER_GEN and CKM_DH_PKCS_PARAMETER_GEN)

Table 560. Keywords for generate secret key (continued)	
Keyword	**Meaning**
PBEKEY	Generate password-based encryption key material and a secret key object according to the key type attribute in the *attribute_list* parameter (for CKM_PBE_SHA1_DES3_EDE_CBC only)
PBKDF2	Generate password-based encryption key material and a secret key object according to the key type attribute in the *attribute_list* parameter (for CKM_PKCS5_PBKD2 only). Only clear objects are supported.
SSL	Generate a generic secret key object where the client is using SSL (for CKM_SSL3_PRE_MASTER_KEY_GEN)
TLS	Generate a generic secret key object where the client is using TLS (for CKM_TLS_PRE_MASTER_KEY_GEN)

attribute_list_length

Direction	Type
Input	Integer

The length of the attributes supplied in the *attribute_list* parameter in bytes. The minimum value for this field is 2 and the maximum value for this field is 32752.

attribute_list

Direction	Type
Input	String

List of attributes for the secret key object. See “Attribute list” on page 111 for the format of an *attribute_list*.

parms_list_length

Direction	Type
Input	Integer

The length of the parameters supplied in the *parms_list* parameter in bytes.

parms_list

Direction	Type
Input/Output	String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 561. parms_list parameter format for SSL and TLS mechanism			
Offset	**Length in bytes**	**Direction**	**Description**
0	2	Input	SSL or TLS version number. For example, for version 3.01, this would be X'0301'.

Table 562. parms_list parameter format for PBEKEY mechanism

Offset	Length in bytes	Direction	Description
0	2	Input	Length in bytes of the password (p), where 1 <= p <= 128.
2	2	Input	Length in bytes of the salt (s), where 1 <= s <= 128.
4	4	Input	Number of iterations required (n), where 1 <= n <= 65,536.
8	8	Output	8-byte IV returned.
16	p	Input	Password.
16+p	s	Input	Salt.

Table 563. parms_list parameter format for PBKDF2 mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Source of the salt value. Currently, this must be X'00000001' (CKZ_SALT_SPECIFIED).
4	4	Input	Number of iterations to perform when generating each block of random data (n), where 1 <= n <= 65,536.
8	4	Input	Pseudo-random function to use to generate the key (PRF). Supported values are: **X'00000001'** CKP_PKCS5_PBKD2_HMAC_SHA1 **X'00000003'** CKP_PKCS5_PBKD2_HMAC_SHA224 **X'00000004'** CKP_PKCS5_PBKD2_HMAC_SHA256 **X'00000005'** CKP_PKCS5_PBKD2_HMAC_SHA384 **X'00000006'** CKP_PKCS5_PBKD2_HMAC_SHA512
12	2	Input	Length of the salt source input in bytes (s), where 1 <= s <= 128.
14	2	Input	Length of the input data for the PRF in bytes (d); Currently, d must be 0.
16	2	Input	Length of the password in bytes (p), where 1 <= p <= 128.
18	s	Input	Salt data (ASCII).
18+s	d	Input	PRF data (empty since d must be 0).
18+s+d	p	Input	Password (ASCII).

For the KEY and PARMS mechanisms, there are no parameters. The *parms_list_length* parameter must be set to zero for these mechanisms.

Authorization

To generate a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.

To generate a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

Usage Notes

Domain parameters are generated in hardware when an Enterprise PKCS #11 coprocessor is present. Otherwise, they are generated in software.

BLOWFISH, RC4, TLS, and SSL key generation is performed in software. All other key generation may be performed in hardware or software.

PKCS #11 Generate Keyed MAC (CSFPHMG and CSFPHMG6)

Use the PKCS #11 Generate Keyed MAC callable service to generate either an encryption-based message authentication code (MAC) or hashed-based MAC (HMAC). This service does not support any recovery methods. The key handle must be a handle of a PKCS #11 secret key object.

For HMAC, the secret key object must be a generic secret key. Otherwise, the secret key object must be typed according to the encryption algorithm. The mechanism keyword specified in the rule array indicates the algorithm to use. The CKA_SIGN attribute for the secret key object must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPHMG6.

Format

```
CALL CSFPHMG(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              text_length,
              text,
              text_id,
              chain_data_length,
              chain_data,
              key_handle,
              hmac_length,
              hmac )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1, 2, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justisfied in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 564. Keywords for Generate Keyed MAC

Keyword	Meaning
Mechanism (Required)	
MD5	Generate an HMAC. Use MD5 hashing. Output returned in the *hmac* parameter is 16 bytes in length.
SHA-1	Generate an HMAC. Use SHA-1 hashing. Output returned in the *hmac* parameter is 20 bytes in length.
SHA-224	Generate an HMAC. Use SHA-224 hashing. Output returned in the *hmac* parameter is 28 bytes in length.
SHA-256	Generate an HMAC. Use SHA-256 hashing. Output returned in the *hmac* parameter is 32 bytes in length.
SHA-384	Generate an HMAC. Use SHA-384 hashing. Output returned in the *hmac* parameter is 48 bytes in length.
SHA-512	Generate an HMAC. Use SHA-512 hashing. Output returned in the *hmac* parameter is 64 bytes in length.
SHA3-224	Generate an HMAC. Use SHA3-224 hashing. Output returned in the *hmac* parameter is 28 bytes in length.
SHA3-256	Generate an HMAC. Use SHA3-256 hashing. Output returned in the *hmac* parameter is 32 bytes in length.
SHA3-384	Generate an HMAC. Use SHA3-384 hashing. Output returned in the *hmac* parameter is 48 bytes in length.
SHA3-512	Generate an HMAC. Use SHA3-512 hashing. Output returned in the *hmac* parameter is 64 bytes in length.
SSL3-MD5	Generate a MAC according to the SSL v3 protocol. Use MD5 hashing. Output returned in the *hmac* parameter is 16 bytes in length.

Table 564. Keywords for Generate Keyed MAC (continued)	
Keyword	**Meaning**
SSL3-SHA	Generate a MAC according to the SSL v3 protocol. Use SHA1 hashing. Output returned in the *hmac* parameter is 20 bytes in length.
Length selection (Optional)	
FIXED	The length of the returned value is predetermined according to the mechanism rule. This is the default length selection.
GENERAL	The length of the returned value is caller selectable. This rule is not supported for the HMAC mechanisms.
Chaining selection (Optional)	
FIRST	Specifies this is the first call in a series of chained calls.
MIDDLE	Specifies this is a middle call in a series of chained calls.
LAST	Specifies this is the last call in a series of chained calls.
ONLY	Specifies this is the only call and the call is not chained. This is the default.

text_length

Direction	Type
Input	Integer

Length of the *text* parameter in bytes. The length can be from 0 to 2147483647.

text

Direction	Type
Input	String

Value for which an HMAC will be generated.

text_id

Direction	Type
Input	Integer

The ALET identifying the space where the text resides.

chain_data_length

Direction	Type
Input/Output	Integer

The byte length of the *chain_data* parameter. This must be 128 bytes.

chain_data

Direction	Type
Input/Output	String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE and LAST calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

Table 565. chain_data parameter format

Offset	Length	Description
0	4	Flag word **Bit** **Meaning when set on** **0** Cryptographic state object has been allocated **1-31** Reserved for IBM's use
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

key_handle

Direction	Type
Input	String

The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests. Must be the handle of an Enterprise PKCS #11 secure object when the mechanism is a form of SHA3 hashing. See "Handles" on page 111 for the format of a *key_handle*.

hmac_length

Direction	Type
Input	Integer

The length of the *hmac* parameter in bytes. When the FIXED length selection is specified or defaulted, this field is ignored. When the GENERAL length selection is specified, this parameter specifies the exact length of the MAC desired, ranging from 4 to 8 bytes.

hmac

Direction	Type
Output	String

Upon successful completion of an ONLY or LAST request, this field contains the generated MAC or HMAC value, left justified. The caller must provide an area large enough to hold the generated MAC or HMAC as defined by the mechanism specified and the length selection rule. This field is ignored for FIRST and MIDDLE requests.

Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

Usage Notes

HMAC operations are performed in hardware or software.

If the FIRST rule is used to start a series of chained calls:

- The key used to initiate the chained calls must not be deleted until the chained calls are complete.

- The application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the *chain_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

The use of a SHA3 hash method requires an active Enterprise PKCS #11 coprocessor.

PKCS #11 Verify Keyed MAC (CSFPHMV and CSFPHMV6)

Use the PKCS #11 Verify Keyed MAC callable service to verify either an encryption-based message authentication code (MAC) or hashed-based MAC (HMAC). This service does not support any recovery methods. The key handle must be a handle of a PKCS #11 secret key object.

For HMAC, the secret key object must be a generic secret key. Otherwise, the secret key object must be typed according to the encryption algorithm. The mechanism keyword specified in the rule array indicates the algorithm to use. The CKA_VERIFY attribute for the secret key object must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPHMV6.

Format

```
CALL CSFPHMV(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              text_length,
              text,
              text_id,
              chain_data_length,
              chain_data,
              key_handle,
              hmac_length,
              hmac )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1 or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 566. Keywords for Verify Keyed MAC

Keyword	Meaning
Mechanism (Required)	
MD5	Verify an HMAC. Use MD5 hashing. Data supplied in the *hmac* parameter must be 16 bytes in length.
SHA-1	Verify an HMAC. Use SHA-1 hashing. Data supplied in the *hmac* parameter must be 20 bytes in length.
SHA-224	Verify an HMAC. Use SHA-224 hashing. Data supplied in the *hmac* parameter must be 28 bytes in length.
SHA-256	Verify an HMAC. Use SHA-256 hashing. Data supplied in the *hmac* parameter must be 32 bytes in length.
SHA-384	Verify an HMAC. Use SHA-384 hashing. Data supplied in the *hmac* parameter must be 48 bytes in length.
SHA-512	Verify an HMAC. Use SHA-512 hashing. Data supplied in the *hmac* parameter must be 64 bytes in length.
SHA3-224	Verify an HMAC. Use SHA3-224 hashing. Data supplied in the *hmac* parameter must be 28 bytes in length.
SHA3-256	Verify an HMAC. Use SHA3-256 hashing. Data supplied in the *hmac* parameter must be 32 bytes in length.
SHA3-384	Verify an HMAC. Use SHA3-384 hashing. Data supplied in the *hmac* parameter must be 48 bytes in length.
SHA3-512	Verify an HMAC. Use SHA3-512 hashing. Data supplied in the *hmac* parameter must be 64 bytes in length.
SSL3-MD5	Verify a MAC according to the SSL v3 protocol. Use MD5 hashing. Data supplied in the *hmac* parameter must be 16 bytes in length.
SSL3-SHA	Verify a MAC according to the SSL v3 protocol. Use SHA1 hashing. Data supplied in the *hmac* parameter must be 20 bytes in length.

Table 566. Keywords for Verify Keyed MAC (continued)	
Keyword	**Meaning**
Length selection (Optional)	
FIXED	The length of the input MAC or HMAC is predetermined according to the mechanism rule. This is the default length selection.
GENERAL	The length of the input MAC is caller selectable. This rule is not supported for the HMAC mechanisms.
Chaining selection (Optional)	
FIRST	Specifies this is the first call in a series of chained calls.
MIDDLE	Specifies this is a middle call in a series of chained calls.
LAST	Specifies this is the last call in a series of chained calls.
ONLY	Specifies this is the only call and the call is not chained. This is the default.

text_length

Direction	Type
Input	Integer

Length of the *text* parameter in bytes. The length can be from 0 to 2147483647.

text

Direction	Type
Input	String

Value for which an HMAC will be generated.

text_id

Direction	Type
Input	Integer

The ALET identifying the space where the text resides.

chain_data_length

Direction	Type
Input/Output	Integer

The byte length of the *chain_data* parameter. This must be 128 bytes.

chain_data

Direction	Type
Input/Output	String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE and LAST calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

Table 567. chain_data parameter format

Offset	Length	Description
0	4	Flag word **Bit** **Meaning when set on** **0** Cryptographic state object has been allocated **1-31** Reserved for IBM's use
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

key_handle

Direction	Type
Input	String

The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests. Must be the handle of an Enterprise PKCS #11 secure object when the mechanism is a form of SHA3 hashing. See "Handles" on page 111 for the format of a *key_handle*.

hmac_length

Direction	Type
Input	Integer

The length of the *hmac* parameter in bytes. When the FIXED length selection is specified or defaulted, this field is ignored. When the GENERAL length selection is specified, this parameter specifies the exact length of the MAC to be verified, ranging from 4 to 8 bytes.

hmac

Direction	Type
Input	String

This field contains the MAC or HMAC value to be verified on ONLY and LAST requests, left justified. The caller must provide a MAC or an HMAC value of the required length as determined by the mechanism specified and the length selection rule. This field is ignored for FIRST and MIDDLE requests.

Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

Usage Notes

HMAC operations are performed in hardware or software.

Return code 4, reason code 8000 indicates the HMAC did not verify.

If the FIRST rule is used to start a series of chained calls:

- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the *chain_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

The use of a SHA3 hash method requires an active Enterprise PKCS #11 coprocessor.

PKCS #11 One-Way Hash, Sign, or Verify (CSFPOWH and CSFPOWH6)

Use the PKCS #11 One-Way Hash, Sign, or Verify callable service to generate a one-way hash on specified text, sign specified text, or verify a signature on specified text. For one-way hash, this service supports the following methods:

- MD2 - software only.
- MD5 - software only.
- SHA-1.
- RIPEMD-160 - software only.
- SHA-224.
- SHA-256.
- SHA-384.
- SHA-512.

For sign and verify, the following methods are supported:

- MD2 with RSA-PKCS 1.5.
- MD5 with RSA-PKCS 1.5.
- SHA1 with RSA-PKCS 1.5, RSA-PKCS PSS, DSA, or ECDSA.
- SHA-224 with RSA-PKCS 1.5, RSA-PKCS PSS, DSA, or ECDSA.
- SHA-256 with RSA-PKCS 1.5, RSA-PKCS PSS, DSA, or ECDSA.
- SHA-384 with RSA-PKCS 1.5, RSA-PKCS PSS, DSA, or ECDSA.
- SHA-512 with RSA-PKCS 1.5, RSA-PKCS PSS, DSA, ECDSA, or CRYSTALS-Dilithium (LI2).
- RSA-PKCS PSS without hashing.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPOWH6.

Format

```
CALL CSFPOWH(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             text_length,
             text,
             text_id,
             chain_data_length,
             chain_data,
             handle,
             hash_length,
             hash )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1, 2, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 568. Keywords for PKCS #11 One-Way Hash, Sign, or Verify

Keyword	Meaning
Hash Method (required)	
MD2	Hash algorithm is MD2 algorithm. Length of hash generated is 16 bytes.
MD5	Hash algorithm is MD5 algorithm. Length of hash generated is 16 bytes.
RPMD-160	Hash algorithm is RIPEMD-160. Length of hash generated is 20 bytes.

Table 568. Keywords for PKCS #11 One-Way Hash, Sign, or Verify (continued)	
Keyword	**Meaning**
SHA-1	Hash algorithm is SHA-1. Length of hash generated is 20 bytes.
SHA-224	Hash algorithm is SHA-224. Length of hash generated is 28 bytes.
SHA-256	Hash algorithm is SHA-256. Length of hash generated is 32 bytes.
SHA-384	Hash algorithm is SHA-384. Length of hash generated is 48 bytes.
SHA-512	Hash algorithm is SHA-512. Length of hash generated is 64 bytes.
SHA1LG	Hash algorithm is similar to the SHA-1 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Use this hash method for DSS (applies to One-Way Hash Generate only.) Length of hash generated is 20 bytes. Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.
SHA224LG	Hash algorithm is similar to the SHA-224 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Length of hash generated is 28 bytes. Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.
SHA256LG	Hash algorithm is similar to the SHA-256 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Length of hash generated is 32 bytes. Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.
SHA384LG	Hash algorithm is similar to the SHA-384 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Length of hash generated is 48 bytes. Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.
SHA512LG	Hash algorithm is similar to the SHA-512 algorithm. Use only when *text_length* is greater than or equal to 256 megabytes (512 megabytes on IBM eServer zSeries 990, IBM eServer zSeries 890, or later hardware on HCR7770). Length of hash generated is 64 bytes. Legacy hash values from release HCR7770 and higher prior to APAR OA43937 will be generated for verification purposes with previously archived hash values.
DETERMIN	For use with non-chained RSA signature verifies only. Hash algorithm is to be determined from the input signature.
NULL	For use with non-chained signature generate and verifies only. Hashing is not to be performed. For PSS, the data in text parameter must be the output of the same hashing algorithm specified in the chain_data PSS mask. Can only be specified in combination with SIGN-PSS, or VER-PSS.
Chaining Flag (optional)	
FIRST	Specifies this is the first call in a series of chained calls. Intermediate results are stored in the *hash* and *chain_data* fields. Cannot be specified with hash method DETERMIN.

Table 568. Keywords for PKCS #11 One-Way Hash, Sign, or Verify (continued)	
Keyword	**Meaning**
MIDDLE	Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the *hash* and *chain_data* fields. Cannot be specified with hash method DETERMIN.
LAST	Specifies this is the last call in a series of chained calls. Cannot be specified with hash method DETERMIN.
ONLY	Specifies this is the only call and the call is not chained. This is the default.
Requested Operation (optional)	
HASH	The specified text is to be hashed only. This is the default. Cannot be specified (either explicitly or by default) with hash method DETERMIN.
SIGN-DSA	The data is to be hashed then signed using DSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
SIGN-EC	The data is to be hashed then signed using ECDSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
SIGN-LI2	The data is to be hashed then signed using CRYSTALS-Dilithium. The hash method must be SHA-512.
SIGN-PSS	The data is to be optionally hashed then signed using RSA-PKCS PSS formatting. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, or NULL. The hash method must match the chain_data Hash method.
SIGN-RSA	The data is to be hashed then signed using RSA-PKCS 1.5 formatting. Any hash method is acceptable except RPMD-160 and DETERMIN.
VER-DSA	The data is to be hashed then signature verified using DSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
VER-EC	The data is to be hashed then signature verified using ECDSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
VER-LI2	The data is to be hashed then signature verified using CRYSTALS-Dilithium. The hash method must be SHA-512.
VER-PSS	The data is to be optionally hashed then signature verified using RSA-PKCS PSS formatting. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, or NULL. The Hash method must match the chain_data Hash method.
VER-RSA	The data is to be hashed then signature verified using RSA-PKCS 1.5 formatting. Any hash method is acceptable except RPMD-160. This operation is required for hash method DETERMIN.

text_length

Direction	**Type**
Input	Integer

The length of the text parameter in bytes.

If you specify the FIRST or MIDDLE keyword, the text length must be a multiple of the block size of the hash method. For MD2, this is a multiple of 16 bytes. For MD5, RPMD-160, SHA-1, SHA-224, and SHA-256, this is a multiple of 64 bytes. For SHA-384 and SHA-512, this is a multiple of 128 bytes. For

ONLY and LAST, this service performs the required padding according to the algorithm specified. The length can be from 0 to 2147483647.

If NULL and SIGN-PSS or VER-PSS was specified, *text_length* must match the output length of the SHA hash method specified on the chain_data PSS mask.

text

Direction	Type
Input	String

Value to be hashed

text_id

Direction	Type
Input	Integer

The ALET identifying the space where the text resides.

chain_data_length

Direction	Type
Input/Output	Integer

The byte length of the *chain_data* parameter. This parameter must be 128 bytes.

chain_data

Direction	Type
Input/Output	String

This field is a 128-byte work area. The chain data permits chaining data from one call to another and allows for the passing of initialization values for specific operations. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

Table 569. chain_data parameter format on input (FIRST and ONLY for SIGN-PSS and VER-PSS)

Offset	Length	Initialization values provided by the caller
0	4	Hash method. Must match the method specified in rule_array if NULL was not specified. 0x00000220 (CKM_SHA_1) 0x00000255 (CKM_SHA224) 0x00000250 (CKM_SHA256) 0x00000260 (CKM_SHA384) 0x00000270 (CKM_SHA512)
4	4	PSS mask generation function. Must be the same digest method as the Hash method. 0x00000001 (CKG_MGF1_SHA1), 0x00000005 (CKG_MGF1_SHA224), 0x00000002 (CKG_MGF1_SHA256), 0x00000003 (CKG_MGF1_SHA384), 0x00000004 (CKG_MGF1_SHA512) only.

Table 569. chain_data parameter format on input (FIRST and ONLY for SIGN-PSS and VER-PSS) (continued)

Offset	Length	Initialization values provided by the caller
8	4	PSS salt length in bytes. For SIGN-PSS, the value must be 0 or the size of the hash generated by the Hash Method and must be less than or equal to the maximum salt length specified by the PKCS #1 standard for the RSA PSS mechanism. For VER-PSS, the value must be less than or equal to the maximum salt length specified by the PKCS #1 standard for the RSA PSS mechanism.
12	116	Reserved – to be initialized by ICSF

Table 570. chain_data parameter format on input (FIRST and ONLY for non-PSS operations)

Offset	Length	Description
0	128	Reserved – to be initialized by ICSF

Table 571. chain_data parameter format on output (all calls) and input (MIDDLE and LAST)

Offset	Length	Description
0	4	Flag word **Bit** **Meaning when set on** **0** Cryptographic state object has been allocated **1-31** Reserved for IBM's use
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

handle

Direction	Type
Input	String

For hash requests, this is the 44-byte name of the token to which this hash operation is related. The first 32 bytes of the handle are meaningful. The remaining 12 bytes are reserved. See "Handles" on page 111 for the format of a *handle*.

For sign and verify requests, this is the 44-byte handle to the key object that is to be used. For FIRST and MIDDLE chaining requests, only the first 32 bytes of the handle are meaningful, to identify the token.

hash_length

Direction	Type
Input/Output	Integer

The length of the supplied hash field in bytes.

For hash requests, this field is input only. For SHA-1 and RPMD-160 this must be at least 20 bytes; for MD2 and MD5 this must be at least 16 bytes.

For SHA-224 and SHA-256, this must be at least 32 bytes. For SHA-224, 32 bytes are returned, the hash value is the left most 28 bytes and padded with 4 bytes of binary zeroes. For SHA-384 and

SHA-512, this must be at least 64 bytes. For SHA-384, 64 bytes are returned, the hash value is the left most 48 bytes and padded with 16 bytes of binary zeroes.

For FIRST and MIDDLE sign and verify requests, this field is ignored.

For LAST and ONLY sign requests, this field is input/output. If the signature generation is successful, ICSF will update this field with the length of the generated signature. If the signature generation is unsuccessful because the supplied hash field is too small, ICSF will update this field with the required length.

For LAST and ONLY verify requests, this field is input only.

hash

Direction	Type
Input/Output	String

This field contains the hash or signature, left-justified. The processing of the rest of the field depends on the implementation.

For hash requests, this field is the generated hash. If you specify the FIRST or MIDDLE keyword, this field contains the intermediate hash value. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message.

For FIRST and MIDDLE sign and verify requests, this field is ignored.

For LAST and ONLY sign requests, this field is the generated signature.

For LAST and ONLY verify requests, this field is input signature to be verified.

Authorization

To use this service to sign or verify with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service to sign or verify with a private object, the caller must have at least USER (READ) authority (user access).

Usage notes

DSA, ECDSA, and RSA operations may be done in hardware or software.

Clear key RSA-PKCS PSS operations can be performed on an accelerator or CCA Crypto express adapter if available. CCA RSA-PKCS PSS support requires a CEX5 with CCA release 5.3 or later and associated control points enabled (see the access control points for the CCA PKA services: PKA Key Import (CSNDPKI), Digital Signature Generate (CSNDDSG), and Digital Signature Verify (CSNDDSV)).

If the FIRST rule is used to start a series of chained calls, the application must not change the Hash Method or Requested Operation rules between the calls. The behavior of the service is undefined if the rules are changed.

If the FIRST rule is used to start a series of chained calls, the application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the *chain_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

The CSFSERV resource name that protects this service is CSFOWH, the same resource name used to protect the non-PKCS #11 One Way Hash service.

If the CSF.CSFSERV.AUTH.CSFOWH.DISABLE resource profile is defined in the XFACILIT SAF resource class, no SAF authorization checks will be performed against the CSFSERV class when using this service. If CSF.CSFSERV.AUTH.CSFOWH.DISABLE is not defined, the SAF authorization check will be performed. Disabling the SAF check may improve the performance of your application.

For hash method DETERMIN, ICSF determines the hashing method by RSA decrypting the input signature using the specified public key and examining the result. ICSF will return the “signature did not verify” error (return code 4, reason code X'2AF8') if this process is unsuccessful for any of the following reasons:

1. ICSF cannot successfully perform the decryption because the public key is the wrong size.
2. The resulting clear text block is not properly RSA-PKCS 1.5 formatted.
3. The resulting clear text block indicates a hashing algorithm not supported by this service was used.

PKCS #11 Private Key Sign (CSFPPKS and CSFPPKS6)

Use the PKCS #11 Private Key Sign callable service to:

- Decrypt or sign data using an RSA private key using zero-pad or PKCS #1 v1.5 formatting.
- Sign data using a CRYSTALS-Dilithium (LI2) private key.
- Sign data using a DSA private key.
- Sign data using an Elliptic Curve private key in combination with DSA.

The key handle must be a handle of a PKCS #11 private key object. When the request type keyword DECRYPT is specified in the rule array, CKA_DECRYPT attribute must be true. When no request type is specified, the CKA_SIGN attribute must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPKS6.

Format

```
CALL CSFPPKS(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              cipher_value_length,
              cipher_value,
              key_handle,
              clear_value_length,
              clear_value )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array_parameter*. This value may be 1 or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service.

Table 572. Keywords for private key sign

Keyword	Meaning
Mechanism (One of the following must be specified)	
DSA	Mechanism is DSA signature generation
ECDSA	Mechanism is Elliptic Curve with DSA signature generation
EC-SDSA	Mechanism is non-ECDSA, non-EDDSA EC-based signature, generally Schnorr variants.
EDDSA	Mechanism is PureEdDSA signature generation (no pre-hashing)
LI2	Mechanism is a CRYSTALS-Dilithium signature generation.
RSA-PKCS	Mechanism is RSA decryption or signature generation using PKCS #1 v1.5 formatting
RSA-ZERO	Mechanism is RSA decryption or signature generation using zero-pad formatting
Request type (optional)	
DECRYPT	The request is to decrypt data. This type of request requires the CKA_DECRYPT attribute to be true. If DECRYPT is not specified, the CKA_SIGN attribute must be true. Valid with RSA only.
Schnorr Subvariant (One, required with EC-SDSA)	
RANDOM	Randomized Schnorr signature; no pre-hashing, SHA-256 only.
COMPMULT	Randomized Schnorr signature with compressed keys and including the signing party’s public key, SHA-256 only.

cipher_value_length

Direction	Type
Input	Integer

Length of the *cipher_value* parameter in bytes.

cipher_value

Direction	Type
Input	String

For decrypt, this is the value to be decrypted. Otherwise, this is the value to be signed.

- For DSA and ECDSA signature requests, the data to be signed is expected to be a SHA1, SHA224, SHA256, SHA384, or SHA512 digest.
- For EC-SDSA signature requests, the data to be signed is expected to be a SHA1, SHA224, or SHA256 digest.
- For CRYSTALS-Dilithium signature requests,
  - The data to be signed is from zero to 5120 bytes.
- For EDDSA signature requests,
  - When using a clear key, the data to be signed is from zero to 2^{14} (16384) bytes.
  - When using a secure key, the data to be signed is from zero to 2^{13} (8192) bytes.
- For RSA-PKCS signature requests, the data to be signed is expected to be a DER encoded DigestInfo structure.

key_handle

Direction	Type
Input	String

The 44-byte handle of a private key object. See “Handles” on page 111 for the format of a *key_handle*.

clear_value_length

Direction	Type
Input/Output	Integer

Length of the *clear_value* parameter in bytes. On input, this must be at least:

- For RSA, modulus in bytes.
- For DSA, two times the length of the subprime q value.
- For ECDSA and EdDSA, two times the private key size.
- For CRYSTALS-Dilithium signatures, this must be at least the signature size for the keysize:
  - 3366 bytes for Dilithium(6,5) Round 2.
  - 3293 bytes for Dilithium(6,5) Round 3.
  - 4668 bytes for Dilithium(8,7) Round 2.
  - 4595 bytes for Dilithium(8,7) Round 3.

On output, this is updated to be the actual length of the decrypted value or the generated signature.

clear_value

Direction	Type
Output	String

For decrypt, this field will contain the decrypted value. Otherwise this field will contain the generated signature.

Authorization

To use this service with a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have USER (READ) authority (user access).

Usage Notes

Operations may be done in hardware or software.

- Signing with a secure private key requires an EP11 coprocessor.
- Signing using a clear RSA private key will use:
  - An accelerator (preferred), if available.
  - A CCA coprocessor (if no accelerators) if the private key is modulus-exponent form and the operation does not require FIPS compliance.
- On IBM z15 and IBM z15 T02 (or later hardware), CP Assist for Cryptographic Functions will be used for signing with clear or protected private key for the following elliptic curves:
  - secp256r1.
  - secp384r1.
  - secp521r1.
  - Curve25519 (twisted Edwards form).
  - Curve448 (twisted Edwards form).
- In all other cases, software will be used.

Request type DECRYPT is not supported for an Elliptic Curve, DSA, or CRYSTALS-Dilithium private key.

For rule EC-SDSA and each subvariant, the signing key must be a secure key.

PKCS #11 Public Key Verify (CSFPPKV and CSFPPKV6)

Use the PKCS #11 Public Key Verify callable service to:

- Encrypt or verify data using an RSA public key using zero-pad or PKCS #1 v1.5 formatting. For encryption, the encrypted data is returned.
- Verify a signature using a CRYSTALS-Dilithium public key. No data is returned.
- Verify a signature using a DSA public key. No data is returned.
- Verify a signature using an Elliptic Curve public key in combination with DSA. No data is returned.

The key handle must be a handle of a PKCS #11 public key object. When the request type keyword ENCRYPT is specified in the rule array, CKA_ENCRYPT attribute must be true. When no request type is specified, the CKA_VERIFY attribute must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPKV6.

Format

```
CALL CSFPPKV(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              clear_value_length,
              clear_value,
              key_handle,
              cipher_value_length,
              cipher_value )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1 or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service.

Table 573. Keywords for public key verify

Keyword	Meaning
Mechanism (One of the following must be specified)	
DSA	Mechanism is DSA signature verification
ECDSA	Mechanism is Elliptic Curve with DSA signature verification
EC-SDSA	Mechanism is non-ECDSA, non-EDDSA EC-based signature, generally Schnorr variants.
EDDSA	Mechanism is PureEdDSA signature verification (no pre-hashing)
LI2	Mechanism is a CRYSTALS-Dilithium signature verification.
RSA-PKCS	Mechanism is RSA encryption or signature verification using PKCS #1 v1.5 formatting
RSA-ZERO	Mechanism is RSA encryption or signature verification using zero-pad formatting
Request type (optional)	
ENCRYPT	The request is to encrypt data. This type of request requires the CKA_ENCRYPT attribute to be true. If ENCRYPT is not specified, the CKA_VERIFY attribute must be true. Valid with RSA only.
Schnorr Subvariant (One, required with EC-SDSA)	
RANDOM	Randomized Schnorr signature; no pre-hashing, SHA-256 only.
COMPMULT	Randomized Schnorr signature with compressed keys and including the signing party's public key, SHA-256 only.

clear_value_length

Direction	Type
Input	Integer

The length of the clear_value parameter

clear_value

Direction	Type
Input	String

For encrypt, this is the value to be encrypted. Otherwise, this is the signature to be verified.

key_handle

Direction	Type
Input	String

The 44-byte handle of public key object. See "Handles" on page 111 for the format of a *key_handle*.

cipher_value_length

Direction	Type
Input/Output	Integer

For encrypt, on input, this is the length of the *cipher_value* parameter in bytes. On output, this is updated to be the actual length of the text encrypted into the *cipher_value* parameter. For signature verification, this is the length of the data to be verified (input only).

cipher_value

Direction	Type
Input	String

For encrypt, this is the encrypted value (output only).

- For CRYSTALS-Dilithium signature verification requests,
  - The data to be verified is from zero to 5120 bytes.
- For DSA and ECDSA signature verification requests, the data to be verified is expected to be a SHA1, SHA224, SHA256, SHA384, or SHA512 digest.
- For EC-SDSA signature requests, the data to be signed is expected to be a SHA1, SHA224, or SHA256 digest.
- For EDDSA signature requests,
  - When using a clear key, the data to be verified is from zero to 2^{14} (16384) bytes.
  - When using a secure key, the data to be verified is from zero to 2^{13} (8192) bytes.
- For RSA-PKCS signature verification requests, the data to be verified is expected to be a DER encoded DigestInfo structure.
- For signature verification, this is the data to be verified (input only).

Authorization

To use this service with a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have USER (READ) authority (user access).

Usage Notes

Operations may be done in hardware or software.

- Verification with a secure public key prefers an EP11 coprocessor (if present). If none is present, verification will occur in software.
- Verification using a clear RSA public key will use:
  - An accelerator (preferred), if available.
  - A CCA coprocessor (if no accelerators) if the operation does not require FIPS compliance.
- Verification using a clear EC public key: If the operation does not require FIPS compliance, a CCA coprocessor will be used for the following elliptic curves:
  - secp256r1.
  - secp384r1.
  - secp521r1.
  - Curve25519 (the twisted Edwards form).
  - Curve448 (the twisted Edwards form).
  - secp256k1.
- On IBM z15 and IBM z15 T02 (or later hardware), CP Assist for Cryptographic Functions will be used for signing with clear public key for the following elliptic curves:
  - secp256r1.
  - secp384r1.

  - secp521r1.
  - Curve25519 (twisted Edwards form).
  - Curve448 (twisted Edwards form).
- In all other cases, software will be used.

Request type ENCRYPT is not supported for an Elliptic Curve, DSA, or CRYSTALS-Dilithium public key.

For rule EC-SDSA and each subvariant, the verifying key must be a secure key.

PKCS #11 Pseudo-Random Function (CSFPPRF and CSFPPRF6)

Use the PKCS #11 Pseudo-Random callable service to generate pseudo-random output of arbitrary length. This service does not support any recovery methods.

The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input/output data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPRF6.

Format

```
CALL CSFPPRF(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             handle,
             parms_list_length,
             parms_list,
             prf_output_length,
             prf_output)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 574. Keywords for PKCS #11 Pseudo-random function

Keyword	Meaning
Mechanism (required)	
TLS-PRF	Use the TLS Pseudo-Random Function derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_PRF. This mechanism derives deterministic random bytes from a caller supplied secret key object and other parameters.
PRNG	Generate pseudo-random bytes using the best source available. Possible sources are: CCA coprocessors, Enterprise PKCS #11 coprocessors, the CP Assist for Cryptographic Function, or a pseudo (deterministic) random algorithm. CCA coprocessors are only used for entropy seeding when ICSF is running in FIPS standard mode or FIPS compatibility mode and the CP Assist for Cryptographic Functions (z/Architecture cryptographic facility) Message-Security-Assist Extension 7 is not available.
PRNGFIPS	Generate pseudo-random bytes consistent with NIST SP 800-90A using the best source available. Possible sources are: Enterprise PKCS #11 coprocessors, the CP Assist for Cryptographic Function, or a pseudo (deterministic) random algorithm. CCA coprocessors are only used for entropy seeding and only when the CP Assist for Cryptographic Functions (z/Architecture cryptographic facility) Message-Security-Assist Extension 7 is not available.

handle

Direction	Type
Input	String

For mechanism TLS-PRF, this is the 44-byte handle of the source secret key object. The CKA_DERIVE attribute for the secret key object must be true. If no key is to be used, set the handle to all blanks.

For mechanisms PRNG and PRNGFIPS, this is the 44-byte name of the token to which this operation is related. The first 32 bytes of the handle are meaningful. The remaining 12 bytes are reserved and must be blanks.

See “Handles” on page 111 for the format of a *handle*.

parms_list_length

Direction	Type
Input	Integer

The length of the parameters supplied in the *parms_list* parameter in bytes.

parms_list

Direction	Type
Input/Output	String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 575. parms_list parameter format for TLS-PRF mechanism

Offset	Length in bytes	Direction	Description
0	1	input	PRF function code – x'00', use combined MD5/SHA1 digest algorithm as defined in TLS 1.0/1.1, otherwise use the following single digest algorithm as defined in TLS 1.2: x'01' = SHA256, x'02' = SHA384, and x'03' = SHA512
1	3	not applicable	reserved
4	4	input	length in bytes of the label (x). where 1 <= length <= 256
8	4	input	length in bytes of the seed (y), where 1 <= length <= 256
12	x	input	label
12+x	y	input	seed

For the PRNG and PRNGFIPS mechanisms, there are no parameters. The *parms_list_length* parameter must be set to zero for this mechanism

prf_output_length

Direction	Type
Input	Integer

The length in bytes of pseudo-random data to be generated and returned in the *prf_output* parameter. The maximum length is 2147483647 bytes.

prf_output

Direction	Type
Output	String

The pre-allocated area in which the pseudo-random data is returned.

Authorization

To use this service with a public object for mechanism TLS-PRF, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object for mechanism TLS-PRF, the caller must have at least USER (READ) authority (user access).

Usage Notes

Pseudo-random functions operations are performed in hardware or software.

The CSFSERV resource name that protects this service is CSFRNG, the same resource name used to protect the non-PKCS #11 Random Number Generation service.

If the CSF.CSFSERV.AUTH.CSFRNG.DISABLE SAF resource profile is defined in the XFACILIT SAF resource class, no SAF authorization checks will be performed against the CSFSERV class when using this service. If CSF.CSFSERV.AUTH.CSFRNG.DISABLE is not defined, the SAF authorization check will be performed. Disabling the SAF check may improve the performance of your application.

PKCS #11 Set Attribute Value (CSFPSAV and CSFPSAV6)

Use the PKCS #11 Set Attribute Value callable service (CSFPSAV) to update the attributes of an object.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSAV6.

Format

```
CALL CSFPSAV(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               handle,
               rule_array_count,
               rule_array,
               attribute_list_length,
               attribute_list)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

handle

Direction	Type
Input	String

The 44-byte handle of the object. See “Handles” on page 111 for the format of a *handle*.

rule_array_count

Direction	Type
Input	Integer

The number of keywords supplied in the *rule_array* parameter. This value must be 0.

rule_array

Direction	Type
Ignored	String

This field is ignored.

attribute_list_length

Direction	Type
Input	Integer

The length of the *attribute_list* parameter in bytes.

The minimum value for this field is 2 and the maximum value for this field is 32752.

attribute_list

Direction	Type
Input	String

A list of object attributes.

Note: Lengths in the attribute list and attribute structures are unsigned integers.

See “Attribute list” on page 111 for the format of an *attribute_list*.

Authorization

Table 576. Authorization requirements for the set attribute value callable service

Action	Object	Authority required
Set	Public object, except a CA certificate	USER (UPDATE) or SO (READ)
Set	Private object, except a CA certificate	USER (UPDATE) or SO (CONTROL)
Set	Public CA certificate object	USER (CONTROL) or SO (READ)
Set	Private CA certificate object	USER (CONTROL) or SO (CONTROL)

Note:

- Session and token objects require the same authority.
- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles and how ICSF determines that a certificate is a CA certificate.

Usage Notes

When updating the attributes of an object, all attributes in the template will be processed and the value used is that of the last instance processed.

Key pair generation may be done in hardware or software.

PKCS #11 Secret Key Decrypt (CSFPSKD and CSFPSKD6)

Use the PKCS #11 Secret Key Decrypt callable service to decipher data using a symmetric key. AES, Blowfish, CHACHA20-Poly1305, DES, and RC4 are supported. This service supports CBC, CTR, ECB, Galois/Counter, and stream modes as well as PKCS #7 padding. The key handle must be a handle of a PKCS #11 secret key object. The CKA_DECRYPT attribute must be true.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the *clear_text_length* parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSKD6.

Format

```
CALL CSFPSKD(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_handle,
              initialization_vector_length,
              initialization_vector,
              chain_data_length,
              chain_data,
              cipher_text_length,
              cipher_text,
              cipher_text_id,
              clear_text_length,
              clear_text,
              clear_text_id )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 0, 1, 2, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service.

Table 577. Keywords for Secret Key Decrypt

Keyword	Meaning
Encryption Mechanism (Optional. No default. If not specified, mechanism will be taken from key type of secret key. If specified, must match key type.)	
AES	AES algorithm will be used.
CHACHA20	CHACHA20 algorithm will be used with Poly1305 for authentication.
DES	DES algorithm will be used. This is only single-key encryption.
DES3	DES3 algorithm will be used, This includes double-key and triple-key decryption.
BLOWFISH	BLOWFISH algorithm will be used.
RC4	RC4 algorithm will be used. This is a stream cipher.
Processing Rule (optional)	
CBC	Performs cipher block chaining. The cipher text length must be a multiple of the block size for the specified algorithm (8 bytes for DES, DES3, and BLOWFISH, 16 bytes for AES). CBC is the default value for AES, BLOWFISH, DES, and DES3. CBC cannot be specified for CHACHA20 or RC4.

Table 577. Keywords for Secret Key Decrypt (continued)	
Keyword	**Meaning**
CBC-CS	Performs cipher block chaining with ciphertext stealing. For INITIAL and CONTINUE calls, the cipher text length must be a multiple of the block size for the specified algorithm. For FINAL and ONLY calls, the cipher text length must be at least one block size and ciphertext stealing will be performed if not a multiple of the block size. The clear text will always be the same length as the cipher text. CBC-CS can only be specified for AES.
CBC-PAD	Performs cipher block chaining. The cipher text length must be greater than zero and a multiple of the block size for the specified algorithm. For FINAL and ONLY calls, PKCS #7 padding is performed. For this reason, the clear text will always be shorter than the cipher text and may even be zero length. CBC-PAD cannot be specified for BLOWFISH, CHACHA20, or RC4.
CTR	Performs counter mode decryption. The cipher text length must be greater than zero. The clear text will be the same length as the cipher text. CTR may only be specified for AES.
ECB	Performs electronic code book decryption. The cipher text length must be a multiple of the block size for the specified algorithm. ECB cannot be specified for BLOWFISH, CHACHA20, or RC4.
GCM	Performs Galois/Counter mode decryption. The cipher text length must be greater than zero. The clear text will be shorter than the cipher text and may even be zero length due to the truncation of the authentication tag. GCM may only be specified with AES. GMAC is a specialized form of GCM where no plain text is returned.
STREAM	Performs a stream cipher. STREAM cannot be specified for AES, BLOWFISH, DES, or DES3. STREAM is the default value for CHACHA20 and RC4.
Chaining Selection (optional)	
INITIAL	Specifies this is the first call in a series of chained calls. For cipher block chaining, the initialization vector is taken from the *initialization_vector* parameter. Cannot be specified with processing rules ECB or GCM, or with mechanism CHACHA20.
CONTINUE	Specifies this is a middle call in a series of chained calls. Intermediate results are read from and stored in the *chain_data* field. Cannot be specified with processing rule ECB or GCM, or with mechanism CHACHA20.
FINAL	Specifies this is the last call in a series of chained calls. Intermediate results are read from the *chain_data* field. Cannot be specified with processing rule ECB or GCM, or with mechanism CHACHA20.
ONLY	Specifies this is the only call and the call is not chained. For cipher block chaining, the initialization vector is taken from the *initialization_vector* parameter. For CHACHA20, CTR, and GCM, the initialization parameters are taken from the *initialization_vector* parameter. ONLY is the default chaining.

key_handle

Direction	**Type**
Input	String

The 44-byte handle of secret key object or the raw object returned by PKCS #11 Token Record Create. See “Handles” on page 111 for the format of a *key_handle* or handle parameter of PKCS #11 Token Record Create.

initialization_vector_length

Direction	Type
Input	Integer

Length of the *initialization_vector* in bytes. For CBC and CBC-PAD, this must be 8 bytes for DES and BLOWFISH and 16 bytes for AES. For CBC-CS, this must be 16 bytes for AES. For CTR, this must be 17 bytes (see the structure described in the *initialization_vector* parameter). For GCM and CHACHA20, this must be 28 bytes (see the corresponding structure described in the *initialization_vector* parameter).

initialization_vector

Direction	Type
Input	String

This field has a varying format depending on the mechanism specified. For CBC, CBC-CS, and CBC-PAD, this is the 8 byte or 16 byte initial chaining value. The formats for CHACHA20, CTR, and GCM are shown in the following tables.

Table 578. initialization_vector parameter format for GCM mechanism and CHACHA20 mechanisms

Offset	Length in bytes	Direction	Description
0	4	Input	Length in bytes of the initialization vector. For GCM: The minimum value is 1. The maximum value is 128. 12 is recommended. For CHACHA20: This value must be 12 (the 96-bit nonce).
4	8	Input	64-bit address of the initialization vector. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers.
12	4	Input	Length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.
16	8	Input	64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. For GCM: This value must be one of 4, 8, 12, 13, 14, 15, or 16. For CHACHA20: This value must be 16.

Table 579. initialization_vector parameter format for CTR mechanism

Offset	Length in bytes	Direction	Description
0	16	Input	Initial counter block.
16	1	Input	The number of low order bytes of the counter to be incremented. The remaining upper order bytes are the nonce. Valid values are 1 to the block size, inclusive. The block size is sixteen for AES.

chain_data_length

Direction	Type
Input/Output	Integer

The byte length of the chain_data parameter. For all processing rules except ECB, this must be 128 bytes.

chain_data

Direction	Type
Input/Output	String

For all processing rules except ECB, this field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on an INITIAL call, and may change it on subsequent CONTINUE calls. Your application must not change the data in this field between the sequence of INITIAL, CONTINUE, and FINAL calls for a specific message. The chain data has the following format:

Table 580. chain_data parameter format

Offset	Length	Description
0	4	Flag word **Bit** **Meaning when set on** **0** Cryptographic state object has been allocated **1-31** Reserved for IBM's use
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

cipher_text_length

Direction	Type
Input	Integer

Length of the *cipher_text* parameter in bytes. Except for processing rule GCM and encryption mode CHACHA20, the length can be up to 2147483647. For processing rule GCM and encryption mode CHACHA20, the length cannot exceed 1048576 plus the length of the tag.

cipher_text

Direction	Type
Input	String

Text to be decrypted. For processing rule GCM and encryption mode CHACHA20, the authentication tag is appended to the actual cipher text.

cipher_text_id

Direction	Type
Input	Integer

The ALET identifying the space where the cipher text resides.

clear_text_length

Direction	Type
Input/Output	Integer

On input, the length in bytes of the *clear_text* parameter. On output, the length of the text decrypted into the *clear_text* parameter

clear_text

Direction	Type
Output	String

Decrypted text

clear_text_id

Direction	Type
Input	Integer

The ALET identifying the space where the clear text resides.

Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

Usage Notes

If the INITIAL rule is used to start a series of chained calls:

- The same key_handle, Encryption Mechanism and Processing Rule must be used on the subsequent CONTINUE and FINAL calls.
- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a FINAL call to free ICSF resources allocated. If processing is to be aborted without making a FINAL call and the *chain_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

GCM decryption may be used to verify a GMAC on some authentication data. To do this, request AES decryption with processing rule. The *cipher_text_length* and *cipher_text* fields must be set to the length and value of the GMAC to be verified. A *return_code* of zero and no *clear_text* data returned means the GMAC verification was successful.

A secure key may not be used for Processing Rules CBC-CS, CTR, or GCM, or for encryption mode CHACHA20.

PKCS #11 Secret Key Encrypt (CSFPSKE and CSFPSKE6)

Use the PKCS #11 Secret Key Encrypt callable service to encipher data using a symmetric key. AES, Blowfish, CHACHA20-Poly1305, DES, and RC4 are supported. This service supports CBC, CTR, ECB, Galois/Counter, and stream modes as well as PKCS #7 padding. The key handle must be a handle of a PKCS #11 secret key object. The CKA_ENCRYPT attribute must be true.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the *cipher_text_length* parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSKE6.

Format

```
CALL CSFPSKE(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              key_handle,
              initialization_vector_length,
              initialization_vector,
              chain_data_length,
              chain_data,
              clear_text_length,
              clear_text,
              clear_text_id,
              cipher_text_length,
              cipher_text,
              cipher_text_id )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 0, 1, 2, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service.

Table 581. Keywords for Secret Key Encrypt

Keyword	Meaning
Encryption Mechanism (Optional. No default. If not specified, mechanism will be taken from key type of secret key. If specified, must match key type.)	
AES	AES algorithm will be used.
CHACHA20	CHACHA20 algorithm will be used with Poly1305 for authentication.
DES	DES algorithm will be used. This is only single-key encryption.
DES3	DES3 algorithm will be used, This includes double- and triple-key encryption.
BLOWFISH	BLOWFISH algorithm will be used.
RC4	RC4 algorithm will be used. This is a stream cipher.
Processing Rule (optional)	
CBC	Performs cipher block chaining. The cipher text length must be a multiple of the block size for the specified algorithm (8 bytes for DES, DES3, and BLOWFISH, 16 bytes for AES). CBC is the default value for AES, BLOWFISH, DES, and DES3. CBC cannot be specified for CHACHA20 or RC4.
CBC-CS	Performs cipher block chaining with ciphertext stealing. For INITIAL and CONTINUE calls, the clear text length must be a multiple of the block size for the specified algorithm. For FINAL and ONLY calls, the clear text length must be at least one block size and ciphertext stealing will be performed if not a multiple of the block size. The cipher text will always be the same length as the clear text. CBC-CS can only be specified for AES.
CBC-PAD	Performs cipher block chaining. Except for FINAL and ONLY chaining calls, the clear text length must be a multiple of the block size for the specified algorithm. For FINAL and ONLY calls: • The clear text length may be shorter than the block size and may even be zero. • PKCS #7 padding is performed. Thus, the cipher text will always be longer than the clear text. CBC-PAD cannot be specified for BLOWFISH, CHACHA20, or RC4.
CTR	Performs counter mode encryption. The clear text length must be greater than zero. The cipher text will be the same length as the clear text. CTR may only be specified for AES.
ECB	Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm. ECB cannot be specified for BLOWFISH, CHACHA20, or RC4.
GCM	Performs Galois/Counter mode encryption. The clear text length may be shorter than the block size and may even be zero. The authentication tag is returned appended to the cipher text. GCM may only be specified with AES. GMAC is a specialized form of GCM where no plain text is specified.

Table 581. Keywords for Secret Key Encrypt (continued)	
Keyword	**Meaning**
GCMIVGEN	Performs similarly to the GCM processing rule except that ICSF will generate part of the initialization vector and return it in the *initialization_vector* parameter. Having ICSF generate the initialization vector ensures that initialization vectors are never repeated for a given key object.
STREAM	Performs a stream cipher. STREAM cannot be specified for AES, BLOWFISH, DES, or DES3. STREAM is the default value for CHACHA20 and RC4. For CHACHA20, the authentication tag is returned appended to the cipher text.
Chaining Selection (optional)	
INITIAL	Specifies this is the first call in a series of chained calls. For cipher block chaining, the initialization vector is taken from the *initialization_vector* parameter. Cannot be specified with processing rules ECB , GCM, or GCMIVGEN, or with mechanism CHACHA20.
CONTINUE	Specifies this is a middle call in a series of chained calls. Intermediate results are read from and stored in the *chain_data* field. Cannot be specified with processing rule ECB, GCM, or GCMIVGEN, or with mechanism CHACHA20.
FINAL	Specifies this is the last call in a series of chained calls. Intermediate results are read from the *chain_data* field. Cannot be specified with processing rule ECB, GCM, or GCMIVGEN, or with mechanism CHACHA20
ONLY	Specifies this is the only call and the call is not chained. For cipher block chaining, the initialization vector is taken from the *initialization_vector* parameter. For CHACHA20, CTR, and GCM, the initialization parameters are taken from the *initialization_vector* parameter. ONLY is the default chaining.

key_handle

Direction	**Type**
Input	String

The 44-byte handle of secret key object or the raw object returned by PKCS #11 Token Record Create. See “Handles” on page 111 for the format of a *key_handle* or handle parameter of PKCS #11 Token Record Create.

initialization_vector_length

Direction	**Type**
Input	Integer

Length of the *initialization_vector* in bytes. For CBC and CBC-PAD, this must be 8 bytes for DES and BLOWFISH and 16 bytes for AES. For CBC-CS, this must be 16 bytes for AES. For CTR, this must be 17 bytes (see the structure described in the *initialization_vector* parameter). For GCM, GCMIVGEN, and CHACHA20, this must be 28 bytes (see the corresponding structure described in the *initialization_vector* parameter).

initialization_vector

Direction	**Type**
Input	String

This field has a varying format depending on the mechanism specified. For CBC, CBC-CS, and CBC-PAD, this is the 8 byte or 16 byte initial chaining value. The formats for CHACHA20, CTR, GCM, and GCMIVGEN are shown in the following tables.

Table 582. initialization_vector parameter format for GCM mechanism and CHACHA20 mechanisms

Offset	Length in bytes	Direction	Description
0	4	Input	Length in bytes of the initialization vector. For GCM: The minimum value is 1. The maximum value is 128. 12 is recommended. For CHACHA20: This value must be 12.
4	8	Input	64-bit address of the initialization vector area. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers.
12	4	Input	length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.
16	8	Input	64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. For GCM: This value must be one of 4, 8, 12, 13, 14, 15, or 16. For CHACHA20: This value must be 16.

Table 583. initialization_vector parameter format for GCMIVGEN mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Nonce value which ICSF is to use as the first 4 bytes of the initialization vector. The remaining 8 bytes will be generated and returned to the caller in the initialization vector area.
4	8	Input	64-bit address of the initialization vector area into which ICSF will store the 8 bytes it generates. The area must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. The complete initialization vector to be used for decryption is the 4-byte nonce concatenated with the 8 bytes stored in the area
12	4	Input	length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.
16	8	Input	64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

Table 584. initialization_vector parameter format for CTR mechanism

Offset	Length in bytes	Direction	Description
0	16	Input	Initial counter block.
16	1	Input	The number of low order bytes of the counter to be incremented. The remaining upper order bytes are the nonce. Valid values are 1 to the block size, inclusive. The block size is sixteen for AES.

chain_data_length

Direction	Type
Input/Output	Integer

The byte length of the *chain_data* parameter. For all processing rules except ECB, this must be 128 bytes.

chain_data

Direction	Type
Input/Output	String

For all processing rules except ECB, this field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on an INITIAL call, and may change it on subsequent CONTINUE calls. Your application must not change the data in this field between the sequence of INITIAL, CONTINUE, and FINAL calls for a specific message. The chain data has the following format:

Table 585. chain_data parameter format

Offset	Length	Description
0	4	Flag word **Bit** **Meaning when set on** **0** Cryptographic state object has been allocated **1-31** Reserved for IBM's use
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

clear_text_length

Direction	Type
Input	Integer

Length of the *clear_text* parameter in bytes. Except for processing rules GCM and GCMIVGEN, the length can be up to 2147483647. For processing rules GCM and GCMIVGEN, the length cannot exceed 1048576.

clear_text

Direction	Type
Input	String

Text to be encrypted

clear_text_id

Direction	Type
Input	Integer

The ALET identifying the space where the clear text resides.

cipher_text_length

Direction	Type
Input/Output	Integer

On input, the length in bytes of the *cipher_text* parameter. On output, the length of the text encrypted into the *cipher_text* parameter, including the authentication tag for processing rule GCM or encryption mode CHACHA20.

cipher_text

Direction	Type
Output	String

Encrypted text. For processing rule GCM and encryption mode CHACHA20, the authentication tag is appended to the actual cipher text.

cipher_text_id

Direction	Type
Output	Integer

The ALET identifying the space where the cipher text resides.

Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

Usage Notes

If the INITIAL rule is used to start a series of chained calls:

- The same key_handle, Encryption Mechanism and Processing Rule must be used on the subsequent CONTINUE and FINAL calls.
- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a FINAL call to free ICSF resources allocated. If processing is to be aborted without making a FINAL call and the *chain_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

GCM encryption may be used to produce a GMAC on some authentication data. To do this, request AES encryption with processing rule GCM or GCMIVGEN. The *clear_text_length* field must be set to zero. The authentication tag (the GMAC) is returned in the *cipher_text* field.

For Processing Rule GCMIVGEN, the total number of initialization vector generations for a token key object is limited to 4294967295. Once this number is exceeded, the key object will no longer be eligible for Processing Rule GCMIVGEN and is considered “retired”. This usage counter is maintained in the TKDS as part of the key object. For keys that are copied using CSFPTRC (C_CopyObject), the existing counter value is copied to the new key object, but not synchronized after that.

For Processing Rule GCMIVGEN, session key objects have no maximum lifetime. They may be retired at any time. Once retired, the key object will no longer be eligible for Processing Rule GCMIVGEN.

For Processing Rule GCMIVGEN, the nonce value portion of the initialization vector is predetermined by the caller. It is used to ensure that initialization vector values are not repeated for any given key value. The caller should provide a random value and change the value as often as practical. It must be changed whenever:

- A given key value is replicated as a new persistent key object
- A given persistent key object is replicated as a new session key object
- A given session key value is re-instantiated after system IPL
- A given key value is re-instantiated after ICSF indicates it has been retired

Use of Processing Rule GCMIVGEN with token key objects requires that the first 4 bytes of ECVTSPLX or CVTSNAME be set to a unique value with respect to other systems. See *z/OS Cryptographic Services ICSF System Programmer's Guide* for information on how to set these fields.

A session key object should never be used for Processing Rule GCMIVGEN if the key value is distributed to multiple systems outside the current sysplex where new initialization vectors may be generated. Use only token key objects in such cases. If session key objects are used, the other systems must use different nonces.

For Processing Rule GCMIVGEN, the 8 bytes of generated initialization vector are stored back into the initialization vector area before the GCM operation is performed. This allows the generated initialization vector to be part of the additional authentication data, if desired.

A secure key may not be used for Processing Rules CBC-CS, CTR, GCM, or GCMIVGEN, or for encryption mode CHACHA20.

PKCS #11 Secret Key Reencrypt (CSFPSKR and CSFPSKR6)

Use the PKCS #11 Secret Key Reencrypt callable service to decrypt data and then reencrypt the data using secure secret keys. The interim clear text created by the decrypt process is never available to the application and never exists outside of the EP11 coprocessor. AES and DES3 secure keys are supported. CBC, CBC-PAD, and ECB modes are supported.

Both key handles must be handles of a PKCS #11 secure secret key object. The CKA_DECRYPT attribute must be true for the decrypt key. The CKA_ENCRYPT attribute must be true for the encrypt key.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the *encrypted_text_length* parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSKR6.

Format

```
CALL CSFPSKR(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
```

```
            decrypt_handle,
            encrypt_handle,
            decrypt_initialization_vector_length,
            decrypt_initialization_vector,
            encrypt_initialization_vector_length,
            encrypt_initialization_vector,
            chain_data_length,
            chain_data,
            decrypt_text_length,
            decrypt_text,
            decrypt_text_id,
            encrypt_text_length,
            encrypt_text,
            encrypt_text_id )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 2.

rule_array

Direction	Type
Input	String

Keywords that apply to the decryption of the *decrypt_text* and the encryption of the interim clear text.

Table 586. Rule Array Keywords for rule_array	
Keyword	**Meaning**
Decrypt Processing rule (one required).	
D-CBC	Performs cipher block chaining on the *decrypt_text*. The interim clear text will be a multiple of the block size for the specified encrypt mechanism (8 bytes for DES3 and 16 bytes for AES).
D-CBCPAD	Performs cipher block chaining on the *decrypt_text*. The interim clear text will be checked for correct PKCS #7 padding, which will be removed prior to reencryption.
D-ECB	Performs electronic code book encryption. The *decrypt_text_length* must be a multiple of the block size for the specified decrypt mechanism (8 bytes for DES3 and 16 bytes for AES).
Encrypt Processing rule (one required).	
E-CBC	Performs cipher block chaining on the interim clear text. The interim clear text length must be a multiple of the block size for the specified encrypt mechanism (8 bytes for DES3 and 16 bytes for AES).
E-CBCPAD	Performs cipher block chaining on the interim clear text. The interim clear text length may be shorter than the block size for the encrypt mechanism or may even be zero. PKCS #7 padding is performed and the *encrypt_text_length* will be greater than the interim clear text length.
E-ECB	Performs electronic code book encryption. The interim clear text length must be a multiple of the block size for the specified encrypt mechanism (8 bytes for DES3 and 16 bytes for AES).

decrypt_handle

Direction	**Type**
Input	String

The 44-byte handle of the secure secret key object used to decrypt the *decrypt_text*.

encrypt_handle

Direction	**Type**
Input	String

The 44-byte handle of the secure secret key object used to encrypt the interim clear text.

decrypt_initialization_vector_length

Direction	**Type**
Input	Integer

Length of the *decrypt_initialization_vector* in bytes. For CBC and CBC-PAD, this must be 8 bytes for DES and 16 bytes for AES. For ECB, this must be zero.

decrypt_initialization_vector

Direction	**Type**
Input	String

This is the 8 byte or 16 byte initial chaining value used for the decryption of the *decrypt_text*.

encrypt_initialization_vector_length

Direction	Type
Input	Integer

Length of the *encrypt_initialization_vector* in bytes. For CBC and CBC-PAD, this must be 8 bytes for DES and 16 bytes for AES. For ECB, this must be zero.

encrypt_initialization_vector

Direction	Type
Input	String

This is the 8 byte or 16 byte initial chaining value used for the encryption of the interim clear text.

chain_data_length

Direction	Type
Input	Integer

This value must be zero.

chain_data

Direction	Type
Input	String

This parameter is ignored when *chain_data_length* is zero.

decrypt_text_length

Direction	Type
Input	Integer

The length of the *decrypt_text* parameter in bytes. The length can be up to 10600.

decrypt_text

Direction	Type
Input	String

Text to be decrypted and then encrypted.

decrypt_text_id

Direction	Type
Input	Integer

The ALET identifying the space where the *decrypt_text* resides.

encrypt_text_length

Direction	Type
Input/Output	Integer

On input, the length in bytes of the *encrypt_text* parameter. On output, the length of the text reencrypted into the encrypt text parameter.

encrypt_text

Direction	Type
Output	String

The encrypted text resulting from the decryption and reencryption of the *decrypt_text*.

encrypt_text_id

Direction	Type
Input	Integer

The ALET identifying the space where the *encrypt_text* resides.

Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

Usage Notes

The use of this service keys requires an active EP11 Coprocessor. If there is not an EP11 Coprocessor online that supports Secret Key Reencrypt, the service will return with return code C reason code 2B34 (11060).

PKCS #11 Token Record Create (CSFPTRC and CSFPTRC6)

Use the Token Record Create callable service (CSFPTRC) to do these tasks:

- Initialize or re-initialize a z/OS PKCS #11 token.
- Create or copy a token object in the token data set.
- Create or copy a session object for the current PKCS #11 session.
- Create a raw object.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPTRC6.

Format

```
CALL CSFPTRC(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               handle,
               rule_array_count,
               rule_array,
               attribute_list_length,
               attribute_list)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

handle

Direction	Type
Input/Output	String

For rule TOKEN, this is the 44-byte handle of the z/OS PKCS #11 token to be initialized or reinitialized. On input, only the first 32 bytes of the handle are meaningful and it is recommended that you pad this with 12 blanks on the right. On output, this is the 44-byte handle of the z/OS PKCS #11 token created.

For rule OBJECT,

- On input,

  When creating an object (rule COPY is not specified):
  This is the 44-byte handle of a preexisting token or the omnipresent token (32 bytes of meaningful data left justified in a 44-byte field padded to the right with blanks).

  When copying from an existing object (rule COPY is specified):
  This is the 44-byte handle of the object being copied.

- On output, this is the 44-byte handle of the z/OS PKCS #11 object created.

For rule RAWOBJ, this is a buffer with a 4-byte length field, left-justified, containing the length of the buffer including the length of the length field itself. This buffer must be at least 44 bytes on input. On output, the buffer is updated with the raw object. The length field is updated with the actual number of bytes used (including its own length).

See “Handles” on page 111 for the format of a *handle*.

rule_array_count

Direction	Type
Input	Integer

The number of keywords supplied in the *rule_arrray* parameter. The value must be 1 or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justisfied in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 587. Token record create keywords

Keyword	Meaning
One of these three keywords must be specified:	
TOKEN	Specifies that a token is to be initialized. If the token exists in the token data set, the RECREATE keyword must be specified.
OBJECT	Specifies that an object (token object or session object) is to be created. If the object is to be a copy of an existing object, the COPY keyword must be specified.
RAWOBJ	Specifies that a raw object is to be created and returned in the buffer provided by the *handle* parameter.
This keyword is optional, and valid only with TOKEN:	
RECREATE	Specifies that the token exists and is to be re-initialized. All objects of the existing token will be deleted.
This keyword is optional, and valid only with OBJECT:	
COPY	Specifies that the object specified by the handle is to be copied into a new object.

attribute_list_length

Direction	Type
Input	Integer

Length of the *attribute_list* parameter in bytes.

The minimum value for this field is 2 and the maximum value for this field is 32752.

attribute_list

Direction	Type
Input	String

List of token or object attributes.

When creating or re-creating a token, the *attribute_list* parameter has this format:

Bytes	Description
0 - 31	Manufacturer ID
32 - 47	Model
48 - 63	Serial number
64 - 67	Reserved for IBM's use. Must be hexadecimal zeros.

Note: The strings supplied for Manufacturer ID, Model, and Serial number are assumed to be from code page IBM1047.

For objects, see “Attribute list” on page 111 for the format of an *attribute_list*.

Authorization

Note: Session and token objects require the same SAF authority.

Table 588. Authorization requirements for the token record create callable service

Action	Source object (Copy only)	Token / Object being created	PKCS #11 role Authority required
Create or recreate token	N/A	Token	SO (UPDATE)
Create object	N/A	Public object, except a CA certificate	USER (UPDATE) or SO (READ)
Create object	N/A	Private object, except a CA certificate	USER (UPDATE) or SO (CONTROL)
Create object	N/A	Public CA certificate object	USER (CONTROL) or SO (READ)
Create object	N/A	Private CA certificate object	USER (CONTROL) or SO (CONTROL)
Copy object	Public object, except a CA certificate	Public object, except a CA certificate	USER (UPDATE) or SO (READ)
Copy object	Public object or private object, except a CA certificate	Private object, except a CA certificate	USER (UPDATE) or SO (CONTROL)
Copy object	Private object, except a CA certificate	Public object, except a CA certificate	USER (UPDATE)
Copy object	Public object, where source or target or both are CA certificate objects	Public object, where source or target or both are CA certificate objects	USER (CONTROL) or SO (READ)
Copy object	Public object or private object, where source or target or both are CA certificate objects	Private object, where source or target or both are CA certificate objects	USER (CONTROL) or SO (CONTROL) or both USER (UPDATE) and SO (READ)
Copy object	Private object, where source or target or both are CA certificate objects	Public object, where source or target or both are CA certificate objects	USER (CONTROL) or both USER (UPDATE) and SO (READ)

Note:

- Session and token objects require the same authority.
- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles and on how ICSF determines that a certificate is a CA certificate.

Usage Notes

When creating an object, these attribute processing rules will be in effect:

- All attributes will be processed and the value of the last instance of an attribute in the template will be saved.

When copying an object, these attribute processing rules will be in effect:

- All attributes will be processed and the value of the last instance of an attribute in the template will be saved except for CKA_EXTRACTABLE and CKA_SENSITIVE. CKA_EXTRACTABLE will be copied from the source object and may be set to False if the value in the source object is True. CKA_SENSITIVE will be copied from the source object and may be set to True if the value in the source object is False.

When creating a raw object, these attribute processing rules are in effect:

- All attributes are processed and the value of the last instance of an attribute in the template is saved.
- CKA_CLASS must be CKO_SECRET_KEY.
- CKA_IBM_SECURE must be CK_FALSE.
- CKA_TOKEN must be CK_FALSE.

PKCS #11 Token Record Delete (CSFPTRD and CSFPTRD6)

Use the Token Record Delete callable service (CSFPTRD) to delete a z/OS PKCS #11 token, token object, session object, or state object. When a token is deleted, all associated objects are deleted as well. The deletions occur in the token data set (TKDS), and all session memory areas in the ICSF address space.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPTRD6.

Format

```
CALL CSFPTRD(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               handle,
               rule_array_count,
               rule_array)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	Integer

This field is ignored.

handle

Direction	Type
Input	String

44-byte name of the token or object to be deleted. See “Handles” on page 111 for the format of a *handle*.

rule_array_count

Direction	Type
Input	Integer

The number of keywords supplied in the *rule_array* parameter. This value must be 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justisfied in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 589. Token record delete keywords

Keyword	Meaning
One of these two keywords must be specified:	
TOKEN	Specifies that a token and all associated objects are to be deleted.
OBJECT	Specifies that an object is to be deleted.

Authorization

Table 590. Authorization requirements for the token record delete callable service

Token / Object Type	PKCS #11 Role Authority Required
Token	SO (UPDATE)
Public object, except CA certificate	USER (UPDATE) or SO (READ)
Private object, except CA certificate	USER (UPDATE) or SO (CONTROL)
Public CA certificate object	USER (CONTROL) or SO (READ)
Private CA certificate object	USER (CONTROL) or SO (CONTROL)
State object	None

Note:

- Session and token objects require the same authority.

- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles and how ICSF determines that a certificate is a CA certificate.

Usage Notes

An application can free state objects allocated by certain PKCS #11 callable services by calling this service. To do so, specify the handle of the state object in the *handle* parameter and "OBJECT " in the *rule_array* parameter. For more information on the PKCS #11 callable services that can allocate state objects, refer to:

- "PKCS #11 Secret Key Decrypt (CSFPSKD and CSFPSKD6)" on page 1391 CSFPSKD
- "PKCS #11 Secret Key Encrypt (CSFPSKE and CSFPSKE6)" on page 1396 CSFPSKE
- "PKCS #11 One-Way Hash, Sign, or Verify (CSFPOWH and CSFPOWH6)" on page 1372 CSFPOWH
- "PKCS #11 Generate Keyed MAC (CSFPHMG and CSFPHMG6)" on page 1364 CSFPHMG
- "PKCS #11 Verify Keyed MAC (CSFPHMV and CSFPHMV6)" on page 1368 CSFPHMV

PKCS #11 Token Record List (CSFPTRL and CSFPTRL6)

Use the Token Record List callable service (CSFPTRL) to:

- Obtain a list of z/OS PKCS #11 tokens. The caller must have SAF authority to the token for a particular token to be listed.
- Obtain a list of token and session objects for a token. Use a search template to restrict the search for specific attributes. The caller must have SAF authority to the token.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPTRL6.

Format

```
CALL CSFPTRL(
               return_code,
               reason_code,
               exit_data_length,
               exit_data,
               handle,
               rule_array_count,
               rule_array,
               search_template_length,
               search_template,
               list_length,
               handle_count,
               output_list)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

handle

Direction	Type
Input	String

For tokens, an empty string (blanks) for the first call, or the 44-byte handle of the last token found for subsequent calls.

For objects, the 44-byte handle of the token for the first call, or the 44-byte handle of the last object found for subsequent calls.

See Usage Notes for more information. See “Handles” on page 111 for the format of a *handle*.

rule_array_count

Direction	Type
Input	Integer

The number of keywords supplied in the *rule_array* parameter. This value must be 1 or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Each keyword is left-justisfied in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 591. Token record list keywords

Keyword	Meaning
Processing entity (required)	
TOKEN	Specifies that the list will contain all tokens to which the caller has SAF access. The *search_template* parameter is ignored.
OBJECT	Specifies that the list will contain the handles of all objects that match the attributes specified in the *search_template* parameter and to which the caller has SAF access.
List options (optional, valid only with OBJECT)	

Table 591. Token record list keywords (continued)	
Keyword	**Meaning**
ALL	Specifies that when listing objects, both public and private objects that meet the search criteria should be listed if the caller has SAF authority for the token. There may be no sensitive attributes in the search template. See the Authorization topic for details.
RTL	Specifies that when listing objects, the search should start with objects having the highest alphanumeric sequence numbers. In general, these would be the most recently created objects. The default is to start the search with the lowest alphanumeric sequence numbers (generally, the least recently created objects).

search_template_length

Direction	**Type**
Input	Integer

The length of the *search_template* parameter in bytes. The value must be 0 when the TOKEN keyword is specified.

The maximum size in bytes is 32752.

search_template

Direction	**Type**
Input	String

A list of criteria (attribute values) that an object must meet to be added to the list. If the *search_template_length* parameter is 0, no criteria are checked.

See “Attribute list” on page 111 for the format of an *attribute_list*.

list_length

Direction	**Type**
Input/Output	Integer

On input, the length in bytes of the *output_list* parameter. On output, the number of bytes used for the *output_list* parameter. If the supplied length is insufficient to hold one record, the *list_length* parameter is set to the minimum length required for a record.

handle_count

Direction	**Type**
Input/Output	Integer

On input, the maximum number of tokens or object handles to return in the list. On output from a successful call (return_code < 8), the actual number of tokens or object handles in the list.

output_list

Direction	**Type**
Output	String

A list of token names and descriptions or a list of object handles meeting the search criteria.

Authorization

To list tokens, the caller must have at least USER (READ) or SO (READ) authority.

Authority to list objects depends on the object's attributes and the search criteria as follows:

- To list secret key or private key objects where sensitive key attributes are specified in the search template, this must be true:
  - The object must be marked CKA_SENSITIVE=F and CKA_EXTRACTABLE=T and
  - The caller must have USER (READ) authority
- Otherwise (no sensitive attributes in the search criteria)
  - To list public objects, the caller must have at least USER (READ) or SO (READ) authority
  - To list private objects when the ALL rule array keyword is specified, the caller must have at least USER (READ) or SO (READ) authority
  - To list private objects when the ALL rule array keyword is not specified, the caller must have USER (READ) or SO (CONTROL) authority

Token / Object Type	Sensitive Attributes in search criteria	ALL Rule Specified	PKCS #11 Role Authority Required
Token	N/A	N/A	USER (READ) or SO (READ)
Public object	No	N/A	USER (READ) or SO (READ)
Private object	No	No	USER (READ) or SO (CONTROL)
Private object	No	Yes	USER (READ) or SO (READ)
Secret key or Private key object (public or private object class) CKA_SENSITIVE=F and CKA_EXTRACTABLE=T	Yes	N/A	USER (READ)
Secret key or Private key object (public or private object class) CKA_SENSITIVE=T or CKA_EXTRACTABLE=F	Yes	N/A	N/A (object is not listed)

Note:

- Session and token objects require the same authority.
- When the caller does not possess sufficient authority to list a given token or object, that record is skipped. (No information for the token or object is returned.) Processing continues with the next token or object.
- The sensitive attributes are as follows:
  - CKA_VALUE for a secret key object, Elliptic Curve private key, DSA private key, or Diffie-Hellman private key object.
  - CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, and CKA_COEFFICIENT for an RSA private key object.
- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and USER PKCS #11 roles.

Usage Notes

For tokens: On the initial call to get a list of tokens, the *handle* parameter should be all blanks. On subsequent calls, the *handle* parameter should be the last token handle from the *output_list* returned in the previous call.

The output records are in this format:

Bytes	Description
0 - 31	Token name
32 - 63	Manufacturer ID
64 - 79	Model
80 - 95	Serial number
96 - 103	Date that the token information or any token object was last updated, expressed as Coordinated Universal Time (UCT) in the format *yyyymmdd*
104 - 111	Time that the token information or any token object was last updated, expressed as Coordinated Universal Time (UCT) in the format *hhmmssth*
112 - 115	Flags **Bit** **Meaning when set on** **0** Token is write protected.

For objects: On the initial call to get a list of object handles matching the search template, the *handle* parameter contains the token handle. On subsequent calls, the *handle* parameter should contain the last object handle from the *output_list* returned in the previous call. The output records are the 44-byte handles of the objects.

PKCS #11 Unwrap Key (CSFPUWK and CSFPUWK6)

Use the Unwrap Key callable service to unwrap and create a key object using another key. The following formatting is supported:

- PKCS 1.2 formatting is supported for a secret wrapped by an RSA public key.
  - A new secret key object is created with the decrypted key value.
  - The unwrapping key must be an RSA private key object.
  - The CKA_UNWRAP attribute must be true.
- PKCS 8 formatting (CBC mode with padding and GCM mode) is supported for a private or secret key wrapped by a secret key.
  - A new private key or secret key object is created with the decrypted key values.
  - For CBC mode, the wrapping key must be a DES, DES2, DES3, or AES secret key object. For GCM mode, the wrapping key must be an AES secret key object.
  - The CKA_UNWRAP attribute must be true.
  - The encryption mechanism must be specified in the rule array and must match the key type of the unwrapping secret key object.
- IBM Proprietary Attribute Bound format – where the key's usage flags are restored from the wrapped key data.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPUWK6.

Format

```
CALL CSFPUWK(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              wrapped_key_length,
              wrapped_key,
              initialization_vector_length,
              initialization_vector,
              unwrapping_key_handle,
              attribute_list_length,
              attribute_list,
              target_key_handle )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1 or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service.

Table 592. Keywords for unwrap key

Keyword	Meaning
Formatting method (Required)	
ATTRBND	The *wrapped_key* is an IBM proprietary format. The private or secret key and its usage flags are to be unwrapped together. A signature verification public key or secret key handle must be supplied through the *initialization_vector* parameter.
PKCS-1.2	For unwrapping a secret key with a private key. RSA PKCS #1 block type 02 will be used to recover the secret key value for unwrapping with an RSA private key.
PKCS-8	For unwrapping a private or secret key with another secret key. When unwrapping a private key, the recovered private key values are expected to be DER encoded as specified by PKCS-8. The encryption mechanism rule array keyword must be specified.
Encryption mechanism (Required when PKCS-8 is specified. Otherwise, it is ignored.)	
AES	For PKCS-8 processing, AES CBC mode decryption with PKCS #7 padding removal is used to recover the raw key value. The unwrapping key must be an AES secret key object.
DES	For PKCS-8 processing, DES CBC mode decryption with PKCS #7 padding removal is used to recover the raw key value. The unwrapping key must be a DES secret key object.
DES3	For PKCS-8 processing, DES3 CBC mode decryption with PKCS #7 padding removal is used to recover the raw key value. The unwrapping key must be a DES2 or DES3 secret key object.
GCM	For PKCS-8 processing, Galois/Counter mode decryption is used to recover the raw key value. The unwrapping key must be an AES secret key object. Not supported for secure keys.

wrapped_key_length

Direction	Type
Input	Integer

Length of the wrapped key in the *wrapped_key* parameter. For GCM, this includes the length of the authentication tag.

wrapped_key

Direction	Type
Input	String

The key to be unwrapped. For GCM, this includes the authentication tag appended to the wrapped key.

initialization_vector_length

Direction	Type
Input	Integer

The length of the *initialization_vector* parameter. The initial value can only be used with PKCS-8 (CBC mode and GCM mode) or ATTRBND. This parameter is ignored for PKCS-1.2. For PKCS-8 (CBC mode), the length must match the key type of the wrapping key (8 for DES, DES2, DES3, and 16 for AES). If the length is zero, the *initialization_vector* parameter is ignored and an initial value of zero is used. For ATTRBND, the length must be 44. For GCM mode, this must be the size of the *initialization_vector* parameter field for the GCM mechanism (28 bytes).

initialization_vector

Direction	Type
Input	String

For formatting method PKCS-8 (CBC mode), this is the initial chaining value for symmetric encryption. The length must match the key type of the wrapping key.

For formatting method ATTRBND, this is the 44-byte handle of the public or secret key object to be used to verify the signature on the key data.

For formatting method PKCS-1.2, this parameter is ignored.

For formatting method PKCS-8 (GCM mode), the format is shown in Table 593 on page 1420:

Table 593. initialization_vector parameter format for GCM mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Length in bytes of the GCM initialization vector. The minimum value is 1. The maximum value is 128.
4	8	Input	64-bit address of the GCM initialization vector. It is the same value as GCM initialization vector used for the corresponding PKCS #11 Wrap Key request.
12	4	Input	Length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.

Table 593. initialization_vector parameter format for GCM mechanism (continued)

Offset	Length in bytes	Direction	Description
16	8	Input	64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

unwrapping_key_handle

Direction	Type
Input	String

The 44-byte handle of the private key or secret key object to unwrap the key. See "Handles" on page 111 for the format of a *unwrapping_key_handle*.

attribute_list_length

Direction	Type
Input	Integer

Length of the *attribute_list* parameter in bytes. The minimum value for this field is 2 and the maximum value for this field is 32752.

attribute_list

Direction	Type
Input	String

List of token or object attributes for the target key. The attributes must be consistent with the class of the object. See "Attribute list" on page 111 for the format of an *attribute_list*.

target_key_handle

Direction	Type
Output	String

The 44-byte handle of the secret key or private key object created for the unwrapped key. The object will use to token name of the unwrapping key object.

Authorization

There are two or three keys involved in this service: the unwrapping key and the target key (the new key created from the wrapped key), and (optionally) a signature verification key.

- To use an unwrapping or verification key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use an unwrapping or verification key that is a private object, the caller must have USER (READ) authority (user access).
- To unwrap a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority
- To unwrap a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority

Usage Notes

For Attribute Bound unwrapping:

- All keys involved (target, unwrapping, and verification) must have the CKA_IBM_ATTRBOUND attribute set TRUE.
- The unwrap template is restricted to the following attributes:
  - CKA_TOKEN
  - CKA_LABEL
  - CKA_SUBJECT – For private keys only
  - CKA_ID
  - CKA_START_DATE
  - CKA_END_DATE
  - CKA_APPLICATION
  - CKA_IBM_FIPS140
  - CKA_PRIVATE

A secure key may not be used for rules GCM and GCMIVGEN.

PKCS #11 Wrap Key (CSFPWPK and CSFPWPK6)

Use Wrap Key callable service to wrap a key with another key. The following formatting is supported:

- PKCS 1.2 is supported for wrapping a secret key with an RSA public key.
  - The wrapping key must be an RSA public key object.
  - The CKA_WRAP attribute must be true.
- PKCS 8 formatting (CBC mode with padding and GCM mode) is supported for wrapping a private or secret key with a secret key.
  - For CBC mode, the wrapping key must be a DES, DES2, DES3, or AES secret key object. For GCM mode, the wrapping key must be an AES secret key object.
  - The CKA_WRAP attribute must be true
  - The encryption mechanism must be specified in the rule array and must match the key type of the wrapping secret key object.
- IBM Proprietary Attribute Bound format – where the key's usage flags are to be included in the wrapped key data.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the *wrapped_key_length* parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPWPK6.

Format

```
CALL CSFPWPK(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              source_key_handle,
              wrapping_key_handle,
              initialization_vector_length,
              initialization_vector,
              wrapped_key_length,
              wrapped_key )
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1 or 2.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service.

Table 594. Keywords for wrap key

Keyword	Meaning
Formatting method (Required)	
ATTRBND	The private or secret key and its usage flags are to be wrapped together in an IBM proprietary format. A signing key handle must be supplied through the *initialization_vector* parameter.
PKCS-1.2	For wrapping a secret key with a public key. RSA PKCS #1 block type 02 will be used to format the secret key value for wrapping with an RSA public key.
PKCS-8	For wrapping a private or secret key with another secret key. When wrapping a private key, the private key values are DER encoded as specified by PKCS-8. When wrapping a secret key, the raw key value may or may not be padded according to the encryption mechanism. The encryption mechanism rule array keyword must be specified.
Encryption mechanism (Required when PKCS-8 is specified. Otherwise, it is ignored.)	
AES	For PKCS-8 processing, PKCS #7 padding is applied to the encoded or raw key value. AES CBC mode encryption is used. The wrapping key must be an AES secret key object.
DES	For PKCS-8 processing, PKCS #7 padding is applied to the encoded or raw key value. DES CBC mode encryption is used. The wrapping key must be a DES secret key object.
DES3	For PKCS-8 processing, PKCS #7 padding is applied to the encoded or raw key value. DES3 CBC mode encryption is used. The wrapping key must be a DES2 or DES3 secret key object.
GCM	For PKCS-8 processing, Galois/Counter mode encryption is performed on the raw or encoded key value. The authentication tag is returned appended to the encrypted key material. The wrapping key must be an AES secret key object. Not supported for secure keys.
GCMIVGEN	For PKCS-8 processing, Galois/Counter mode encryption is performed on the raw or encoded key value. The authentication tag is returned appended to the encrypted key material. ICSF generates part of the initialization vector and returns it in the *initialization_vector* parameter. Having ICSF generate the initialization vector ensures that initialization vectors are never repeated for a given key object. The wrapping key must be an AES secret key object. Not supported for secure keys.

source_key_handle

Direction	Type
Input	String

The 44-byte handle of the secret key or private key object to be wrapped.

wrapping_key_handle

Direction	Type
Input	String

The 44-byte handle of the public key or secret key object to wrap the secret key. See "Handles" on page 111 for the format of a *wrapping_key_handle*.

Initialization_vector_length

Direction	Type
Input	Integer

The length of the *initialization_vector* parameter. The initial value can only be used with PKCS-8 (CBC mode and GCM mode) or ATTRBND. This parameter is ignored for PKCS-1.2. For PKCS-8 (CBC mode), the length must match the key type of the wrapping key (8 for DES, DES2, DES3, and 16 for AES). If the length is zero, the initialization vector parameter is ignored and a value of zero is used. For ATTRBND, the length must be 44. For GCM mode, this must be the size of the *initialization_vector* parameter field for the GCM or GCMIVGEN mechanism (28 bytes).

Initialization_vector

Direction	Type
Input	String

For formatting method PKCS-8 (CBC mode), this is the initial chaining value for symmetric encryption. The length must match the key type of the wrapping key.

For formatting method ATTRBND, this is the 44-byte handle of the private or secret key object to be used to sign the key data.

For formatting method PKCS-1.2, this parameter is ignored.

For formatting method PKCS-8 (GCM mode), the format is shown in Table 595 on page 1425 and Table 596 on page 1426:

Table 595. initialization_vector parameter format for GCM mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Length in bytes of the GCM initialization vector area. The minimum value is 1. The maximum value is 128. The recommended length is 12.
4	8	Input	64-bit address of the GCM initialization vector area.
12	4	Input	Length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.

Table 595. initialization_vector parameter format for GCM mechanism (continued)

Offset	Length in bytes	Direction	Description
16	8	Input	64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

Table 596. initialization_vector parameter format for GCMIVGEN mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Nonce value which ICSF uses as the first 4 bytes of the GCM initialization vector. The remaining 8 bytes are generated and returned to the caller in the initialization vector area.
4	8	Input	64-bit address of the GCM initialization vector area into which ICSF stores the 8 bytes it generates. The area must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. The complete 12 byte GCM initialization vector to be used for decryption is the 4-byte nonce concatenated with the 8 bytes stored in the area.

Table 596. initialization_vector parameter format for GCMIVGEN mechanism (continued)

Offset	Length in bytes	Direction	Description
12	4	Input	Length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.
16	8	Input	64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. This value must be one of 12, 13, 14, 15, or 16.

wrapped_key_length

Direction	Type
Input/Output	Integer

On input, the length of the *wrapped_key* parameter. On output, the actual length of the wrapped key returned in the *wrapped_key* parameter.

For rules GCM and GCMIVGEN, the returned length includes the length of the appended authentication tag.

wrapped_key

Direction	Type
Output	String

The wrapped key. For rules GCM and GCMIVGEN, the authentication tag is appended to the end of the wrapped key.

Authorization

There are two or three key objects used by this service, the source key (the key to be wrapped), the wrapping key, and (optionally) a signing key.

- To wrap a source key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To wrap a source key that is a private object, the caller must have USER (READ) authority (user access)

- To use a wrapping or signing key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a wrapping or signing key that is a private object, the caller must have USER (READ) authority (user access).

Usage Notes

Clear keys may not be used to wrap secure keys and vice versa. (See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for the more information on clear keys and secure keys.)

- One exception, clear RSA public keys may be used to perform non-attribute bound wrap of secure secret keys.

For Attribute Bound wrapping, all keys involved (source, wrapping, and signing) must have the CKA_IBM_ATTRBOUND attribute set TRUE.

For processing rule GCMIVGEN, the total number of initialization vector generations for a token key object is limited to 4294967295. Once this number is exceeded, the key object is no longer eligible for processing rule GCMIVGEN and is considered 'retired'. This usage counter is maintained in the TKDS as part of the key object. For keys that are copied using CSFPTRC (C_CopyObject), the existing counter value is copied to the new key object, but not synchronized after that.

For processing rule GCMIVGEN, session key objects have no maximum lifetime. They may be retired at any time. Once retired, the key object is no longer eligible for processing rule GCMIVGEN.

For processing rule GCMIVGEN, the nonce value portion of the initialization vector is predetermined by the caller. It is used to ensure that initialization vector values are not repeated for any given key value. The caller should provide a random value and change the value as often as practical. It must be changed whenever:

- A given key value is replicated as a new persistent key object.
- A given persistent key object is replicated as a new session key object.
- A given session key value is re-instantiated after system IPL.
- A given key value is re-instantiated after ICSF indicates it has been retired.

Use of processing rule GCMIVGEN with token key objects requires that the first 4 bytes of ECVTSPLX or CVTSNAME be set to a unique value with respect to other systems. For information on how to set these fields, see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

A session key object should never be used for processing rule GCMIVGEN if the key value is distributed to multiple systems outside the current sysplex where new initialization vectors may be generated. Use only token key objects in such cases. If session key objects are used, the other systems must use different nonces.

For processing rule GCMIVGEN, the 8 bytes of generated initialization vector are stored back into the initialization vector area before the GCM operation is performed. This allows the generated initialization vector to be part of the additional authentication data, if desired.

A secure key may not be used for rules GCM and GCMIVGEN.

Chapter 18. Using the PKCS #11 key structure callable services

This topic describes the PKCS #11 key structure callable services which offer a fast-path alternative to some of the traditional PKCS #11 callable services. Under the standard PKCS #11 callable services, a key must first be established as an object before it may be used for a cryptographic operation. When that key is no longer needed, that key should be deleted to prevent it from consuming resources unnecessarily.

The PKCS #11 key structure callable services allow you to specify the key structure (rather than the key handle) as an in-memory parameter. This eliminates the need to create and delete the key object.

The PKCS #11 key structure callable services support the ASN.1 clear key structures:

The PKCS #8 PrivateKeyInfo structure

This key structure is defined in section 5 of RFC 5208 *Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2*.

```
Private-Key Information Syntax

This section gives the syntax for private-key information.

Private-key information shall have ASN.1 type PrivateKeyInfo:
   PrivateKeyInfo ::= SEQUENCE {
     version                   Version,
     privateKeyAlgorithm       PrivateKeyAlgorithmIdentifier,
     privateKey                PrivateKey,
     attributes           [0]  IMPLICIT Attributes OPTIONAL }
   Version ::= INTEGER
   PrivateKeyAlgorithmIdentifier ::= AlgorithmIdentifier
   PrivateKey ::= OCTET STRING
   Attributes ::= SET OF Attribute
```

The SubjectPublicKeyInfo structure

This key structure is defined in section 4.1 of RFC 3280 *Internet X.509 Public Key Infrastructure*.

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
   algorithm            AlgorithmIdentifier,
   subjectPublicKey     BIT STRING  }
```

See each PKCS #11 key structure callable service for information on which key structure is supported:

- “PKCS #11 Private Key Structure Decrypt (CSFPPD2 and CSFPPD26)” on page 1429
- “PKCS #11 Private Key Structure Sign (CSFPPS2 and CSFPPS26)” on page 1432
- “PKCS #11 Public Key Structure Encrypt (CSFPPE2 and CSFPPE26)” on page 1434
- “PKCS #11 Public Key Structure Verify (CSFPPV2 and CSFPPV26)” on page 1437

PKCS #11 Private Key Structure Decrypt (CSFPPD2 and CSFPPD26)

Use the PKCS #11 private key structure decrypt callable service to decrypt data using a clear RSA private key. PKCS #1 v1.5 formatting is used. The key must be a DER encoded PrivateKeyInfo structure as specified by PKCS #8.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPD26.

Format

```
CALL CSFPPD2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             cipher_value_length,
             cipher_value,
             reserved,
             clear_value_length,
             clear_value,
             private_key_info_length,
             private_key_info)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Table 597 on page 1431 lists the keywords. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 597. Keywords for Private Key Structure Decrypt

Keyword	Meaning
Mechanism (The following must be specified)	
RSA-PKCS	Mechanism is RSA decryption using PKCS #1 v1.5 formatting.

cipher_value_length

Direction	Type
Input	Integer

The length of the *cipher_value* parameter in bytes. Must be the size of the modulus of the key.

cipher_value

Direction	Type
Input	String

The value to be decrypted.

reserved

Direction	Type
Ignored	String

This field is currently not used.

clear_value_length

Direction	Type
Input/Output	Integer

The length in bytes of the *clear_value* parameter. On input, this must be at least the size of the RSA modulus in bytes. On output, this is updated to be the actual length of the decrypted value.

clear_value

Direction	Type
Output	String

This field contains the decrypted value.

private_key_info_length

Direction	Type
Input	Integer

Length in bytes of the *private_key_info* parameter. The maximum size you can specify is 3000 bytes.

private_key_info

Direction	Type
Input	String

The DER encoded PrivateKeyInfo structure as specified by PKCS #8. The *privateKeyAlgorithm* field of this structure must indicate that the key is an RSA key.

Authorization

The CSFSERV resource name that protects this service is CSFPKD, and it is the same resource name that is used to protect the PKA Decrypt service.

Usage notes

- This service always enforces FIPS restrictions.
- This service requires an IBM eServer zSeries 990 or later machine type.
- The *attributes* field in the key structure is ignored.

PKCS #11 Private Key Structure Sign (CSFPPS2 and CSFPPS26)

Use the PKCS #11 private key structure sign callable service to sign data using a clear RSA private key. PKCS #1 v1.5 formatting is used. The key must be a DER encoded PrivateKeyInfo structure as specified by PKCS #8.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPS26.

Format

```
CALL CSFPPS2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             hash_length,
             hash,
             reserved,
             signature_length,
             signature,
             private_key_info_length,
             private_key_info)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and cryptographic coprocessor return/reason codes," on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Table 598 on page 1433 lists the keywords. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 598. Keywords for Private Key Structure Sign

Keyword	Meaning
Mechanism (The following must be specified)	
RSA-PKCS	Mechanism is RSA signature generation using PKCS #1 v1.5 formatting.

hash_length

Direction	Type
Input	Integer

The length of the *hash* parameter in bytes.

hash

Direction	Type
Input	String

The value to be signed. This is expected to be a DER encoded DigestInfo structure.

reserved

Direction	Type
Ignored	String

This field is currently not used.

signature_length

Direction	Type
Input/Output	Integer

The length in bytes of the *signature* parameter. On output, this is updated to be the actual length of the generated signature.

signature

Direction	Type
Output	String

This field contains the generated signature.

private_key_info_length

Direction	Type
Input	Integer

Length in bytes of the *private_key_info* parameter. The maximum size you can specify is 3000 bytes.

private_key_info

Direction	Type
Input	String

The DER encoded PrivateKeyInfo structure as specified by PKCS #8. The *privateKeyAlgorithm* field of this structure must indicate that the key is an RSA key.

Authorization

The CSFSERV resource name that protects this service is CSFDSG, and it is the same resource name that is used to protect the PKA Digital Signature Generate service.

Usage notes

- This service always enforces FIPS restrictions.
- This service requires an IBM eServer zSeries 990 or later machine type.
- The *attributes* field in the key structure is ignored.

PKCS #11 Public Key Structure Encrypt (CSFPPE2 and CSFPPE26)

Use the PKCS #11 public key structure encrypt callable service to encrypt data using an RSA public key. PKCS #1 v1.5 formatting is used. The key must be a DER encoded SubjectPublicKeyInfo structure as specified by RFC 3280.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPE26.

Format

```
CALL CSFPPE2(
              return_code,
              reason_code,
              exit_data_length,
              exit_data,
              rule_array_count,
              rule_array,
              clear_value_length,
              clear_value,
              reserved,
              cipher_value_length,
              cipher_value,
              subject_public_key_info_length,
              subject_public_key_info)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Table 599 on page 1436 lists the keywords. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 599. Keywords for Public Key Structure Encrypt

Keyword	Meaning
Mechanism (The following must be specified)	
RSA-PKCS	Mechanism is RSA signature encryption using PKCS #1 v1.5 formatting.

clear_value_length

Direction	Type
Input	Integer

The length in bytes of the *clear_value* parameter.

clear_value

Direction	Type
Input	String

The signature value to be encrypted.

reserved

Direction	Type
Ignored	String

This field is currently not used.

cipher_value_length

Direction	Type
Input/Output	Integer

The length of the *cipher_value* parameter in bytes. On output, this is updated to be the actual length of the encrypted value.

cipher_value

Direction	Type
Output	String

This field contains the encrypted value.

subject_public_key_info_length

Direction	Type
Input	Integer

Length in bytes of the *subject_public_key_info* parameter. The maximum size you can specify is 3000 bytes.

subject_public_key_info

Direction	Type
Input	String

The DER encoded SubjectPublicKeyInfo structure as specified by RFC 3280. The *algorithm* field of this structure must indicate that the key is an RSA key.

Authorization

The CSFSERV resource name that protects this service is CSFPKE, and it is the same resource name used to protect the PKA Encrypt service.

Usage notes

- This service always enforces FIPS restrictions.
- This service requires an IBM eServer zSeries 990 or later machine type.

PKCS #11 Public Key Structure Verify (CSFPPV2 and CSFPPV26)

Use the PKCS #11 public key structure verify callable service to verify data using an RSA public key. PKCS #1 v1.5 formatting is used. The key must be a DER encoded SubjectPublicKeyInfo structure as specified by RFC 3280.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPV26.

Format

```
CALL CSFPPV2(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             signature_length,
             signature,
             reserved,
             hash_length,
             hash,
             subject_public_key_info_length,
             subject_public_key_info)
```

Parameters

return_code

Direction	Type
Output	Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the return codes.

reason_code

Direction	Type
Output	Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and cryptographic coprocessor return/reason codes,” on page 1441 lists the reason codes.

exit_data_length

Direction	Type
Ignored	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction	Type
Ignored	String

This field is ignored.

rule_array_count

Direction	Type
Input	Integer

The number of keywords you supplied in the *rule_array* parameter. This value must be 1.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Table 600 on page 1438 lists the keywords. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 600. Keywords for Public Key Structure Verify

Keyword	Meaning
Mechanism (The following must be specified)	
RSA-PKCS	Mechanism is RSA signature verification using PKCS #1 v1.5 formatting.

signature_length

Direction	Type
Input	Integer

The length in bytes of the *signature* parameter.

signature

Direction	Type
Input	String

The signature value to be validated.

reserved

Direction	Type
Ignored	String

This field is currently not used.

hash_length

Direction	Type
Input	Integer

The length of the *hash* parameter in bytes.

hash

Direction	Type
Input	String

The value to be verified. This is expected to be a DER encoded DigestInfo structure.

subject_public_key_info_length

Direction	Type
Input	Integer

Length in bytes of the *subject_public_key_info* parameter. The maximum size you can specify is 3000 bytes.

subject_public_key_info

Direction	Type
Input	String

The DER encoded SubjectPublicKeyInfo structure as specified by RFC 3280. The *algorithm* field of this structure must indicate that the key is an RSA key.

Authorization

The CSFSERV resource name that protects this service is CSFDSV, and it is the same resource name used to protect the PKA Digital Signature Verify service.

Usage notes

- This service always enforces FIPS restrictions.
- This service requires an IBM eServer zSeries 990 or later machine type.

Appendix A. ICSF and cryptographic coprocessor return/reason codes

This topic includes this information:

- Return codes and reason codes issued on the completion of a call to an ICSF callable service.
- Return codes and reason codes issued on the completion of a process on a IBM 4767 PCIe, IBM 4765 PCIe, and IBM 4764 PCI-X Cryptographic Coprocessor.
- ICSF return and reason codes can be specified in the installation options data set on the REASONCODES parameter. If the REASONCODES option is not specified, the default of REASONCODES(ICSF) is used. A REASONCODES line in the description indicates a conversion was done as a result of the REASONCODES option in the installation options data set.

  If you specified REASONCODES(ICSF) and your service was processed on a CCA coprocessor, a cryptographic coprocessor reason code may be returned if there is no 1-1 corresponding ICSF reason code. See *CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors* for additional information.

Return codes and reason codes

This topic describes return codes and reason codes.

The Crypto Express coprocessor return and reason codes have been merged with the ICSF codes. If there is a REASONCODES line in the description, it will indicate an alternate reason code you should investigate.

Each return code returns unique reason codes to your application program. The reason codes associated with each return code are described in these topics. The reason code tables present the hexadecimal code followed by the decimal code in parenthesis.

Obtaining a dump for ICSF reason codes

Use the following slip to produce a dump for a reason code that is issued by ICSF:

```
SLIP
SET,IF,A=SYNCSVCD,RANGE=(10?+C8?+B8?+74),DATA=(13R??+B2,EQ,xxxx),
SDATA=(CSA,SQA,RGN,TRT,SUM),JOBLIST=(CSF),END
```

where *xxxx* is the lower two bytes of the reason code in hex with a leading zero or zeros.

For example, the following sample shows a SLIP command that is intended to capture diagnostics for the 7F8 reason code:

```
SLIP
SET,IF,A=SYNCSVCD,RANGE=(10?+C8?+B8?+74),DATA=(13R??+B2,EQ,07F8),
SDATA=(CSA,SQA,RGN,TRT,SUM),JOBLIST=(CSF),END
```

Notes:

- In the sample, the 7F8 reason code is prepended with a zero so that it occupies two bytes.
- In rare instances, the SLIP might not capture the requested reason code. Contact the IBM Support Center to check whether the requested reason code is the reason the SLIP did not match.

Return codes

Table 601 on page 1442 lists return codes from the ICSF callable services.

Table 601. Return Codes	
Return Code Hex (Decimal)	**Description**
Return Code 0 (0)	The call to the service was successfully processed. See the reason code for more information.
Return Code 4 (4)	The call to the service was successfully processed, but some minor event occurred during processing. See the reason code for more information. **User action**: Review the reason code.
Return Code 8 (8)	The call to the service was unsuccessful. The parameters passed into the call are unchanged, except for the return code and reason code. There are rare examples where output areas are filled, but their contents are not guaranteed to be accurate. These are described under the appropriate reason code descriptions. The reason code identifies which error was found. **User action**: Review the reason code, correct the problem, and retry the call.
Return Code C (12)	The call to the service could not be processed because ICSF was not active, ICSF found something wrong in its environment, a TSS security product is not available, or a processing error occurred in a TSS product. The parameters passed into the call are unchanged, except for the return code and reason code. **User action**: Review the reason code and take the appropriate action.
Return Code 10 (16)	The call to the service could not be processed because ICSF found something seriously wrong in its environment or a processing error occurred in the coprocessor. The parameters passed into the call are unchanged, except for the return code and reason code. **User action**: Review the reason code and contact your system programmer.
Return Code 14 (20)	The call to the service could not be processed because an unexpected error occurred in ICSF's cryptographic software element. The reason codes for this error are not documented. **User action:** Contact your IBM support center.
Return Code 18 (24)	The call to the service could not be processed because an unexpected error occurred in the Crypto Express Enterprise PKCS #11 coprocessor. The reason codes for this error are not documented. **User action:** Contact your IBM support center.
Return Code 19 (25)	The call to the service could not be processed because a vendor specific error occurred in the Crypto Express Enterprise PKCS #11 coprocessor. The reason codes for this error are not documented. **User action:** Contact your IBM support center.

Reason codes for return code 0 (0)

Table 602 on page 1442 lists reason codes returned from callable services that give return code 0.

Table 602. Reason codes for return code 0 (0)	
Reason Code Hex (Decimal)	**Description**
0 (0)	The call to the ICSF callable service was successfully processed. No error was encountered. **User action**: None.
2 (2)	The call to the ICSF callable service was successfully processed. A minor error was detected. A key used in the service did not have odd parity. This key could be one provided by you as a parameter or be a key (perhaps one of many keys) that was retrieved from the in-storage CKDS. **User action**: Refer to the reason code obtained when the key passed to this service was transformed into operational form using clear key import, multiple clear key import, key import, secure key import, or multiple secure key import callable services. Check if any of the services prepared an even parity key. If one of these services reported an even parity key, you need to know which key is affected. If none of these services identified an even parity key, then the even parity key detected was found on the CKDS. Report this to your administrator. **REASONCODES**: ICSF 4 (4)

Table 602. Reason codes for return code 0 (0) (continued)

Reason Code Hex (Decimal)	Description
4 (4)	The call to the ICSF callable service was successfully processed. A minor error was detected. A key used in the service did not have odd parity. This key could be one provided by you as a parameter or be a key (perhaps one of many keys) that was retrieved from the in-storage CKDS. **User action**: Refer to the reason code obtained when the key passed to this service was transformed into operational form using clear key import, multiple clear key import, key import, secure key import, or multiple secure key import callable services. Check if any of the services prepared an even parity key. If one of these services reported an even parity key, you need to know which key is affected. If none of these services identified an even parity key, then the even parity key detected was found on the CKDS. Report this to your administrator. **REASONCODES**:TSS 2 (2)
8 (8)	The CKDS key record read callable service attempted to read a NULL key record. The returned key token contains a null token. **User action**: None required.
862 (2146)	The call to the callable service was successfully processed. A key was wrapped by a weaker key. This reason code is returned when either the "Warn when weak wrap - Transport keys" or "Warn when weak wrap - Master keys" access control point is enabled. **User action:** None required. If you wish to prohibit weak key wrapping, enable the access control point "Prohibit weak wrapping - Transport keys" and "Prohibit weak wrapping - Master keys" access control points using the TKE workstation.
87D (2173)	The call to the callable service was successfully processed. The key token format was already payload version 1 (fixed-length).
BC2 (3010)	The call to CSFIQF was successful. Additionally, the coprocessor adapter is disabled by TKE.
D25 (3365)	KDS multi-purpose service completed and there are informational messages logged.
D5F (3423)	The key retrieved from the CKDS should not be used to encrypt data. CSNBKRR2 was called with the label of an archived key. The key is a symmetric data-encrypting key. The PROTKEY rule array keyword was specified. The CSF.KDS.KEY.ARCHIVE.DATA.DECRYPT control is enabled. **User action:** Do not use this key to encrypt data. Contact your ICSF administrator for guidance.
DAC (3500)	The Options Data Set Refresh function completed. All changes were successful.
F9D (3997)	Clear RSA CRT private key material passed to a callable service had prime p less than prime q which is in violation of the standard. The key parts have been corrected by swapping primes p and q, swapping CRT exponents dp and dq, and recalculating qInv = q^-1 mod p (called U in IBM Common Cryptographic Architecture (CCA) publications).
2710 (10000)	The call to the callable service was successfully processed. The keys in one or more key identifiers have been reenciphered from encipherment under the old master key to encipherment under the current master key. **User action**: If you obtained your operational token from a file, replace the token in the file with the token just returned from ICSF. Management of internal tokens is a user responsibility. Consider the possible case where the token for this call was fetched from a file, and where this reason code is ignored. For the next invocation of the service, the token will be fetched from the file again, and the service will give this reason code again. If this continues until the master key is changed again, then the next use of the internal token will fail.
2711 (10001)	The call to the callable service was successfully processed. The keys in one or more key identifiers were encrypted under the old master key. The callable service was unable to reencipher the key.
2713 (10003)	The call to the callable service was successfully processed. Weak key used. The strength of the KEK key is less than the strength of the key to be wrapped. If Access Control Point 'Prohibit weak wrapping - Transport keys' is not enabled, this informational Reason Code will be returned. If Access Control Point 'Prohibit weak wrapping - Transport keys' is enabled you will receive an error from the callable service. **User action:** None.
2715 (10005)	During X.509 certificate processing, the certificate revocation list was expired. **User action**: Obtain an up-to-date certificate revocation list for processing.

Table 602. Reason codes for return code 0 (0) (continued)	
Reason Code Hex (Decimal)	**Description**
2716 (10006)	During X.509 certificate processing, the KRD credential was expired. **User action**: Obtain an up-to-date KRD credential for processing.

Reason codes for return code 4 (4)

Table 603 on page 1444 lists reason codes returned from callable services that give return code 4.

Table 603. Reason codes for return code 4 (4)	
Reason Code Hex (Decimal)	**Description**
1 (1)	The verification test failed. **REASONCODES**: This reason code also corresponds to these ICSF reason codes: FA0 (4000), 1F40 (8000), 1F44 (8004), 2328 (9000), 232C (9004), 2AF8 (11000), or 36B8 (14008).
13 (19)	This is a combination reason code value. The call to the Encrypted PIN verify (PINVER) callable service was successfully processed. However, the trial PIN that was supplied does not match the PIN in the PIN block. **User action**: The PIN is incorrect. If you expected the reason code to be zero, check that you are using the correct key. **REASONCODES**: ICSF BD4 (3028)
14 (20)	The input text length was odd rather than even. The right nibble of the last byte is padded with X'00'. **User action**: None **REASONCODES**: ICSF 7D0 (2000)
A6 (166)	The control vector is not valid because of parity bits, anti-variant bits, inconsistent KEK bits, or because bits 59 to 62 are not zero.
B3 (179)	The control vector keywords that are in the rule array are ignored.
D7 (215)	PTR2AUTH usage of the authentication key is being ignored because the **Encrypted PIN Translate2 - Permit ISO-4 to ISO-4 PTR2AUTH** access control is not enabled in the domain role. **User action**: Contact your ICSF administrator to enable the access control.
1AD (429)	The digital signature verify ICSF callable service completed successfully but the supplied digital signature failed verification. **User action**: None **REASONCODES**: ICSF 2AF8 (11000)
7D0 (2000)	The input text length was odd rather than even. The right nibble of the last byte is padded with X'00'. **User action**: None **REASONCODES**: TSS 14 (20)
81E (2078)	The call to CKDS Key Record Read was successful. The key label exists in the CKDS. The key label contains a clear DES or AES key token and is not returned to the caller.
872 (2162)	A weak master key was detected when the final key part was loaded for the DES or RSA master key. A key is weak if any of the three parts are the same as another part. For example, when the first and third key parts are the same, the key is weak (effectively a double-length key). **User action**: Create new key values for the new master key and retry master key entry.
BBA (3002)	The call to the CVV Verify callable service was successfully processed. However, the trial CVV that was supplied does not match the generated CVV. In addition, a key in the key identifier has been reenciphered. **REASONCODES**: See reason code 4000 (return code 4) for more details about the incorrect CVV. See reason code 10000 (return code 0) for more details about the key reencipherment.
BC9 (3017)	The call to create a list of information completed successfully, however the storage supplied for the list was insufficient to hold the complete list.

Table 603. Reason codes for return code 4 (4) (continued)	
Reason Code Hex (Decimal)	**Description**
BD4 (3028)	The call to the Encrypted PIN verify (PINVER) callable service was successfully processed. However, the trial PIN that was supplied does not match the PIN in the PIN block. **User action**: The PIN is incorrect. If you expected the reason code to be zero, check that you are using the correct key. **REASONCODES**: TSS 13 (19)
BD8 (3032)	This is a combination reason code value. The call to the Encrypted PIN verify (PINVER) callable service was successfully processed. However, the trial PIN that was supplied does not match the PIN in the PIN block. In addition, a key in a key identifier token has been reenciphered. **REASONCODES**: See reason code 3028 (return code 4) for more detail about the incorrect PIN. See reason code 10000 (return code 0) for more detail about the key reencipherment.
BFC (3068)	The verification pattern of an encrypted CPACF key block does not match the current wrapping key's verification pattern.
CE4 (3300)	The KDS list service found no records that matched the label filter and search criteria specified.
CE5 (3301)	The key data set specified for the KDS list, KDS metadata read, and KDS metadata write services is empty or was not specified in the options data set.
CE6 (3302)	The key data set selected for the KDS list, KDS metadata read, and KDS metadata write services is not in KDSR format. Either a rule array keyword, the search criteria, or metadata specified requires the data set to be in KDSR format. **User action**: Convert the key data set to KDSR format or restrict the metadata to the type supported by your key data set format.
CE7 (3303)	The call to the KDS list service completed successfully, but the storage supplied for the list was insufficient to hold the complete list. The label count parameter contains the number of labels returned and the output list length parameter contains the number of bytes in the output area that are filled in. **User action**: Call the service again to get more entries for the list. The *continuation_area* parameter contains the information necessary to continue the search where this request left off. Pass the *continuation_area* parameter unchanged on subsequent requests.
CEB (3307)	The KDS metadata write service attempted to archive a record that is already archived.
CF4 (3316)	The PKCS #11 token handle specified for the KDS list service is for a token that does not have objects in the TKDS.
D00 (3328)	The KDS metadata write service was not able to set the archive flag because either the prohibit archive flag is enabled or was not able to set the prohibit archive flag because the archive flag is enabled.
D01 (3329)	The KDS metadata write service was supplied a date that is out of range: • The key material validity end date may not be in the past. • The key material validity start date may not be after the end date. • The last referenced date may not be in the future. **User action**: Correct the date specified and rerun the request.
D0D (3341)	The KDS metadata write service attempted to recall a record that is not archived.
D12 (3346)	The KDS metadata write service processed all the records in the label list and the processing of one or more records did not complete successfully. **User action**: Check the results list and determine which records failed.
D23 (3363)	KDS multi-purpose service completed, but there are messages logged that require attention.
D26 (3366)	KDS multi-purpose service completed and there are informational messages logged.
D41 (3393)	The ARQC could not be verified. **User action**: Ensure that the correct cryptogram information was passed, the correct key mode was specified, and the correct issuer master key was used.

Table 603. Reason codes for return code 4 (4) (continued)	
Reason Code Hex (Decimal)	**Description**
D45 (3397)	Failure to verify the data authentication code. **User action**: Ensure that the correct PAN and PAN sequence number were passed and the correct issuer master key was used.
D46 (3398)	Failure to verify the dynamic number. **User action**: Ensure that the correct ATC was passed and the correct issuer master key was used.
DAD (3501)	The Options Data Set Refresh function completed. No changes were detected.
DAF (3503)	The Options Data Set Refresh function completed. Some changes were made. **User action**: Check the ICSF joblog.
DD5 (3541)	An attempt was made to compliant-check or compliant-tag a key token. The key token already has the compliant tag. **User action**: None.
FA0 (4000)	The CVV did not verify. **User action**: Regenerate the CVV. **REASONCODES**: TSS 1 (1)
FA4 (4004)	Rewrapping is not allowed for one or more keys.
138F (5007)	The PAN presented as input to the Card Number Update2 verb was different from the PAN in the clear PBF-0 block.
1F40 (8000)	The call to the MAC verification (MACVER) callable service was successfully processed. However, the trial MAC that you supplied does not match that of the message text. **User action**: The message text may have been modified, such that its contents cannot be trusted. If you expected the reason code to be zero, check that you are using the correct key. Check that all segments of the message were presented and in the correct sequence. Also check that the trial MAC corresponds to the message being authenticated. **REASONCODES**: TSS 1 (1)
1F44 (8004)	This is a combination reason code value. The call to the MAC verification (MACVER) callable service was successfully processed. However, the trial MAC that you supplied does not match that of the message text. In addition, a key in a key identifier token has been reenciphered. **User action**: See reason code 8000 (return code 4) for more detail about the incorrect MAC. See reason code 10000 (return code 0) for more detail about the key reencipherment. **REASONCODES**: TSS 1 (1)
2328 (9000)	The call to the key test service processed successfully, but the key test pattern was not verified. **User action**: Investigate why the key failed. When determining this, you can reinstall or regenerate the key. **REASONCODES**: TSS 1 (1)
232C (9004)	This is a combination reason code value. The call to the key test service processed successfully, but the key test pattern was not verified. Also, the key token has been reenciphered. **User action**: Investigate why the key failed. When determining this, you can reinstall or regenerate the key. **REASONCODES**: TSS 1 (1)
2AF8 (11000)	The digital signature verify ICSF callable service completed, but the supplied digital signature failed verification or an input wrapped PKCS #11 object failed validation. **User action**: None required. **REASONCODES**: TSS 1AD (429)
36B8 (14008)	The PKDS record failed the authentication test. **User action**: The record has changed since ICSF wrote it to the PKDS. The user action is application dependent. **REASONCODES**: TSS 1 (1)

Table 603. Reason codes for return code 4 (4) (continued)	
Reason Code Hex (Decimal)	**Description**
8D10 (36112)	CKDS conversion completed successfully, but some tokens could not be rewrapped. For more information about the key tokens that could not be rewrapped, see the CSFM729I message or messages.

Reason codes for return code 8 (8)

Table 604 on page 1447 lists reason codes returned from callable services that give return code 8.

Most of these reason codes indicate that the call to the service was unsuccessful. No cryptographic processing took place. Therefore, no output parameters were filled. Exceptions to this are noted in the descriptions.

Table 604. Reason codes for return code 8 (8)	
Reason Code Hex (Decimal)	**Description**
00C (12)	A key identifier was passed to a service or token. It is checked in detail to ensure that it is a valid token, and that the fields within it are valid values. There is a token validation value (TVV) in the token, which is a non-cryptographic value. This value was again computed from the rest of the token, and compared to the stored TVV. If these two values are not the same, this reason code is returned. **User action**: The contents of the token have been altered because it was created by ICSF or TSS. Review your program to see how this could have been caused.
016 (22)	The ID number in the request field is not valid. The PAN data or transaction information is incorrect.
017 (23)	Offset length not correct for data to be inserted.
018 (24)	A key identifier was passed to a service. The master key verification pattern in the token shows that the key was created with a master key that is neither the current master key nor the old master key. Therefore, it cannot be reenciphered to the current master key. **User action**: Re-import the key from its importable form (if you have it in this form), or repeat the process you used to create the operational key form. If you cannot do one of these, you cannot repeat any previous cryptographic process that you performed with this token. **REASONCODES**: ICSF 2714 (10004)
019 (025)	A length parameter has an incorrect value. The value in the length parameter could have been zero (when a positive value was required) or a negative value. If the supplied value was positive, it could have been larger than your installation's defined maximum, or for MDC generation with no padding, it could have been less than 16 or not an even multiple of 8. **User action**: Check the length you specified. If necessary, check your installation's maximum length with your ICSF administrator. Correct the error.
01D (29)	A key identifier was passed to a service or token. It is checked in detail to ensure that it is a valid token, and that the fields within it are valid values. There is a token validation value (TVV) in the token, which is a non-cryptographic value. This value was again computed from the rest of the token, and compared to the stored TVV. If these two values are not the same, this reason code is returned. **User action**: The contents of the token have been altered because it was created by ICSF or TSS. Review your program to see how this could have been caused. **REASONCODES**: ICSF 2710 (10000)
01E (30)	A key label was supplied for a key identifier parameter. This label is the label of a key in the in-storage CKDS or PKDS. A key record with that label (and the specific type if required by the ICSF callable service) could not be found. For a retained key label, this error code is also returned if the key is not found in the CCA coprocessor specified in the PKDS record. **User action**: Check with your administrator if you believe that this key should be in the in-storage CKDS or the PKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label. **REASONCODES**: ICSF 271C (10012)
01F (31)	The control vector did not specify a DATA key. The key may be a CIPHER key which does not have the XPRTCPAC bit set in the control vector. **REASONCODES**: ICSF 272C (10028)

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
020 (32)	You called the CKDS key record create callable service, but the *key_label* parameter syntax was incorrect. **User action**: Correct *key_label* syntax. **REASONCODES**: ICSF 3EA0 (16032)
021 (33)	The *rule_array* parameter contents or a parameter value is not correct. **User action**: Refer to the rule_array parameter described in this publication under the appropriate callable service for the correct value. **REASONCODES**: ICSF 7E0 (2016)
022 (34)	A *rule_array* keyword combination is not valid or a keyword is specified that conflicts with another parameter. **REASONCODES**: ICSF 7E0 (2016)
023 (35)	The *rule_array_count* parameter contains a number that is not valid. **User action**: Refer to the *rule_array_count* parameter described in this publication under the appropriate callable service for the correct value. **REASONCODES**: ICSF 7DC (2012)
027 (39)	A control vector violation occurred. **REASONCODES**: This reason code also corresponds to these ICSF reason codes: 272C (10028), 2730 (10032), 2734 (10036), 2744 (10052), 2768 (10088), 278C (10124), 3E90 (16016), 2724 (10020).
028 (40)	The service code does not contain numerical data. **REASONCODES**: ICSF BE0 (3040)
029 (41)	The *key_form* parameter is neither IM nor OP. Most constants, these included, can be supplied in lowercase or uppercase. Note that this parameter is 4 bytes long, so the value IM or OP is not valid. They must be padded on the right with blanks. **User action**: Review the value provided and change it to IM or OP, as required.
02A (42)	The expiration date is not numeric (X'F0' through X'F9'). The parameter must be character representations of numerics or hexadecimal data. **User action**: Review the numeric parameters or fields required in the service that you called and change to the format and values required. **REASONCODES**: ICSF BE0 (3040)
02B (43)	The value specified for the *key_length* parameter of the key generate callable service is not valid. **User action**: Review the value provided and change it as appropriate. **REASONCODES**: See also the ICSF reason code 80C (2060) or 2710 (10000) for additional information.
02C (44)	The CKDS key record create callable service requires that the key created not already exist in the CKDS. A key of the same label was found. **User action**: Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer.
02D (45)	An input character is not in the code table. **User action**: Correct the code table or the source text.
02F (47)	A source key token is unusable because it contains data that is not valid or undefined. **REASONCODES**: This reason code also corresponds to these ICSF reason codes: 83C (2108), 2754 (10068), 2758 (10072), 275C (10076), 2AFC (11004), 2B04 (11012), 2B08 (11016), 2B10 (11024). See those reason codes for additional information.
030 (48)	One or more keys has a master key verification pattern that is not valid. This reason code also corresponds to these ICSF reason codes: 2714 (10004) and 2B0C (11020). See those reason codes for additional information.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
031 (49)	Key identifiers contain a version number. The version number in a supplied key identifier (internal or external) is inconsistent with one or more fields in the key identifier, making the key identifier unusable. **User action**: Use a token containing the required version number. **REASONCODES**: ICSF 2738 (10040)
033 (51)	The encipher and decipher callable services sometime require text (plaintext or ciphertext) to have a length that is an exact multiple of 8 bytes. Padding schemes always create ciphertext with a length that is an exact multiple of 8. If you want to decipher ciphertext that was produced by a padding scheme, and the text length is not an exact multiple of 8, then an error has occurred. The CBC mode of enciphering requires a text length that is an exact multiple of 8. The value that the *text_length* parameter specifies is not a multiple of the cryptographic algorithm block length. **User action**: Review the requirements of the service you are using. Either adjust the text you are processing or use another process rule.
038 (56)	The master key verification pattern in the OCV is not valid.
03D (61)	The keyword supplied with the key_type parameter is not valid. **REASONCODES**: This reason code also corresponds to these ICSF reason codes: 2720 (10016), 2740 (10048), 274C (10060). See those reason codes for additional information.
03E (62)	The source key was not found. **REASONCODES**: ICSF 271C (10012)
03F (63)	This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a *key_type* of IMP-PKA for a key in importable form. **User action**: Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence. **REASONCODES**: ICSF 7F8 (2040)
040 (64)	The supplied key is not permitted to perform the requested operation. Probable causes are: • The private key can be used only for digital signature. Key management services are disallowed. • This service requires an RSA private key that is translatable. The specified key may not be used in the PKA Key Translate callable service. • The private key restricts the signature formatting rule it can be used with and the rule array indicates a different formatting rule. **User action**: Supply a private key with the correct key usage for the service.
041 (65)	The RSA public or private key specified a modulus length that is incorrect for this service. **User action**: Re-invoke the service with an RSA key with the proper modulus length. **REASONCODES**: ICSF 2B18 (11032) and 2B58 (11096)
042 (66)	The recovered encryption block was not a valid PKCS-1.2 or zero-pad format. (The format is verified according to the recovery method specified in the rule-array.) If the recovery method specified was PKCS-1.2, refer to PKCS-1.2 for the possible error in parsing the encryption block. **User action**: Ensure that the parameters passed to CSNDSYI or CSNFSYI are correct. Possible causes for this error are incorrect values for the RSA private key or incorrect values in the *RSA_enciphered_key* parameter, which must be formatted according to PKCS-1.2 or zero-pad rules when created. **REASONCODES**: ICSF 2B20 (11040)
043 (67)	DES or RSA encryption failed.
044 (68)	DES or RSA decryption failed.
046 (70)	Identifier tag for optional block is invalid: conflicts with IBM reserved tag, is a duplicate to a tag already found, is bad in combination with a tag already found when parsing a section of optional blocks, or is otherwise invalid. **User action:** Check the TR-31 key block header for correctness.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
048 (72)	The value specified for length parameter for a key token, key, or text field is not valid. **User action**: Correct the appropriate length field parameter. **REASONCODES**: This reason code also corresponds to these ICSF reason codes: 2AF8 (11000) and 2B14 (11028). See those reason codes for additional information.
05A (90)	Access is denied for this request. This is due to an access control point in the domain role either being disabled or an access control point being enabled that restricts the use of a parameter such as a rule array keyword. **User action:** Check the reference information for the callable service to determine which access control points are involved in the request. Contact the ICSF administrator to determine if the access control points are in the correct state. The access control points can be enabled/disabled using the TKE workstation.
064 (100)	A request was made to the Clear PIN generate or Encrypted PIN verify callable service, and the *PIN_length* parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive. **User action**: Correct the value in the *PIN_length* parameter to be within the valid range from 4 to 16. **REASONCODES**: ICSF BBC (3004)
065 (101)	A request was made to the Clear PIN generate callable service, and the *PIN_check_length* parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive. **User action**: Correct the value in the *PIN_check_length* parameter to be within the valid range from 4 to 16. **REASONCODES**: ICSF BC0 (3008)
066 (102)	The value of the decimalization table is not valid. **REASONCODES**: ICSF BE0 (3040)
067 (103)	The value of the validation data is not valid. **REASONCODES**: ICSF BE0 (3040)
068 (104)	The value of the customer-selected PIN is not valid or the PIN length does not match the value specified. **REASONCODES**: ICSF BE0 (3040)
069 (105)	The *trans_sec_parm* field in the *data_array* parameter is not valid. The key index may be incorrect. **User action**: Correct the value in the key index, held within the *trans_sec_parm* field, to hold a number from the valid range. **REASONCODES**: ICSF BC4 (3012)
06A (106)	A request was made to the Encrypted PIN Translate or the Encrypted PIN verify callable service, and the PIN block value in the *input_PIN_profile* or *output_PIN_profile* parameter has a value that is not valid. **User action**: Correct the PIN block value.
06B (107)	A request was made to the Encrypted PIN Translate callable service and the format control value in the *input_PIN_profile* or *output_PIN_profile* parameter has a value that is not valid. The only valid value is NONE. **User action**: Correct the format control value to NONE.
06C (108)	The value of the PAD data is not valid. **REASONCODES**: ICSF B08 (3016)
06D (109)	The extraction method keyword is not valid.
06E (110)	The value of the PAN data is not valid. **REASONCODES**: ICSF BE0 (3040)
06F (111)	A request was made to the Encrypted PIN Translate callable service. The *sequence_number* parameter was required, but was not the integer value 99999. **User action**: Specify the integer value 99999.
074 (116)	The supplied PIN value is incorrect. **User action**: Correct the PIN value. **REASONCODES**: ICSF BBC (3004)

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
079 (121)	The *source_key_identifier* or *inbound_key_identifier* you supplied is not a valid string. **User action**: In the PKA key generate service, an invalid exponent or modulus length was specified.
07A (122)	The *outbound_KEK_count* or *inbound_KEK_count* you supplied is not a valid ASCII hexadecimal string. **User action**: Check that you specified a valid ASCII hexadecimal string for the *outbound_KEK_count* or *inbound_KEK_count* parameter.
081 (129)	A Required Rule Array keyword was not specified. **User action:** Refer to the *rule_array* parameter described in this publication under the appropriate callable service for the correct value.
09A (154)	This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a *key_type* of IMP-PKA for a key in importable form. **User action**: Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence. **REASONCODES**: ICSF 7F8 (2040)
09B (155)	The value that the *generated_key_identifier* parameter specifies is not valid,or it is not consistent with the value that the *key_form* parameter specifies.
09C (156)	A keyword is not valid with the specified parameters. **REASONCODES**: ICSF 2790 (10128)
09D (157)	The *rule_array* parameter contents are incorrect. **User action**: Refer to the *rule_array* parameter described in this publication under the appropriate callable service for the correct value. **REASONCODES**: ICSF 7E0 (2016)
09F (159)	A parameter requires Rule Array keyword that is not specified. **User action:** Refer to the *rule_array* parameter described in this publication under the appropriate callable service for the correct value.
0A0 (160)	The key_type and the key_length are not consistent. **User action**: Review the *key_type* parameter provided and match it with the *key_length* parameter.
A2 (162)	A request was made to the Remote Key Export callable service, and the *certificate_parms* parameter contains incorrect values. One or more of the offsets and/or lengths for the modulus, public exponent, and/or digital signature would indicate overlap between two or all three of the fields within the *certificate* parameter. **User Action:** Correct the values in the certificate_parms parameter to indicate the actual offsets and lengths of the modulus, public exponent, and digital signature within the *certificate* parameter.
A4 (164)	Two parameters (perhaps the plaintext and ciphertext areas, or *text_in* and *text_out* areas) overlap each other. That is, some part of these two areas occupy the same address in memory. This condition cannot be processed. **User action**: Determine which two areas are responsible, and redefine their positions in memory.
0A5 (165)	The contents of a chaining vector passed to a callable service are not valid. If you called the MAC Generate callable service, or the MDC Generate callable service with a MIDDLE or LAST segmenting rule, the count field has a number that is not valid. If you called the MAC verification callable service, then this will have been a MIDDLE or LAST segmenting rule. **User action**: Check to ensure that the chaining vector is not modified by your program. The chaining vector returned by ICSF should only be used to process one message set, and not intermixed between alternating message sets. This means that if you receive and process two or more independent message streams, each should have its own chaining vector. Similarly, each message stream should have its own key identifier. If you use the same chaining vector and key identifier for alternating message streams, you will **not** get the correct processing performed. **REASONCODES**: ICSF 7F4 (2036)

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
AF (175)	The DES key token cannot be created because no control vector or a control vector of all zeroes is present. **User action**: Supply a key token with a valid control vector for the key type required.
0B4 (180)	A null key token was passed in the key identifier parameter. When the key type is TOKEN, a valid token is required. **User action**: Supply a valid token to the key identifier parameter.
0B5 (181)	This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a *key_type* of IMP-PKA for a key in importable form. **User action**: Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence. This reason code also corresponds to these ICSF reason codes: 7F8 (2040), 2B24 (11044) and 3E98 (16024). See those reason codes for additional information.
0B7 (183)	A cross-check of the control vector the key type implies has shown that it does not correspond with the control vector present in the supplied internal key identifier. **User action**: Change either the key type or key identifier. **REASONCODES**: ICSF 273C (10044)
0B8 (184)	An input pointer is null.
0C7 (199)	The public exponent in the RSA public key is not valid. **User action**: If you created a skeleton token using the CSNDPKB service, correct the key value structure and rerun the CSNDPKB service. If you are using a key generated on another system, the key cannot be used with ICSF.
0CC (204)	A memory allocation failed.
14F (335)	The requested function is not implemented on the coprocessor.
154 (340)	One of the input control vectors has odd parity.
157 (343)	Either the data block or the buffer for the block is too small.
159 (345)	Insufficient storage space exists for the data in the data block buffer.
15A (346)	The requested command is not valid in the current state of the cryptographic hardware component.
176 (374)	Less data was supplied than expected or less data exists than was requested. **REASONCODES**: ICSF 7D4 (2004) and ICSF 7E0 (2016)
181 (385)	The cryptographic hardware component reported that the data passed as part of the command is not valid for that command.
197 (407)	A PIN block consistency check error occurred. **REASONCODES**: ICSF BC8 (3016)
1B7 (439)	Key cannot be completed because all required key parts have not yet been accumulated, or key is already complete. **User action:** If the key is not already complete, add the required number of key parts to the key before completing it.
1B9 (441)	One or more input parameters indicates the key to be processed should be partial, but the key is not partial according to the CV or other control bits of the key. **User action:** Check that the partial key option of any input parameters is consistent with the partial key setting of any key tokens being used.
1BA (442)	A DES key with the control vector form bits indicating unique key parts has replicated key parts. **User action:** This key cannot be used with ICSF. Contact your ICSF administrator.
25D (605)	The number of output bytes is greater than the number that is permitted.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
2BF (703)	A new master key value was found to be one of the weak DES keys.
2C0 (704)	The new master key would have the same master key verification pattern as the current master key.
2C1 (705)	The same key-encrypting key was specified for both exporter keys.
2C2 (706)	While deciphering ciphertext that had been created using a padding technique, it was found that the last byte of the plaintext did not contain a valid count of pad characters. Note that some cryptographic processing has taken place, and the *clear_text* parameter may contain some or all of the deciphered text. **User action**: The *text_length* parameter was not reduced. Therefore, it contains the length of the base message, plus the length of the padding bytes and the count byte. Review how the message was padded prior to being enciphered. The count byte that is not valid was created prior to the message's encipherment. You may need to check whether the ciphertext was not created using a padding scheme. Otherwise, check with the creator of the ciphertext on the method used to create it. You could also look at the plaintext to review the padding scheme used, if any. **REASONCODES**: ICSF 7EC (2028)
2C3 (707)	The master key registers are not in the state required for the requested function. **User action**: Contact your ICSF administrator.
2CA (714)	A reserved parameter was not a null pointer or an expected value. **REASONCODES**: ICSF 844 (2116)
2CB (715)	A parameter was specified with a non-zero value. For example: **Key Token Build** The value of the *master_key_version_number* parameter must be zero when the KEY keyword is specified. **Key Token Build** The value of the *pad_character* parameter must be zero when building a MAC token. **DK PIN Change** The value of the *script_initialization_vector* parameter must be zero. **Recover PIN from Offset** The *reserved_1* field must be zero. **User action**: Check that you specified the valid value for the parameter. **REASONCODES**: ICSF 834 (2100)
2CF (719)	The RSA-OAEP block did not verify when it decomposed. The block type is incorrect (must be X'03'). **User action**: Re-create the RSA-OAEP block. **REASONCODES**: ICSF 2B38 (11064)
2D0 (720)	The RSA-OAEP block did not verify when it decomposed. The random number I is not correct (must be non-zero with the high-order bit equal to zero). **User action**: Re-create the RSA-OAEP block. **REASONCODES**: ICSF 2B40 (11072)
2D1 (721)	The RSA-OAEP block did not verify when it decomposed. The verification code is not correct (must be all zeros). **User action**: Re-create the RSA-OAEP block. **REASONCODES**: ICSF 2BC3 (11068)
2F8 (760)	The RSA public or private key specified a modulus length that is incorrect for this service. **User action**: Re-invoke the service with an RSA key with the proper modulus length. **REASONCODES**: ICSF 2B48 (11080)
2FA (762)	The key values structure for CSNDPKB has a field in error. A length or format is not correct. **User action**: Correct the key values structure.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
302 (770)	A reserved field in a parameter, probably a key identifier, has a value other than zero. **User action**: Key identifiers should not be changed by application programs for other uses. Review any processing you are performing on key identifiers and leave the reserved fields in them at zero. This reason code also corresponds to these ICSF reason codes: 7E8 (2024) and 2B00 (11008). See those reason codes for additional information. **REASONCODES**: ICSF 2B00 (11008)
309 (777)	The authentication data or its length is not valid. **User action**: Correct the authorization data parameters.
30F (783)	The command is not permitted by the function's control vector value. **REASONCODES**: ICSF Return code 12, reason code 2B0C (11020)
335 (821)	The subject distinguished name (SDN) provided is either missing, malformed, or of invalid length. **User action**: Correct the SDN value and retry the function.
336 (822)	The issuer distinguished name (IDN) provided is either missing, malformed, or of invalid length. **User action**: Correct the IDN value and retry the function.
337 (823)	The serial number provided is either unexpected, missing, malformed, or of invalid length. **User action**: Correct the serial number value and retry the function.
339 (825)	The extension data provided is either unexpected, missing, malformed, or of invalid length. **User action**: Correct the extension data and retry the function.
33A (826)	The validity (notBefore/notAfter) expiration days value provided is either unexpected, missing, or out of range. **User action**: Correct the validity value and retry the function.
33B (827)	The pathLenConstraint value provided is either unexpected, missing, or out of range. **User action**: Correct the pathLenConstraint value and retry the function.
33D (829)	Error in GSK/SSL/ASN.1 processing. **User action**: Contact the IBM Support Center.
33E (830)	ASN.1 DER encoding error detected in an input data. More data was expected, but none was found. **User action**: Correct the ASN.1 DER encoded input.
33F (831)	ASN.1 DER encoding error detected in an input value. A length value is not valid. **User action**: Correct the ASN.1 DER encoded input.
341 (833)	ASN.1 DER encoding error detected in an input value. An attribute value separator is missing. **User action**: Correct the ASN.1 DER encoded input.
342 (834)	ASN.1 DER encoding error detected in an input value. An unknown attribute identifier was found. **User action**: Correct the ASN.1 DER encoded input.
343 (835)	ASN.1 DER encoding error detected in an input value. An object identifier syntax error was found. **User action**: Correct the ASN.1 DER encoded input.
345 (837)	ASN.1 DER encoding error detected in an input value. An validity interval is not valid. **User action**: Correct the ASN.1 DER encoded input.
346 (838)	ASN.1 DER encoding error detected in an input value. **User action**: Correct the ASN.1 DER encoded input.Error in ASN.1 processing. X.500 name syntax error.
347 (839)	ASN.1 DER encoding error detected in an input value. An unexpected data type was found. **User action**: Correct the ASN.1 DER encoded input.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
349 (841)	ASN.1 DER encoding error detected in an input value. A character string cannot be converted. **User action**: Correct the ASN.1 DER encoded input.
34A (842)	ASN.1 DER encoding error detected in an input value. Indefinite-length encoding was encountered, but is not supported. **User action**: Correct the ASN.1 DER encoded input.
34B (843)	ASN.1 DER encoding error detected in an input value. A data element must be constructed, but is not. **User action**: Correct the ASN.1 DER encoded input.
34D (845)	ASN.1 DER encoding error detected in an input value. A data element must be an ASN.1 primitive, but it is not. **User action**: Correct the ASN.1 DER encoded input.
34E (846)	ASN.1 DER encoding error detected in an input value. Indefinite-length encoding was found, but is not allowed. **User action**: Correct the ASN.1 DER encoded input.
34F (847)	ASN.1 DER encoding error detected in an input value. A data encoding is not valid. **User action**: Correct the ASN.1 DER encoded input.
351 (849)	ASN.1 DER encoding error detected in an input value. Data value overflow was encountered. **User action**: Correct the ASN.1 DER encoded input.
352 (850)	ASN.1 DER encoding error detected in an input value. The unused bit count in a BIT STRING is not valid. **User action**: Correct the ASN.1 DER encoded input.
353 (851)	ASN.1 DER encoding error detected in an input value. An unused bit count was encountered, but it is not valid for a segmented bit string. **User action**: Correct the ASN.1 DER encoded input.
356 (854)	ASN.1 DER encoding error detected in an input value. Excess data was found at the end of the data element. **User action**: Correct the ASN.1 DER encoded input.
357 (855)	ASN.1 DER encoding error detected in an input value. A parameter is not valid. **User action**: Correct the ASN.1 DER encoded input.
359 (857)	ASN.1 DER encoding error detected in an input value. A data value is not present where it is expected. **User action**: Correct the ASN.1 DER encoded input.
35A (858)	ASN.1 DER encoding error detected in an input value. A selection value is not within the valid range. **User action**: Correct the ASN.1 DER encoded input.
35B (859)	ASN.1 DER encoding error detected in an input value. No selection was found where it was expected. **User action**: Correct the ASN.1 DER encoded input.
35D (861)	ASN.1 DER encoding error detected in an input value. Syntax is already set. **User action**: Correct the ASN.1 DER encoded input.
35E (862)	ASN.1 DER encoding error detected in an input value. The codeset is not allowed. **User action**: Correct the ASN.1 DER encoded input.
35F (863)	ASN.1 DER encoding error detected in an input value. The specified attribute value is not valid. **User action**: Correct the ASN.1 DER encoded input.
361 (865)	ASN.1 DER encoding error detected in an input value. An attribute value is missing. **User action**: Correct the ASN.1 DER encoded input.
362 (866)	ASN.1 DER encoding error detected in an input value. An object identifier element count is not valid. **User action**: Correct the ASN.1 DER encoded input.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
363 (867)	ASN.1 DER encoding error detected in an input value. An incorrect value for the first object identifier element was found. **User action**: Correct the ASN.1 DER encoded input.
365 (869)	ASN.1 DER encoding error detected in an input value. An incorrect value for the second object identifier element was found. **User action**: Correct the ASN.1 DER encoded input.
366 (870)	ASN.1 DER encoding error detected in an input value. The specified version is not supported. **User action**: Correct the ASN.1 DER encoded input.
367 (871)	A duplicate extension was found in a certificate. **User action**: Correct the certificate.
369 (873)	The extension data provided conflicts with the rule array data provided. **User action**: Modify the DER encoded extensions or the rule array, or both.
36A (874)	The Elliptic Curve algorithm was used, but it is not supported. **User action**: Correct the algorithm.
36B (875)	A certificate signature was not found where it is expected. **User action**: Correct the certificate.
36E (878)	The cryptographic algorithm specified is not supported. **User action**: Correct the algorithm.
36F (879)	An error was found in the Base64 encoding of an input certificate. **User action**: Correct the certificate.
371 (881)	An unrecognized file or message encoding was found. **User action**: Correct the file or message.
372 (882)	A request cannot be processed because the coprocessor internal clock has not been set. **User action**: Use the TKE workstation to set the internal clock.
373 (883)	The key specified is not supported by encryption or the signature algorithm. **User action**: Correct the key.
375 (885)	The input certificate has an invalid or missing KeyUsage extension. **User action**: Correct the certificate.
376 (886)	An input certificate extension is not supported. **User action**: Correct the extension.
377 (887)	The input certificate does not have a valid signature. **User action**: Correct the certificate. If this is a compliance check, ensure that the root certificate that signed the operational certificate is loaded onto all CEX6C and later CCA coprocessors.
37B (891)	Error in certificate processing. Signature not supplied. **User action**: Correct the certificate or certificates. If using a self-signed certificate in the Digital Signature Verify (CSNDDSV/CSNFDSV) callable service, check the required hardware table for the service to ensure the correct hardware is available. If this is a compliance check, ensure that a root certificate is loaded onto all CEX6C and later CCA coprocessors.
37D (893)	An extension has an incorrect critical indicator. **User action**: Correct the extension.
37E (894)	A required certificate extension was not supplied. **User action**: Supply the required extension.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
37F (895)	During certificate processing, a certificate was found to be not valid for the host. **User action**: Correct the certificate.
381 (897)	A subject distinguished name (SDN) is not valid. **User action**: Correct the SDN.
382 (898)	Certificate extension data is not valid. **User action**: Correct the extension.
383 (899)	A certificate validation option is not valid. **User action**: Correct the validation option.
385 (901)	Name constraint restrictions have been violated in a certificate or certificate chain. **User action**: Correct the certificate or certificates.
387 (903)	A certificate chain is not trusted. **User action**: Correct the certificate or certificates.
389 (905)	The required certificate basic constraints extension was not found. **User action**: Supply the required extension.
38A (906)	During certificate processing, an internal error occurred. **User action**: Contact the IBM support center.
38B (907)	Error in certificate processing. Issuer certificate not found. **User action**: Review the supplied certificates and correct the problem.
38D (909)	The name format is not supported. **User action**: Specify a supported name format.
38E (910)	The end entity certificate for a certificate or certificate chain has not been loaded into the coprocessor adapter. **User action**: The root certificate must be loaded using the TKE workstation. Contact the system administrator.
38F (911)	Error in certificate processing. Certificate is expired. **User action**: Review the supplied certificates and correct the problem.
391 (913)	A certificate is not valid according to its validity period. **User action**: Correct the certificate.
392 (914)	A certificate issuer distinguished name (IDN) is not valid. **User action**: Correct the IDN.
393 (915)	Error in certificate processing. Certificate is revoked. **User action**: Review the supplied certificates and correct the problem.
395 (917)	A certificate numeric value is not valid. **User action**: Correct the numeric value.
396 (918)	A certificate variable argument security level is not valid. **User action**: Correct the security level.
397 (919)	A variable argument validate root was found that is not valid. **User action**: Correct the argument validate root.
399 (921)	A variable argument count is not valid. **User action**: Correct the argument count.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
39A (922)	Extended key usage comparison checking failed. **User action**: Correct the key usage or usages.
39B (923)	An input certificate does not have an extended key usage extension. **User action**: Correct the certificate.
39D (925)	An extended key usage setting is not supported for this operation **User action**: Correct the key usage or usages.
39E (926)	An extended key usage was not supplied. **User action**: Correct the key usage or usages input.
39F (927)	An extended key usage input count is not valid. **User action**: Correct the key usage input count.
3A2 (930)	An incorrect key usage was found. **User action**: Correct the key usage.
3A5 (933)	Error in certificate processing. Acceptable policy intersection cannot be found. **User action**: Review the supplied certificates and correct the problem. Consider using the RFC-ANY keyword.
3AD (941)	A certificate presented to use as an end entity has a true value for cA in basic constraints certificate extension. **User action**: Correct the certificate.
3BF (959)	The coprocessor adapter contains certificates signed by the certificate. **User action**: Use an operational certificate that has not been used to sign other certificates.
3C5 (965)	Error in X.509 certificate processing. The enumeration value is not valid. **User action**: Review the supplied certificates and correct the problem.
3C6 (966)	The certificate revocation list provided is either missing, malformed, or the length is not valid. **User action**: Supply a valid certificate revocation list.
3C7 (967)	The TR-34 input token provided is either missing, malformed, or the length is not valid. **User action**: Provide a correct, well-formed TR-34 token as required by the service.
3C9 (969)	The freshness indicator provided is either missing, malformed, or the length is not valid. **User action**: Provide a correct and well-formed freshness indicator as required by the service.
3CA (970)	Error in X.509 certificate processing. The certificate revocation list is expired. **User action**: Obtain an up-to-date certificate revocation list for processing.
3CB (971)	Error in X.509 certificate processing. The revocation information is not yet valid. **User action**: Obtain an up-to-date certificate revocation list for processing.
3CD (973)	Error in X.509 certificate processing. The certificate revocation list cannot be found. **User action**: Obtain an up-to-date certificate revocation list for processing.
3CE (974)	The Signed Attributes data is either missing, malformed, or of invalid length.
3CF (975)	The Credential IDs provided in separate inputs do not match.
3D1 (977)	The clear KBH does not match encrypted KBH.
3D2 (978)	The random data in key token does not match reference value.
3D3 (979)	Error in certificate processing. The PKCS #7 CMS version is not supported.
3D5 (981)	Error in certificate processing. An unsupported PKCS #7 content type is encountered.
3D6 (982)	Error in certificate processing. The PKCS #7 content information does not contain any content data.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
3D7 (983)	Error in certificate processing. The API is not supported.
3D9 (985)	Error in certificate processing. An unsupported version is encountered.
3DA (986)	Error in certificate processing. An X.509 cryptographic algorithm is not available.
3DB (987)	Error in certificate processing. A recipient certificate is not found while creating or processing an enveloped message.
3DD (989)	Error in certificate processing. The encryption key size is not supported.
3DE (990)	Error in certificate processing. A signer certificate is not found while creating or processing a signed message.
3DF (991)	Error in certificate processing. The specified digest algorithm and the key algorithm are incompatible.
3E1 (993)	Error in certificate processing. The set of authenticated attributes that are supplied within the *attributes_signers* parameter must not include the content-type authenticated attribute.
3E2 (994)	Error in certificate processing. The set of authenticated attributes that are supplied within the *attributes_signers* parameter must not include the message-digest authenticated attribute.
3E3 (995)	Error in certificate processing. DES and Triple DES encryption keys must have odd parity for each key byte.
3E5 (997)	The QSA algorithm identifier provided is either missing, malformed, or not valid.
3E6 (998)	The QSA algorithm parameters provided are either missing, malformed, or not valid.
3ED (1005)	During X.509 certificate processing, the private/public key pair provided in one parameter and the certificate were mismatched. **User action**: Provide matching private/public key pair and certificate.
3EE (1006)	The provided TR-31 Optional Block is not allowed with the provided Key Block Header algorithm. **User action**: Correct the optional block or algorithm.
3EF (1007)	The provided TR-31 Optional Block is malformed. **User action**: Correct the optional block.
3F1 (1009)	The provided TR-31 Optional Block contains data that cannot be used by the HSM (crypto card), so it was rejected. **User action**: Correct the optional block.
3F2 (1010)	The provided TR-31 Optional Block is not allowed with the provided Key Block Header usage. When using CSNBT31X/CSNET31X with the COMP-TAG keyword, this indicates that the opt blocks parameter is not empty. **User action**: Correct the optional block.
3F3 (1011)	The provided set of TR-31 Optional Blocks contains duplicate block IDs provided either directly as input or indirectly by specifying rule array keywords. **User action**: Correct the optional blocks or rule array keywords.
3F5 (1013)	The provided skeleton attributes do not match the attributes provided in the optional block. **User action**: Correct the attributes or optional block.
3F6 (1014)	The DA optional block has more members than are allowed (expected one member). **User action**: Correct the DA optional block.
3F7 (1015)	TR-31 block support is not available for this option. **User action**: Use supported optional blocks.
401 (1025)	Registered public key or retained private key name already exists.
402 (1026)	Registered public key or retained private key name does not exist.
405 (1029)	There is an error in the Environment Identification data.
40B (1035)	The signature does not match the certificate signature during an RKX call. **User Action:** Check that the key used to check the signatures is the correct.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
41A (1050)	A KEK RSA-enciphered at this node (EID) cannot be imported at this same node.
41C (1052)	Token identifier of the trusted block's header section is in the range 0x20 and 0xFF. **User Action:** Check the token identifier of the trusted block.
41D (1053)	The Active flag in the trusted block's trusted block section 0x14 is not disabled. **User Action:** Use the trusted block create callable service to create an inactive/external trusted block.
41E (1054)	Token identifier of the trusted block's header section is not 0x1E (external). **User Action:** Use the trusted block create callable service to create an inactive/external trusted block.
41F (1055)	The Active flag of the trusted block's trusted block section 0x14 is not enabled. **User Action:** Use the trusted block create callable service to create an active/external trusted block.
420 (1056)	Token identifier of the trusted block's header section is not 0x1F (internal). **User Action:** Use the PKA public key import callable service to import the trusted block.
421 (1057)	Trusted block rule section 0x12 Rule ID does not match input parameter rule ID. **User Action:** Verify the trusted block used has the rule section specified.
422 (1058)	Trusted block contains a value that is too small/too large.
423 (1059)	A trusted block parameter that must have a value of zero (or a grouping of bits set to zero) is invalid.
424 (1060)	Trusted block public key section failed consistency checking.
425 (1061)	Trusted block contains extraneous sections or subsections (TLVs). **User Action:** Check the trusted block for undefined sections or subsections.
426 (1062)	Trusted block is missing sections or subsections (TLVs). **User Action:** Check the trusted block for required sections and subsections applicable to the callable service invoked.
427 (1063)	Trusted block contains duplicate sections or subsections (TLVs). **User Action:** Check the trusted block's sections and subsections for duplicates. Multiple rule sections are allowed.
428 (1064)	Trusted block expiration date has expired (as compared to the 4764 clock). **User Action:** Validate the expiration date in the trusted block's trusted information section's Activation and Expiration Date TLV Object.
429 (1065)	Trusted block expiration date is at a date prior to the activation date. **User Action:** Validate the expiration date in the trusted block's trusted information section's Activation and Expiration Date TLV Object.
42A (1066)	Trusted Block Public Key Modulus bit length is not consistent with the byte length. The bit length must be less than or equal to byte length * 8 and greater than (byte length - 1) * 8.
42B (1067)	Trusted block Public Key Modulus Length in bits exceeds the maximum allowed bit length as defined by the Function Control Vector.
42C (1068)	One or more trusted block sections or TLV Objects contained data which is invalid (an example would be invalid label data in label section 0x13).
42D (1069)	Trusted block verification was attempted by a function other than CSNDDSV, CSNDKTC, CSNDKPI, CSNDRKX, or CSNDTBC.
42E (1070)	Trusted block rule ID contained within a Rule section contains invalid characters.
42F (1071)	The source key's length or CV does not match what is expected by the rule section in the trusted block that was selected by the rule ID input parameter.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
430 (1072)	The activation data is not valid. **User Action:** Validate the activation data in the trusted block's trusted information section's Activation and Expiration Date TLV Object.
431 (1073)	The source-key label does not match the template in the export key DES token parameters TLV object of the selected trusted block rule section.
432 (1074)	The control-vector value specified in the common export key parameters TLV object in the selected rule section of the trusted block contains a control vector that is not valid.
433 (1075)	The source-key label template in the export key DES token parameters TLV object in the selected rule section of the trusted block contains a label template that is not valid.
439 (1081)	The ISO-1 format PIN block is not allowed by your configuration. The **Disallow PIN block format ISO-1** access control is enabled. **User Action:** Check with your ICSF administrator.
43A (1082)	The key strength of the input or output key is not allowed by your access control point settings. For DES/TDES keys, consider also the effective strength of the key, whether there are repeated 56-bit sections among K1,K2 or K1,K2,K3. For example, if effective single-length TDES keys are disabled by ACP, consider if K1=K2, K2=K3, or K1=K2=K3. **User Action:** If weak key usage is permitted by your installation, determine the failing key strength and disable access control point 'Disable 56-bit length DES Keys', 'Disable 56-bit effective length DES keys', 'Disable RSA keys with less than 1024-bit modulus length', 'Disable RSA keys with less than 2048-bit modulus length', or 'Disable ECC keys weaker than 224-bit'.
043B (1083)	When the wrap type in the key token indicates WRAPENH3, this service requires a skeleton token. The token cannot contain a key. **User Action:** Supply a skeleton key token with the required key attributes for this service.
043D (1085)	The clear key value of a key to be wrapped with the WRAPENH3 method has a value of all zeros for K2, K3, or both. This value is not allowed. **User Action:** Shorten the key to not include a value of all zeros or change the value to a non-zero value.
7D1 (2001)	TKE: DH generator is greater than the modulus.
7D2 (2002)	TKE: DH registers are not in a valid state for the requested operation.
7D3 (2003)	TKE: TSN does not match TSN in pending change buffer.
7D4 (2004)	A length parameter has an incorrect value. The value in the length parameter could have been zero (when a positive value was required) or a negative value. If the supplied value was positive, it could have been larger than your installation's defined maximum, or for MDC generation with no padding, it could have been less than 16 or not an even multiple of 8. **User action**: Check the length you specified. If necessary, check your installation's maximum length with your ICSF administrator. Correct the error. **REASONCODES**: TSS 019 (025)
7D5 (2005)	TKE: PCB data exceeds maximum data length.
7D8 (2008)	Two parameters (perhaps the plaintext and ciphertext areas, or *text_in* and *text_out* areas) overlap each other. That is, some part of these two areas occupy the same address in memory. This condition cannot be processed. **User action**: Determine which two areas are responsible, and redefine their positions in memory. **REASONCODES**: TSS 0A4 (164)
7D9 (2009)	TKE: ACI cannot load both roles and profiles in one call.
7DA (2010)	TKE: ACI can only load one role or one profile at a time.
7DB (2011)	TKE: DH transport key algorithm match.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
7DC (2012)	The *rule_array_count* parameter contains a number that is not valid. **User action**: Refer to the *rule_array_count* parameter described in this publication under the appropriate callable service for the correct value. **REASONCODES**: TSS 023 (035)
7DD (2013)	TKE: Length of hash pattern for keypart is not valid for DH transport key algorithm specified.
7DE (2014)	TKE: PCB buffer is empty.
7DF (2015)	An error occurred in the Domain Manager.
7E0 (2016)	The *rule_array* parameter contents are incorrect. One or more of the rules specified are not valid for this service OR some of the rules specified together may not be combined. **User action**: Refer to the *rule_array* parameter described in this publication under the appropriate callable service for the correct value. **REASONCODES**: TSS 021 (033)
7E2 (2018)	The *form* parameter specified in the random number generate callable service should be ODD, EVEN, or RANDOM. One of these values was not supplied. **User action**: Change *form* parameter to use one of the required values for the *form* parameter. **REASONCODES**: TSS 021 (033)
7E3 (2019)	TKE: Signature in request CPRB did not verify.
7E4 (2020)	TKE: TSN in request CPRB is not valid.
7E8 (2024)	A reserved field in a parameter, probably a key identifier, has a value other than zero. **User action**: Key identifiers should not be changed by application programs for other uses. Review any processing you are performing on key identifiers and leave the reserved fields in them at zero.
7EB (2027)	TKE: DH transport key hash pattern does not match.
7EC (2028)	While deciphering ciphertext that had been created using a padding technique, it was found that the last byte of the plaintext did not contain a valid count of pad characters. Note that all cryptographic processing has taken place, and the *clear_text* parameter contains the deciphered text. When deciphering ciphertext that had been created using Galois/Counter Mode (GCM) either through PKCS #11 Secret key decrypt (CSFPSKD or CSFPSKD6), PKCS #11 Unwrap Key (CSFPUWK and CSFPUWK6), or Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD, or CSNESYD1), the GCM tag provided did not match the data provided. No cleartext was returned. **User action**: The *text_length* parameter was not reduced. Therefore, it contains the length of the base message, plus the length of the padding bytes and the count byte. Review how the message was padded prior to it being enciphered. The count byte that is not valid was created prior to the message's encipherment. You may need to check whether the ciphertext was not created using a padding scheme. Otherwise, check with the creator of the ciphertext on the method used to create it. You could also look at the plaintext to review the padding scheme used, if any. If using GCM, verify that the parameters provided (ciphertext, additional authenticated data, and tag) match those provided to, or returned from, the corresponding call to PKCS #11 Secret key encrypt (CSFPSKE or CSFPSKE6), PKCS #11 Wrap Key (CSFPWPK and CSFPWPK6), or Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE, or CSNESYE1). **REASONCODES**: TSS 2C2 (706)
7ED (2029)	TKE: Request data block hash does not match hash in CPRB.
7EE (2030)	TKE: DH supplied hash length is not correct.
7EF (2031)	Reply data block too large.
7F1 (2033)	TKE: Change type does not match PCB change type.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
7F4 (2036)	The contents of a chaining vector or the chaining data passed to a callable service are not valid. If you called the MAC Generate callable service, or the MDC Generate callable service with a MIDDLE or LAST segmenting rule, the count field has a number that is not valid. If you called the MAC verification callable service, then this will have been a MIDDLE or LAST segmenting rule. If you called the Symmetric Key Encipher, Symmetric Key Decipher, PKCS #11 Secret Key Encrypt or PKCS #11 Secret Key Decrypt, the chaining data passed is unusable, either because a CONTINUE or FINAL was not preceded by an INITIAL or CONTINUE, or because an attempt was made to continue chaining calls after a partial block has been processed. **User action**: Check to ensure that the chaining vector or chaining data is not modified by your program. The chaining vector or chaining data returned by ICSF should only be used to process one message set, and not intermixed between alternating message sets. This means that if you receive and process two or more independent message streams, each should have its own chaining vector. Similarly, each message stream should have its own key identifier. If you use the same chaining vector and key identifier for alternating message streams, you will **not** get the correct processing performed. **REASONCODES**: TSS 0A5 (165)
7F6 (2038)	No RSA private key information was provided in the supplied token. **User action**: Check that the token supplied was of the correct type for the service.
7F8 (2040)	This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a *key_type* of IMP-PKA for a key in importable form. **User action**: Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence. **REASONCODES**: TSS 03F (063) and TSS 09A (154)
7FC (2044)	The caller must be in task mode, not SRB mode.
800 (2048)	The *key_form* is not valid for the *key_type* **User action**: Review the *key_form* and *key_type* parameters. For a *key_type* of IMP-PKA, the secure key import callable service supports only a *key_form* of OP.
802 (2050)	A DUKPT keyword was specified, but there is an error in the *PIN_profile* key serial number. **User action**: Correct the PIN profile key serial number.
803 (2051)	Invalid message length in OAEP-decoded information.
804 (2052)	A single-length key, passed to the secure key import callable service in the *clear_key* parameter, must be padded on the right with binary zeros. The fact that it is a single-length key is identified by the *key_form* parameter, which identifies the key as being DATA, MACGEN, MACVER, and so on. **User action**: If you are providing a single-length key, pad the parameter on the right with zeros. Alternatively, if you meant to pass a double-length key, correct the *key_form* parameter to a valid double-length key type.
805 (2053)	No message found in OAEP-decoded information.
806 (2054)	Invalid RSA enciphered key cryptogram; OAEP optional encoding parameters failed validation.
807 (2055)	Based on the hash method and size of the symmetric key specified, the RSA public key size is too small to format the symmetric key into a PKOAEP2 message.
808 (2056)	The *key_form* parameter is neither IM nor OP. Most constants, these included, can be supplied in lowercase or uppercase. Note that this parameter is 4 bytes long, so the value IM or OP is not valid. They must be padded on the right with blanks. **User action**: Review the value provided and change it to IM or OP, as required. **REASONCODES**: TSS 029 (041)
80C (2060)	The value specified for the *key_length* parameter of the key generate callable service is not valid. **User action**: Review the value provided and change it as appropriate. **REASONCODES**: TSS 02B (043)

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
810 (2064)	The key_type and the key_length are not consistent. **User action**: Review the *key_type* parameter provided and match it with the *key_length* parameter. **REASONCODES**: TSS 0A0 (160)
811 (2065)	A null key token was not specified for a key identifier parameter. **User action:** Check the service description and determine which key identifier parameter must be a null token.
813 (2067)	TKE: A key part register is in an invalid state. This includes the case where an attempt is made to load a FIRST key part, but a register already contains a key or key part with the same key name. **User action**: Supply a different label name for the key part register or clear the existing key part register with the same label name.
814 (2068)	You supplied a key identifier or token to the key generate, key import, multiple secure key import, key export, or CKDS key record write callable service. This key identifier holds an importer or exporter key, and the NOCV bit is on in the token. Only programs running in supervisor state or in a system key (key 0–7) may provide a key identifier with this bit set on. Your program was not running in supervisor state or a system key. **User action**: Either use a different key identifier, or else run in supervisor state or a system key.
815 (2069)	TKE: The control vector in the key part register does not match the control vector in the key structure.
816 (2070)	TKE: All key part registers are already in use. **User action**: Either free existing key part registers by loading keys from ICSF or clearing selected key part registers from TKE or select another coprocessor for loading the key part register.
817 (2071)	TKE: The key part hash pattern supplied does not match the hash pattern of the key part currently in the register.
81B (2075)	TKE: The length of the key part received is different from the length of the accumulated value already in the key part register.
81C (2076)	A request was made to the key import callable service to import a single-length key. However, the right half of the key in the *source_key_identifier* parameter is not zeros. Therefore, it appears to identify the right half of a double-length key. This combination is not valid. This error does not occur if you are using the word TOKEN in the *key_type* parameter. **User action**: Check that you specified the value in the *key_type* parameter correctly, and that you are using the correct or corresponding *source_key_identifier* parameter.
81D (2077)	TKE: An error occurred storing or retrieving the key part register data. **User action**: Verify that the selected coprocessor is functioning correctly and retry the operation.
81F (2079)	An encrypted symmetric key token was passed to the service (CSNBSMG, CSNBSMV, CSNBSYD, CSNBSYE, or CSNDPKE) where only a clear key token is supported.
829 (2089)	The algorithm does not match the algorithm of the key identifier. **User action**: Make sure the *rule_array* keywords specified are valid for the type of key specified. Refer to the *rule_array* parameter described in this publication under the appropriate callable service for the valid values.
82D (2093)	Key identifiers contain a version number. The version number in a supplied key identifier (internal or external) is inconsistent with one or more fields in the key identifier, making the key identifier unusable. **User action**: Use a token containing the required version number.
82E (2094)	The *key_length* value is not compatible with the *key_form* value.
82F (2095)	The value in the *key_form* parameter is incompatible with the value in the *key_type* parameter. **User action**: Ensure compatibility of the selected parameters.
831 (2097)	The value in the *key_identifier_length* parameter is incompatible with the value in the *key_type* parameter. **User action**: Ensure compatibility of the selected parameters.
832 (2098)	Either a key bit length that was not valid was found in an AES key token (length not 128, 192, or 256 bits) or a version X'01' DES token had a token-marks field that was not valid.
833 (2099)	Encrypted key length in an AES key token was not valid when an encrypted key is present in the token.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
834 (2100)	A parameter was specified with a non-zero value. For example: **Key Token Build** The value of the *master_key_version_number* parameter must be zero when the KEY keyword is specified. **Key Token Build** The value of the *pad_character* parameter must be zero when building a MAC token. **DK PIN Change** The value of the *script_initialization_vector* parameter must be zero. **PKA Key Generate** The value of the *regeneration_data_length* parameter must be zero when generating a DSS key. **User action**: Check that you specified the valid value for the parameter. **REASONCODES**: TSS 2CB (715)
836 (2102)	In operational key load, the key part register specified is incompatible with the rule provided.
838 (2104)	An input character is not in the code table. **User action**: Correct the code table or the source text. **REASONCODES**: TSS 02D (045)
83C (2108)	An unused field must be binary zeros, and an unused key identifier field generally must be zeros. **User action**: Correct the parameter list. **REASONCODES**: TSS 02F (047)
83E (2110)	The supplied symmetric key token is wrapped using a method that is not supported by the CCA coprocessor or this release of ICSF. The token cannot be used for this request. **User action**: See "CCA key wrapping" on page 20 for support requirements.
83F (2111)	There is an inconsistency between the wrapping information in the key token and the request to wrap a key.
840 (2112)	The length is incorrect for the key type. **User action**: Check the key length parameter. DATA keys may have a length of 8, 16, or 24. MAC keys must have a length of 8. All other keys should have a length of 16. Also check that the parameters are in the required sequence.
841 (2113)	A key token contains invalid payload. **User action:** Re-create the key token.
844 (2116)	Parameter contents or a parameter value is not correct. **User action**: Specify a valid value for the parameter. **REASONCODES**: TSS 021 (033)
846 (2118)	Invalid value or values in TR-31 key block header. **User action:** Check the TR-31 key block header for correctness. Also check that the PADDING optional block is the last optional block in a set of optional blocks.
847 (2119)	"Mode" value in the TR-31 header is invalid or is not acceptable in the chosen operation. **User action:** Check the TR-31 key block header for correctness.
849 (2121)	"Algorithm" value in the TR-31 header is invalid or is not acceptable in the chosen operation. **User action:** Check the TR-31 key block header for correctness.
84A (2122)	If importing a TR-31 key block, the exportability byte in the TR-31 header contains a value that is not supported. If exporting a TR-31 key block, the requested exportability is inconsistent with the key block. For example a 'B' Key Block Version ID key can only be wrapped by a KEK that is wrapped in CBC mode, the ECB mode KEK violates ANSI X9.24. **User action:** Check the TR-31 key block header for correctness.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
84B (2123)	The length of the cleartext key in the TR-31 block is invalid, for example the algorithm is “D” for single-DES but the key length is not 64 bits. **User action:** Check that the values in the TR-31 header are consistent with the key fields.
84D (2125)	The Key Block Version ID in the TR-31 header contains an invalid value. **User action:** Check the TR-31 key block header for correctness.
84E (2126)	The key usage field in the TR-31 header contains a value that is not supported for import of the key into CCA. **User action:** Check the TR-31 key block header for correctness.
84F (2127)	The key usage field in the TR-31 header contains a value that is not valid with the other parameters in the header. **User action:** Check the TR-31 key block header for correctness
851 (2129)	A parameter to a TR-31 service such as a TR-31 key block, a set of optional blocks, or a single optional block contains invalid characters. It may be that the parameter contains EBCDIC characters when ASCII is expected or vice-versa, or the wrong characters were found in a field which only accepts a limited range of characters. For example some length fields can be populated by characters '0' - '9' and 'A' - 'F', while other length fields can only contain characters '0' - '9'. **User action:** Check the TR-31 parameters for correctness
852 (2130)	The CV carried in the TR-31 key block optional blocks is inconsistent with other attributes of the key **User action:** Check the TR-31 key block header for correctness.
853 (2131)	The MAC validate step failed for a parameter. This may result from tampering, corruption, or attempting to use a different key to validate the MAC from the one used to generate it. **User action:** Check each parameter which includes a MAC for correctness. If the parameter is wrapped by a key-encrypting-key (KEK), ensure that the correct KEK is supplied.
856 (2134)	The requested PIN decimalization table does not exist or no PIN decimalization tables have been stored in the coprocessor.
857 (2135)	The supplied PIN decimalization table is not in the list of active tables stored in the coprocessor.
85D (2141)	A key verification pattern failed to verify. Either the key-encrypting key provided to unwrap an encrypted key contained in an external key-token is incorrect or an external key-token is invalid.
85E (2142)	The key usage attributes of the variable-length key token does not allow the requested operation. For example, the request might have been to encrypt data, but encryption is not allowed, or the request might have been to use the ECB cipher mode, but that mode is not allowed. **User action:** Use the variable-length key token in a manner consistent with its usage attributes or create a new key token with the desired attributes.
85F (2143)	On a call to Key Translate2 using the REFORMAT Encipherment rule and providing a variable-length AES token, the key management fields for input_key_token contain disallowed values or prohibit the operation. **User action:** Call Key Translate2 using a key token whose key-management fields contain allowed values.
861 (2145)	The service failed because a key would have been wrapped by a weaker key (transport or master key). This is disallowed by the "Prohibit weak wrapping - Transport keys" and "Prohibit weak wrapping - Master keys" access control points. **User action:** If weak key wrapping is to be allowed, disable access control point "Prohibit weak wrapping - Transport keys" and "Prohibit weak wrapping - Master keys" using the TKE workstation.
863 (2147)	The key type that was to be generated by this callable service is not valid. **User action:** Refer to the parameters described in this publication under the appropriate callable service for the correct parameter values.
865 (2149)	The key that was to be generated by this callable service is stronger than the input material. **User action:** Validate the key material is is at least as strong as the key to be generated.
869 (2153)	The input token is incompatible with the service (for example, clear key when encrypted key was expected).

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
86A (2154)	At least one key token passed to this callable service does not have the required key type for the specified function. For TR-31 tokens, this may indicate wrong usage or mode. For example, a KEK with mode of use E when D is required. **User action:** Refer to the parameters described in this publication under the appropriate callable service for the correct parameter values.
86E (2158)	There is a mismatch between ECC key tokens of either curve types, key lengths, or both. **User action:** Correct the inputs so that the curve types and the key lengths match.
86F (2159)	One or more key-encrypting key passed to the service is not valid for the service. **User action:** Check the requirements of the service and the key-encrypting keys you supplied, determine which key is incorrect and supply a key that is correct.
871 (2161)	The requested or default wrapping method conflicts with one or both input tokens. **User action**: On the call to the CVV Key Combine service, make sure that the desired wrapping method (either specified as a *rule_array* keyword or the default wrapping method) is consistent with the wrapping method of the input token or tokens. For example, an input token that can only be wrapped in the enhanced method (ENH-ONLY flag on in the CV) cannot produce an output token wrapped in the original method (ECB mode).
873 (2163)	A weak master key was detected when the final key part was loaded for the DES or RSA master key. A key is weak if any of the three parts are the same as another part. For example, when the first and third key parts are the same, the key is weak (effectively a double-length key). **User action**: Create new key values for the new master key and retry master key entry.
875 (2165)	The RSA key token contains a private section that is not valid with the service.
87A (2170)	Translation of text using an outbound key that has an effective key strength weaker than the effective strength of the inbound key is not allowed. **User action**: Provide an outbound key of equal or greater key strength of the inbound key.
87F (2175)	A weak PIN was presented. The PIN change has been rejected. **User action**: Provide another PIN.
881 (2177)	The PAN presented to the DK PAN change service was the same as the PAN in the encrypted PIN block. The change has been rejected. **User action**: Check the PAN parameters and correct the parameter in error.
882 (2178)	The PAN data supplied to the DK Deterministic PIN Generate service does not match the supplied data in the *account_info_ER* parameter. **User action**: Supply the correct PAN.
886 (2182)	A rule array keyword was passed to the TR-31 Import Callable Service (CSNBT31I) callable service or a TR-31 key block header field indicated that a TR-31 optional block was required. This optional block was not found in the TR-31 key block or the optional block as data that is invalid for the service call. **User action**: Check that the rule array and TR-31 key block are consistent for the requested service call.
895 (2197)	The input PIN could not be verified. **User action**: Ensure that the correct values were supplied for the parameters used to verify the PIN and ensure that the input PIN is correct.
896 (2198)	The supplied MAC was compared against a MAC calculated from the supplied parameters. The MACs did not match. **User action**: Ensure that the correct values were supplied for the parameters used to calculate the MAC and ensure that the supplied MAC is correct.
897 (2199)	A variable-length symmetric key-token (version X'05') contains invalid key-usage field data. **User action**: Supply a valid key token

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
899 (2201)	A variable-length symmetric key-token (version X'05') contains invalid key-management field data. **User action**: Supply a valid key token
89B (2203)	A malformed request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
89C (2204)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
89D (2205)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
89E (2206)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
89F (2207)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8A0 (2208)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8A1 (2209)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8A2 (2210)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8A3 (2211)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8A4 (2212)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8A5 (2213)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8A6 (2214)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8A7 (2215)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8A8 (2216)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
8A9 (2217)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8AA (2218)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8AB (2219)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8AC (2220)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8AD (2221)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8AE (2222)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8AF (2223)	A request caused processor recovery and ICSF takes a dump to capture the data for analysis. **User action**: Contact the system programmer to save the dump and contact the ICSF administrator to contact IBM.
8B5 (2229)	The type of key specified is not valid because a diversified key generating key must be used to derive this symmetric key type. **User action**: Supply a valid key type or token for the service.
8B7 (2231)	There was a problem converting or formatting the PAN. **User action**: Refer to the *rule_array* parameter described in this publication under the appropriate callable service for the valid values.
8B8 (2232)	There was a problem converting or formatting the cardholder name. **User action**: Refer to the *rule_array* parameter described in this publication under the appropriate callable service for the valid values.
8B9 (2233)	There was a problem converting or formatting the track 1 data. **User action**: Refer to the *rule_array* parameter described in this publication under the appropriate callable service for the valid values.
8BB (2235)	There was a problem converting or formatting the track 2 data. **User action**: Refer to the *rule_array* parameter described in this publication under the appropriate callable service for the valid values.
8BD (2237)	Data presented for VFPE processing is not in VFPE enciphered.
8BE (2238)	The supplied PIN profile has an invalid value. **User action**: Review the requirement of the service and correct the PIN profile.
8BF (2239)	The check digit compliance keyword denotes compliant check digit, but the input PAN does not have a compliant check digit.
8C3 (2243)	The key derivation section is missing or the attributes in the key derivation section do not match those in the output skeleton token. **User action**: Ensure the key derivation section is present and correctly matches the output skeleton token.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
8C5 (2245)	The CSNDEDH service was called and the key-token pedigree / key source of the ECC private key did not meet requirements; for example, it was not randomly generated. **User action**: Supply an ECC private key token with the correct pedigree.
8C6 (2246)	The CSNDPKG service was passed an ECC private key token that is ill-formed. The token has an associated data section version of X'01' and is missing the IBM extended associated data required for a version X'01' token. **User action**: Supply an ECC private key token with the correct IBM extended associated data for a version X'01' token.
8C7 (2247)	An error was encountered in the RSA PSS signature salt length. **User action**: Correct the PSS salt length.
8CE (2254)	The SECURE LOG SRDI that is stored on the coprocessor is full. No auditable actions are allowed. **User action**: Inform the system programmer that the coprocessor adapter secure log is full.
8FA (2298)	The hash function specified in the rule array has a digest size less than the bit length of the curve of the key. **User action**: Select a hash function large enough for the curve.
962 (2402)	An attempt was made to use a compliance-tagged key, but the domain is not in an active compliance mode. **User action:** Place the domain in compliance mode and retry the request.
963 (2403)	An attempt was made to use compliant-tagged tokens with a callable service that does not allow compliant-tagged tokens. **User action:** Either use a different callable service or non-compliant-tagged tokens.
965 (2405)	An attempt was made to perform a callable service operation that is not allowed with compliant-tagged tokens. Though the callable service supports compliant-tagged tokens, the specific operation requested of the service does not. **User action:** See the callable service documentation for restrictions on the use of compliant-tagged tokens.
966 (2406)	An attempt was made to use compliant-tagged tokens with non-compliant-tagged tokens. **User action:** Either use all compliant-tagged tokens or all non-compliant-tagged tokens in the service.
967 (2407)	An attempt was made to generate a compliant-tagged key token or check the compliance of a key token. The strength of the key is too weak for the configured compliance mode. **User action:** Increase the strength of the key to be compliant with the configured compliance mode and retry the request.
969 (2409)	An attempt was made to generate a compliant-tagged key token or check the compliance of a key token. The key type or usage is not compliant with the configured compliance mode. **User action:** Update the key type or usage to be compliant with the configured compliance mode and retry the request.
96A (2410)	An attempt was made to generate a NOCV KEK compliant-tagged key token or check the compliance of a NOCV KEK key token. NOCV KEKs cannot be compliant-tagged. **User action:** If attempting to generate a compliant-tagged KEK, recreate the skeleton token without the NOCV flag. NOCV KEKs cannot be compliant-tagged.
96D (2413)	An attempt was made to use a compliant-tagged KEK to wrap or unwrap an external key token, but the key type, a key attribute, or the wrapping method of the external key token is not compliant. **User action:** Only use compliant-tagged KEKs with compliant key tokens, or change to use a non-compliant-tagged KEK.
96E (2414)	An attempt was made to either check the compliance of or apply the compliance tag to an unsupported key token. **User action:** Only attempt to compliant-check or compliant-tag supported key tokens.
971 (2417)	The key derivation function value in the key token is invalid. The token is possibly corrupted. **User action:** Recreate the key token if possible.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
972 (2418)	A token or key block that is already compliant-tagged was passed to a callable service with either the "COMP-CHK" or "COMP-TAG" rule.
973 (2419)	Unable to retrieve the compliance mode flags. This is an internal error. **User action:** Contact IBM Service.
97A (2426)	The operation is not allowed in imprint mode. **User action**: Remove the coprocessor out of imprint mode and retry the operation.
97D (2429)	A CCA service was requested that, because of the compliance state of the domain, requires a signed command from a TKE. However, the request was not received in the correct format. **User action**: Use the TKE to perform the operation.
97F (2431)	An external key token has the compliance tag, but it is not allowed. **User action:** Recreate the external key token without the compliance tag.
985 (2437)	The PIN block translation is not allowed. **User action:** See Table 681 on page 1703 for the PIN block translations allowed when using compliant-tagged key tokens.
9C5 (2501)	The length of the random data is invalid. **User action:** Select a valid length for the random data.
9C6 (2502)	The length of the additional derivation data is invalid. **User action:** Select and define additional derivation data which is valid.
9C7 (2503)	The length of the derivation data is invalid. **User action:** Select a valid length for the derivation data.
9C9 (2505)	The length of the key type vector is invalid. **User action:** Select a valid length for the key type vector.
9CA (2506)	The PIN changes request failed authentication. **User action:** Correct the PAN authentication data.
9CB (2507)	The key type vector contains invalid values. **User action:** Correct the key type vector.
9CD (2509)	The length of the PAN data is invalid. **User action:** Correct the PAN data length.
9CE (2510)	The input tweak length for format FF2 or FF2.1 exceeds the maximum allowed, as calculated by (length * log2(tweak_alphabet_length)) <= (15 - 2) * 8.
9CF (2511)	The input plaintext or ciphertext length for format FF2 or FF2.1 exceeds the maximum allowed, as calculated by (length * log2(alphabet_length)) / 2 <= (15 - 1) * 8.
9D1 (2513)	Duplicate data found in a parameter value. For example, the alphabet for to a FFX service has duplicate characters. **User action:** Correct the parameter value.
9D2 (2514)	An error was found in the ISO PIN block format. The specific error is not noted. **User action:** Examine the PIN profile, PAN data, and other input data to ensure the inputs are correct.
B21 (2849)	A keyword was passed in the *service_data* parameter of Key Token Build2 service and it is not a valid keyword for the service. **User action**: Correct the keywords in the *service_data* parameter.
B22 (2850)	The combination of keywords in the *service_data* parameter of the Key Token Build2 service is not valid. **User action**: Check the keywords allowed for the key type being derived and correct the *service_data* parameter.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
B23 (2851)	The *service_data_length* parameter of the Key Token Build2 service does not have a valid value. **User action**: The length must be a multiple of 8 and the keywords in the *service_data* parameter must be left-justified and padded with blanks.
B81 (2945)	A required keyword for the key type being derived is not in the *service_data* parameter of the Key Token Build2 service. **User action**: Review the keywords for the key type being derived and supply all required keywords.
B82 (2946)	The maximum amount of plaintext/ciphertext that can be processed in the GCM mode by the CSNBSAD and CSNBSAE services was exceeded.
B83 (2947)	When deciphering ciphertext that had been created using Galois/Counter Mode (GCM) with the CSNBSAD service, the GCM tag provided did not match the data provided. No cleartext was returned. **User action**: Verify that the parameters provided (ciphertext, additional authenticated data, and tag) match those provided to, or returned from, the corresponding call to the CSNBSAE service.
BB9 (3001)	SET Block Decompose service was called with an encrypted OAEP block with a block contents identifier that indicates a PIN block is present. No PIN encrypting key was supplied to process the PIN block. The block contents identifier is returned in the *block_contents_identifier* parameter. **User action**: Supply a PIN encrypting key and resubmit the job.
BBB (3003)	An output parameter is too short to hold the output of the request. The length parameter for the output parameter has been updated with the required length for the request. **User action**: Update the size of the output parameter and length specified in the length field and resubmit the request.
BBC (3004)	A request was made to the Clear PIN generate or Encrypted PIN verify callable service, and the *PIN_length* parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive. **User action**: Correct the value in the *PIN_length* parameter to be within the valid range from 4 to 16. **REASONCODES**: TSS 064 (100)
BBE (3006)	The UDX verb in the coprocessor is not authorized to be executed.
BC0 (3008)	A request was made to the Clear PIN generate, Clear PIN generate alternate, or Encrypted PIN verify callable service, and the *PIN_check_length* parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive. **User action**: Correct the value in the *PIN_check_length* parameter to be within the valid range from 4 to 16. **REASONCODES**: TSS 065 (101)
BC1 (3009)	For PKCS #11 attribute processing, an attribute has been specified in the template that is not consistent with another attribute of the object being created or updated. **User action**: Correct the template for the object.
BC3 (3011)	The CRT value (p, q, Dp, Dq or U) is longer than the length allowed by the parameter block for clear key processing on an accelerator. A modulus whose length is less than or equal to 1024 bits is 64 bytes in length. A modulus whose length is greater than 1024 bits but less than or equal to 2048 bits is 128 bytes in length. **User action**: Reconfigure the accelerator as a coprocessor to make use of the key (if the CRT value is not in error and there is no coprocessor installed). **REASONCODES**: TSS 065 (101)
BC4 (3012)	A request was made to the Clear PIN generate, Clear PIN generate alternate, Encrypted PIN generate, or Encrypted PIN verify callable service to generate a VISA-PVV PIN, and the *trans_sec_parm* field has a value outside the valid range. The field being checked in the *trans_sec_parm* is the key index, in the 12th byte. This *trans_sec_parm* field is part of the *data_array* parameter. **User action**: Correct the value in the key index, held within the *trans_sec_parm* field in the *data_array* parameter, to hold a number from the valid range. **REASONCODES**: TSS 069 (105)
BC5 (3013)	The AES clear key value LRC in the token failed validation. **User action**: Correct the AES clear key value.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
BC8 (3016)	A request was made to the Encrypted PIN Translate or the Encrypted PIN verify callable service and the PIN block value or the *input_PIN_profile* or *output_PIN_profile* parameter has a value or contents that are not valid. This could be due to: • The PIN profile is not valid. • The PAD digit field is not valid. • The PIN block has been inadvertantly changed. • The IPINENC or PINVER key value is not the same as was used to create the encrypted PIN block. **User action**: Correct the PIN block value, PIN profile, or key identifier, as appropriate. **REASONCODES**: TSS 06A (106)
BCB (3019)	The call to insert or delete a z/OS PKCS #11 token object failed because the token was not found in the TKDS or a request to delete a PKCS #11 session object failed because the token was not found.
BCC (3020)	For a PKCS #11 callable service, the PKCS #11 object specified is the incorrect class for the request. **User action**: Specify the correct class of object for the service.
BCD (3021)	The call to add a z/OS PKCS #11 token failed because the token already exists in the TKDS or a request to add a z/OS PKCS #11 token object failed because an object with the same handle already exists.
BCE (3022)	The call to add or update a z/OS PKCS #11 token object failed because the supplied attributes are too large to be stored in the TKDS.
BD0 (3024)	A request was made to the Encrypted PIN Translate callable service and the format control value in the *input_PIN_profile* or *output_PIN_profile* parameter has a value that is not valid. The only valid value is NONE. **User action**: Correct the format control value to NONE. **REASONCODES**: TSS 06B (107)
BD1 (3025)	The call to create a list of z/OS PKCS #11 tokens, a list of objects of a z/OS PKCS #11 token, the information for a z/OS PKCS #11 token or the attributes of a PKCS #11 object failed because the length of the output field was insufficient to hold the data. The length field has been updated with the length of a single list or entry, token information or object attributes.
BD2 (3026)	The z/OS PKCS #11 token or object handle syntax is invalid.
BD3 (3027)	The call to read or update a z/OS PKCS #11 token or token object failed because the token or object was not found in the TKDS or the call to read or update a PKCS #11 session object failed because the object was not found.
BD4 (3028)	A request was made to the Clear PIN generate callable service. The clear_PIN supplied as part of the *data_array* parameter for an GBP-PINO request begins with a zero (0). This value is not valid. **User action**: Correct the clear_PIN value. **REASONCODES**: TSS 074 (116)
BD5 (3029)	For PKCS #11 attribute processing, an invalid attribute was specified in the template. The attribute is neither a PKCS #11 or vendor-specified attribute supported by this implementation of PKCS #11. **User action**: Correct the template by removing the invalid attribute or changing the attribute to a valid attribute.
BD6 (3030)	An invalid value was specified for a particular PKCS #11 attribute in a template when creating or updating an object.
BD7 (3031)	The certificate specified in creating a PKCS #11 certificate object was not properly encoded.
BD9 (3033)	The attribute template for creating or updating a PKCS #11 object was incomplete. Required attributes for the object class were not specified in the template.
BDA (3034)	The call to modify PKCS #11 object attributes failed because the CKA_MODIFIABLE attribute was set to false when the object was re-created.
BDB (3035)	For PKCS #11 attribute processing, an attribute was specified in the template which cannot be set or updated by the application. See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for a definition of attributes that can be set or updated by the application. **User action**: Remove the offending attribute from the template.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
BDC (3036)	A request was made to the Encrypted PIN Translate callable service. The *sequence_number* parameter was required, but was not the integer value 99999. **User action**: Specify the integer value 99999. **REASONCODES**: TSS 06F (111)
BDE (3038)	For a PKCS #11 callable service, the attributes of the PKCS #11 object specified do not permit the requested function. **User action**: Specify an object that permits the requested function.
BDF (3039)	For a PKCS #11 callable service, where a PKCS #11 key object is required, the specified object is not of the correct key type for the requested function. **User action**: Specify an object that is the correct class of key.
BE0 (3040)	The PAN, expiration date, service code, decimalization table data, validation data, or pad data is not numeric (X'F0' through X'F9'). The parameter must be character representations of numerics or hexadecimal data. **User action**: Review the numeric parameters or fields required in the service that you called and change to the format and values required. **REASONCODES**: TSS 028 (040), TSS 02A (042), TSS 066 (102), TSS 067 (103), TSS 068 (104), TSS 069 (105), TSS 06E (110)
BE1 (3041)	PKCS #11 wrap key callable service failed because the wrapping key object is not of the correct class to wrap the key specified to be wrapped. **User action**: Specify a wrapping key object of the correct class to wrap the key object.
BE3 (3043)	PKCS #11 wrap key callable service failed because the key object to be wrapped does not exist or the key class does not match the wrapping mechanism. **User action**: Specify an existing key object that is correct for the wrapping mechanism.
BE4 (3044)	A PKCS #11 session data space is full. The request to create or update an object failed and the object was not created or updated. **User action**: Delete unused session objects and cryptographic state objects from incomplete chained operations to create space for new or updated objects.
BE5 (3045)	PKCS #11 wrap key callable service failed because the key object to be wrapped has CKA_EXTRACTABLE set to false. **User action**: Specify another key object that can be extracted.
BE6 (3046)	A key token was passed to a service using high performance encrypted key operations and RACF failed your request to use the key token. **User action**: Contact your ICSF or RACF administrator if you need to pass key tokens to a service using high performance encrypted key operations.
BE7 (3047)	A clear key was provided when a secure key was required. **User action**: Correct the appropriate key identifier.
BEA (3050)	A caller is attempting to overwrite one token type with another (for example, AES over DES).
BEC (3052)	A clear key token was supplied to a service where a secure token is required.
BED (3053)	A service was called with no parameter list, but a parameter list was expected. **User action**: Call the service with a parameter list.
BEE (3054)	A request was made to a callable service with a key token wrapped with the enhanced X9.24 CBC method. Tokens wrapped with the enhanced method are not supported by this release of ICSF. **User action**: Contact your ICSF administrator to resolve which key token is to be used.
BF3 (3059)	The provided *key_identifier* refers to an encrypted variable-length CCA key token or a key label of an encrypted variable-length CCA key token. The key-management field in the CCA token does not allow its use in high performance encrypted key operations. **User action**: Supply a key token or the label of a key token with the required key-management settings.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
BF5 (3061)	The provided asymmetric key identifier cannot be used for the requested function. PKA Key Management Extensions have been enabled by a CSF.PKAEXTNS.ENABLE profile in the XFACILIT class. A CSFKEYS profile covering the key includes an ICSF segment, and the ASYMUSAGE field of that segment restricts the key from being used for the specified function. An SMF type 82 subtype 27 record is logged in the SMF database.
BF6 (3062)	The provided symmetric key identifier cannot be exported using the provided asymmetric key identifier. PKA Key Management Extensions have been enabled by a CSF.PKAEXTNS.ENABLE profile in the XFACILIT class. A CSFKEYS or XCSFKEY profile covering the symmetric key includes an ICSF segment and the SYMEXPORTABLE field of that segment places restrictions on how the key can be exported. The SYMEXPORTABLE field either specifies BYNONE, or else specifies BYLIST but the provided asymmetric key identifier is not one of those permitted to export the symmetric key (as identified by the SYMEXPORTCERTS or SYMEXPORTKEYS fields). An SMF type 82 subtype 27 record is logged to the SMF database.
BF7 (3063)	ICSF key store policy checking is active. The request failed the ICSF token policy check because the caller is not authorized to the label for the token in the key data set (CKDS or PKDS). The request is not allowed to continue because the token check policy is in FAIL mode. SMF type 82 subtype 25 records are logged in the SMF dataset. An SMF type 80 with event code qualifier of ACCESS is logged. The policy is defined by the CSF.CKDS.TOKEN.CHECK.LABEL.FAIL resource or the CSF.PKDS.TOKEN.CHECK.LABEL.FAIL resource in the XFACILIT class.
BF8 (3064)	ICSF key store policy checking is active. The specified token does not exist in the key data set (CKDS or PKDS as appropriate). The CSF-CKDS-DEFAULT or CSF-PKDS-DEFAULT resource in the CSFKEYS class is either not defined or the caller is not authorized to the CSF-CKDS-DEFAULT or CSF-PKDS-DEFAULT resource. The resource is not in WARNING mode, so the request is not allowed to continue. An SMF type 80 record with event qualifier ACCESS is logged indicating the request failed. The policy is defined by the CSF.CKDS.TOKEN.CHECK.DEFAULT.LABEL or the CSF.PKDS.TOKEN.CHECK.DEFAULT.LABEL resource in the XFACILIT class.
BF9 (3065)	ICSF token policy checking is active. The caller is requesting to add a token to the key data set (CKDS or PKDS as appropriate) that already exists within the key data set. The request fails. The policy is defined by the CSF.CKDS.TOKEN.NODUPLICATES resource or the CSF.PKDS.TOKEN.NODUPLICATES resource in the XFACILIT class.
BFB (3067)	The provided *key_identifier* refers to an encrypted CCA key token or a key label of an encrypted CCA key token, and the CSFKEYS profile covering it does not allow its use in high performance encrypted key operations. **User action:** Contact your ICSF or RACF administrator if you need to use this key with an ICSF service that supports secure keys for CPACF. For more details, see 'Enabling use of encrypted keys in callable services that exploit CPACF' in *z/OS Cryptographic Services ICSF Administrator's Guide*.
BFC (3068)	A cryptographic operation using a specific PKCS #11 key object is being requested. The key object has exceeded its useful life for the operation requested. The request is not processed. **User action**: Use a different key.
BFE (3070)	A cryptographic operation that requires FIPS 140-2 compliance is being requested. The desired algorithm, mode, or key size is not approved for FIPS 140-2. The request is not processed. **User action:** Repeat the request using an algorithm, mode, and/or key size approved for FIPS 140-2. Refer to *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for this list of approved algorithms, modes, and key sizes.
BFF (3071)	An application using a z/OS PKCS #11 token that is marked 'Write Protected' is attempting to do one of the following: • Store a persistent object in the token. • Delete the token. • Reinitialize the token. ICSF always marks the session object only omnipresent token as 'Write Protected.' ICSF will also mark an ordinary token 'Write Protected' if it contains objects not supported by this release of ICSF. **User action:** Use a z/OS PKCS #11 token that is not marked 'Read Only' or, if this is an ordinary token (not the omnipresent token), attempt the delete or reinitialization from a different member of the sysplex.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
C04 (3076)	The provided symmetric key label refers to an encrypted CCA key token, and the CSFKEYS profile covering it does not allow it to be returned in its protected-key CPACF form. **User Action:** Contact your ICSF or RACF administrator if you need to use this label in calls to the CKDS Key Record Read2 service with the PROTKEY rule. For information about the SYMCPACFRET field of the ICSF segment, see 'Enabling use of encrypted keys in callable services that exploit CPACF' in *z/OS Cryptographic Services ICSF Administrator's Guide*.
C07 (3079)	A request was made to use a key token wrapped with the X9.24 enhanced wrapping method introduced in HCR7780. Key tokens wrapped with the enhanced method cannot be used on this release. Also, key tokens wrapped with the enhanced method cannot be updated or deleted from the CKDS on this release. **User Action:** Run your application on a release that support the enhanced wrapping method.
C08 (3080)	The use of a PKA key token has been attempted. The token is not supported on the release of ICSF currently running. **User Action:** Check the ICSF release for support of this token type.
C0B (3083)	The specified key token buffer length is of insufficient size for the buffer to contain the output key token. **User action:** Specify a key token buffer that is sufficiently large enough to receive the output key token.
C0C (3084)	The key token associated with the specified key label is a variable-length token, which is not compatible with this callable service. **User action:** Either modify the program logic to utilize a key label that is associated with a compatible key token or use an ICSF callable service that supports the symmetric key token type provided.
C0D (3085)	Rule array keyword specifies a function not supported by this hardware. Some examples include: • ECC specified in rule array for the PKA Key Token Change callable service, but request is being executed on a system that does not support ECC keys. • PROTKEY specified in rule array for the CKDS Key Record Read2 callable service against a clear key label, but request is being executed on a system that does not have CP Assist for Cryptographic Functions. • PROTKEY specified in rule array for the CKDS Key Record Read2 against a secure key label, but request is being executed on a system that either does not have a cryptographic coprocessor or does not have one with a sufficient level of licensed internal code (LIC). **User Action:** Specify a different, supported, rule array keyword or execute the service on a system that supports the function.
C0E (3086)	Specified token is not supported by this hardware. For example, an ECC token is being used but request is being executed on a system that does not support ECC keys. **User Action:** Specify a different, supported, token, or execute the request on a system that supports the function.
C0F (3087)	A coordinated KDS refresh was attempted to an empty KDS. The new KDS of a coordinated KDS refresh must be initialized and must contain the same MKVP values as the active KDS. **User action:** Perform a coordinated KDS refresh using a new KDS that is initialized and that contains the same MKVP values as the active KDS.
C10 (3088)	A coordinated KDS change master key was attempted and either the new KDS or backup KDS contained a different LRECL attribute from the active KDS. The new KDS and optionally the backup KDS must contain the same LRECL attribute as the active KDS during a coordinate KDS change master key. **User action:** Perform a coordinated KDS change master key using a new KDS and optionally a backup KDS with the same LRECL attribute as the active KDS.
C11 (3089)	The new KDS specified for a coordinated KDS change master key was not empty when the operation began. The new KDS must be empty before performing a coordinated KDS change master key. **User action:** Perform the coordinated KDS change master key with a new KDS that is empty.
C12 (3090)	The backup KDS specified for a coordinated KDS change master key was not empty when the operation began. When using the optional backup function, the backup KDS must be empty before performing a coordinated KDS change master key. **User action:** Perform the coordinated KDS change master key with a backup KDS that is empty.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
C13 (3091)	The new KDS specified for a coordinated KDS refresh contains different MKVPs than the active KDS. In order to perform a coordinated KDS refresh, the new KDS specified must contain the same MKVPs as the active KDS. **User action:** Perform the coordinated KDS refresh with a new KDS that contains the same MKVPs as the active KDS.
C14 (3092)	The system that is trying to do the CCMK has rejected update requests for higher version records, so the in-store KDS is incomplete and cannot be used for CCMK. **User action:** Retry the function from a sysplex KDS cluster member running the highest ICSF FMID level.
C1F (3103)	The new KDS specified for either a coordinated KDS refresh or coordinated KDS change master key is not a valid data set name. **User action:** Specify a valid data set name for the new KDS when performing either a coordinated KDS refresh or coordinated KDS change master key.
C20 (3104)	The backup KDS specified for a coordinated KDS change master key is not a valid data set name. **User action:** Specify a valid data set name for the backup KDS when performing a coordinated KDS change master key.
C21 (3105)	A coordinated KDS refresh or coordinated KDS change master key was attempted while at least one ICSF instance in the sysplex was below the HCR7790 FMID level. The coordinated KDS refresh and coordinated KDS change master key functions are only available when all ICSF instances in the sysplex, regardless of active KDS, are running at the HCR7790 FMID level or higher. **User action:** Remove or upgrade ICSF instances in the sysplex that are running below the HCR7790 FMID level and retry the function.
C22 (3106)	Either a coordinated KDS refresh or coordinated KDS change master key was attempted while another coordinated KDS refresh or coordinated KDS change master key was still in progress. The coordinated KDS function was initiated by this ICSF instance. Only one coordinated KDS function may execute at a time in the sysplex. **User action:** Wait for the previous coordinated KDS function to complete and retry the function.
C23 (3107)	A coordinated KDS change master key was attempted using a new KDS with the same name as the active KDS. The new KDS name must be different from the active KDS when performing a coordinated KDS change master key. **User action:** Specify a new KDS with a different name from the active KDS and retry the function. Coordinated KDS change master key requires the new KDS to be allocated and match the same VSAM attributes as the active KDS.
C24 (3108)	A coordinated KDS change master key was attempted using a backup KDS with the same name as the active KDS. When using the backup function, the backup KDS name must be different from the active KDS when performing a coordinated KDS change master key. **User action:** Specify a backup KDS with a different name from the active KDS and retry the function. Coordinated KDS change master key requires the backup KDS to be allocated and match the same VSAM attributes as the active KDS.
C25 (3109)	A coordinated KDS change master key was attempted using a new KDS with the same name as the backup KDS. If a backup KDS is specified, its name must be different from the new KDS. **User action:** Specify a backup KDS with a different name from the new KDS and retry the function. The backup KDS is optional. Coordinated KDS change master key requires the new KDS, and optionally the backup KDS, to be allocated and match the same VSAM attributes as the active KDS.
C26 (3110)	A coordinated KDS refresh or coordinated KDS change master key was attempted using an archive KDS name that is not valid. **User action:** Specify a valid data set name for the archive KDS and retry the function. The archive data set name is optional. The optional archive KDS name must not exist on the system prior to performing a coordinated KDS refresh or a coordinated KDS change master key.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
C27 (3111)	A coordinated KDS change master key was attempted using an archive KDS with the same name as the backup KDS. When using the archive and backup functions, the archive KDS name must be different from the backup KDS. **User action:** Specify an archive KDS with a different name from the backup KDS and retry the function. The archive KDS name and the backup KDS are optional. The archive KDS name must not exist on the system prior to performing a coordinated KDS refresh or a coordinated KDS change master key. The backup KDS must be allocated and match the same VSAM attributes as the active KDS.
C28 (3112)	A coordinated KDS refresh or a coordinated KDS change master key was attempted using an archive KDS with the same name as the active KDS. When using the archive function, the archive KDS name must be different from the active KDS. **User action:** Specify an archive KDS with a different name from the active KDS and retry the function. The archive KDS name must not exist on the system prior to performing a coordinated KDS refresh or a coordinated KDS change master key.
C29 (3113)	A coordinated KDS refresh or a coordinated KDS change master key was attempted using an archive KDS with the same name as the new KDS. When using the archive function, the archive KDS name must be different from the new KDS. **User action:** Specify an archive KDS with a different name than the new KDS and retry the function. The archive KDS name must not exist on the system prior to performing a coordinated KDS refresh or a coordinated KDS change master key.
C2A (3114)	Either a coordinated KDS refresh or coordinated KDS change master key was attempted while another coordinated KDS refresh or coordinated KDS change master key was still in progress. The coordinated KDS function was initiated by another ICSF instance in the sysplex. Only one coordinated KDS function may execute at a time in the sysplex. **User action:** Wait for the previous coordinated KDS function to complete and retry the function.
C30 (3120)	A coordinated KDS change master key was attempted on an active KDS that was not initialized. The active KDS must be initialized before performing a coordinated KDS change master key. **User action:** Initialize the active KDS and retry the function
C31 (3121)	The archive option was specified for a coordinated KDS refresh of the active KDS. The archive option is only valid for coordinated KDS refreshes to a new KDS or coordinated KDS change master key. **User action:** Do not specify an archive data set when performing a coordinated KDS refresh of the active KDS.
C3C (3132)	The archive data set name specified for coordinated KDS refresh or coordinated KDS change master key is too long. The archive data set name must allow enough space for renaming the KDS VSAM data and index portions within 44 characters. **User action:** Specify a shorter name for the archive data set name to allow enough space for renaming the KDS VSAM data and index portions within 44 characters. The archive data set name is optional. When specified, the archive data set name must not exist on the system prior to performing the coordinated KDS function.
C3D (3133)	During a coordinated KDS refresh or coordinated KDS change master key with the archive option specified, the active KDS could not be renamed to the archive data set name. This failure occurred because the active KDS VSAM data and index suffix names were not valid for performing the rename. **User action:** Consider alternate names for the active KDS VSAM data and index suffixes. The archive data set name is optional. When specified the archive data set name must not exist on the system prior to performing the coordinated KDS function.
C3E (3134)	A coordinated KDS change master key attempted to use a new KDS that is currently another sysplex members active KDS. Performing a coordinated KDS change master key to another sysplex member's active KDS is not allowed as it would alter all sysplex members configured in that sysplex KDS group. **User action:** Specify a new KDS that is not currently the active KDS of another sysplex member and retry the function.
C3F (3135)	A coordinated KDS conversion was attempted against a KDS that was already in common record format (KDSR). **User action:** Specify a KDS that is not already in common record format (KDSR) and retry the function.
C5B (3163)	The supplied clear key value has replicated key parts. A rule array keyword or control vector in the supplied key token require that all key parts be unique. **User action:** Supply a key value that has unique key parts.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
C81 (3201)	Operation requested requires a clear key, but a secure key was supplied. **User action:** Use a different key, one that is clear.
CE8 (3304)	There is a mismatch between the key data set specified in the rule array and a search criteria for the KDS list service. The key data set must be CKDS when the criteria is CKDS type. The key data set must be TKDS when the criteria is TKDS type. The key data set must be CKDS or PKDS when the criteria is unsupported CCA key. The key data set must be PKDS when the criteria is a weak CCA key. **User action:** Specify the correct key data set in the rule array.
CE9 (3305)	The metadata type in a structure in the metadata list for the KDS list service is zero and not allowed. **User action:** Specify a valid metadata tag.
CEA (3306)	A criterion flag in the search criteria for the KDS list service was not valid.
CEC (3308)	The length of the handle for a TKDS token for the KDS list service was not correct. **User action:** Specify a valid token handle and a length of 32.
CED (3309)	Output area specified for the KDS list and the KDS metadata read services is too small to contain the requested data. For the KDS metadata read service, the output is restricted to 1000 bytes. **User action:** Increase the size of the output area and specify the new size.
CEE (3310)	For the KDS list service, the continuation area contains inconsistent data. It must be binary zero for the initial call and be returned unchanged for subsequent calls. **User action:** Check that the continuation area is correct and not being changed for subsequent calls.
CEF (3311)	The search criteria length specified for the KDS list service is greater than 500 bytes. **User action:** Correct the length of the search criteria.
CF0 (3312)	A search criteria specified for the KDS list service was in an incorrect format. **User action:** Correct the search criteria.
CF1 (3313)	The search criterion in a search criteria structure was not recognized for the KDS list service. **User action:** Correct the search criteria.
CF2 (3314)	The length field in a search criteria structure was incorrect for the KDS list service. **User action:** Correct the search criteria.
CF3 (3315)	The PKCS #11 token name specified in the label filter for the KDS list service was not found in the TKDS.
CF5 (3317)	The date type in a search criteria for the KDS list service was not recognized. **User action:** Correct the search criteria.
CF6 (3318)	The comparison operator in a search criteria for the KDS list service was not recognized. **User action:** Correct the search criteria.
CF7 (3319)	A reserved length parameter was not zero. **User action:** Specify a length of zero for the reserved length parameters.
CF8 (3320)	The label filter for the KDS list service was not syntactically correct. **User action:** Correct the label filter.
CF9 (3321)	The label filter length for the KDS list service was too long. **User action:** Correct the label filter.
CFA (3322)	The TKDS object type in the search criteria for the KDS list service is incorrect. **User action:** Correct the search criteria.
CFB (3323)	The CKDS key type in the search criteria for the KDS list service is incorrect. **User action:** Correct the search criteria or rule array.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
CFC (3324)	The key type bit specified in the structure for the unsupported or weak CCA keys search criterion for the KDS list service is not valid. **User action:** Specify a valid metadata tag.
D03 (3331)	The metadata type in a structure in the metadata list for the KDS metadata write service is read only. The metadata block specified cannot be changed. **User action:** Remove the metadata tag that is read only.
D04 (3332)	The IBM variable-length metadata blocks are read only. The metadata blocks cannot be changed. **User action:** Remove the IBM variable metadata block from the metadata list.
D05 (3333)	A date in a structure in the action area for the KDS metadata write service is incorrect. **User action:** Correct the date.
D06 (3334)	The metadata list for the KDS metadata write service is incomplete. The metadata list length parameter does not match the sum of the lengths of the structures in the metadata list. **User action:** Correct the action area and length parameters.
D07 (3335)	The object handle specified for the KDS metadata read and KDS metadata write services for the TKDS is not the handle of a token object. **User action:** Only token objects have metadata. Tokens and session objects cannot have metadata.
D08 (3336)	The value specified for the input metadata length for the KDS metadata read and KDS metadata write services is incorrect. The value is either not large enough to contain valid date or is too large for the service. **User action:** Check the input metadata length and the metadata area.
D09 (3337)	The format of the input metadata for the KDS metadata read and KDS metadata write services is incorrect. **User action:** Check the format of the input metadata structure.
D0A (3338)	A data type in the input metadata for the KDS metadata read and KDS metadata write services is not recognized. **User action:** Check the contents of the input metadata structure.
D0B (3339)	A block in the input metadata area has a length specified that is inconsistent for the metadata type. **User action:** Check the contents of the input metadata structure.
D0C (3340)	The variable-length installation metadata in the input metadata area for the KDS metadata write service cannot be written to the record because the total limit of installation metadata would be exceeded. **User action:** Check the contents of the input metadata structure.
D0E (3342)	A service passed the label of a KDS record which is not yet active. The key material validity start date is in the future. The key material of the record is not available. **User action:** Determine if the KDS label is correct. If so, contact the ICSF administrator and determine if the record should be made active.
D0F (3343)	A service passed the label of a deactivated KDS record. The key material validity end date has passed. The key material of the record is not available. **User action:** Determine if the KDS label is correct. If so, contact the ICSF administrator and determine if the record should be made active.
D10 (3344)	A service passed the label of an archived KDS record. The key material of the record is not available. **User action:** Determine if the KDS label is correct. If so, contact the ICSF administrator and determine if the record should be recalled.
D11 (3345)	The value of a metadata flag for the KDS metadata write service or the KDS list service is incorrect. **User action:** Supply a proper value.
D20 (3360)	The KDS multi-Purpose callable service is in use. **User action:** Try again later.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
D42 (3394)	ARPC generation failed. **User action:** Ensure that the correct key mode was specified and the issuer master key being used is valid for ARPC generation.
D43 (3395)	Secure messaging with integrity failure. **User action:** Ensure that the correct key mode was specified and the issuer master key being used is valid for EMV scripting with integrity.
D44 (3396)	Secure messaging with confidentiality failure. **User action:** Ensure that the correct key mode was specified and the issuer master key being used is valid for EMV scripting with confidentiality.
D47 (3399)	Failure to decrypt the encrypted counter. **User action:** Ensure that a valid encrypted counter was passed and the correct issuer master key was used.
D5E (3422)	A service passed the label of an archived key. The service requested was not a data decryption service and the CSF.KDS.KEY.ARCHIVE.DATA.DECRYPT control is enabled. The control disallows the use of an archived key in data encryption services. **User action:** Determine if the label is correct. Determine if the service requested is correct. If so, contact your ICSF administrator for guidance.
D60 (3424)	The Key Record Write service checked the control vector of a key-encrypting key that has the NOCV bit on. The control vector was found not be to be the default control vector for an DES IMPORTER or EXPORTER key. **User action:** Rebuild the key token with the default control vector. The key form bits may indicate any valid length except single-length key. The ENH-ONLY bit may be enabled.
DAE (3502)	The Options Data Set Refresh function completed. No changes were made. **User action:** Check the ICSF joblog.
DB1 (3505)	The Options Data Set Refresh function ended. A syntax error was encountered in the options data set. **User action:** Check the ICSF joblog. Correct the syntax of the option parameters in the options data set. Re-run the Options Data Set Refresh function.
DB2 (3506)	The Options Data Set Refresh function ended. An error was encountered with the data set.
DB4 (3508)	The Options Data Set Refresh function ended. An error was encountered while attempting to allocate the options data set.
DB5 (3509)	The Options Data Set Refresh function ended. An error was encountered while retrieving the option information.
DC0 (3520)	The KDS name passed as input is not the active KDS for the KDS type specified. **User action:** The CSFKDU service only supports the updating of the active KDS in use by ICSF. Update your program to pass the active KDS name.
DC1 (3521)	The length of the original record is larger than the largest record size supported for the specified KDS type. **User action:** Only pass valid records to the service.
DC2 (3522)	The length of the new record is larger than the largest record size supported for the specified KDS type. **User action:** Only pass valid records to the service.
DC3 (3523)	The function code passed to the CSFKDU service is not a valid function. **User action:** Pass only documented function values.
DC4 (3524)	A request to create, update, or delete a record has failed because the state of the current record does not match what was passed as the original state. **User action:** Update your program to pass an original state of the record that matches what is in the KDS.
DC5 (3525)	A request to update a record has failed because the before and after labels are different. **User action:** Ensure that the before and after record labels match exactly.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
DC6 (3526)	The input parameter with option flags has flags turned on that are not currently supported. **User action:** Clear the option flags field and only set supported option flags.
DC8 (3528)	For a TKDS update, the record passed is at a higher version than what the current release of ICSF supports. **User action:** Only attempt to create new records at a version that is supported by the ICSF release you are running. Optionally, ask your system programmer to install a more recent level of ICSF that supports the TKDS version desired.
DC9 (3529)	A key identifier was supplied to a callable service as a key token or the label of a key token in a key data set. Either the key type of the key or the algorithm of the key is unsupported by the cryptographic features available to ICSF. **User action:** Supply a key identifier supported by the cryptographic features.
DCA (3530)	The record type is not valid. **User action:** Ensure that the record has a valid KDS type and object type, if appropriate.
DCB (3531)	The CSFKDU service encountered an unexpected error while trying to create a TKDS record. This is most commonly caused by attempting to create an object for which no token record exists.
DCF (3535)	The cryptographic usage statistic to be updated by CSFSTAT is not enabled. **User action:** Enable the cryptographic usage statistic for tracking. For more details, see *z/OS Cryptographic Services ICSF Administrator's Guide*.
DD2 (3538)	The operation failed because an attempt was made to use or manage a compliant-tagged key token which is not supported on this system. **User action:** Retry the operation on a system that supports the compliant-tagged token being used.
DD6 (3542)	The value specified in the *input_PAN_data_length*, the *PAN_data_length*, or the *reference_PAN_data_length* parameter is not valid. **User action:** Correct the *input_PAN_data_length* or the *PAN_data_length* parameter.
DD7 (3543)	The value specified in the *output_PAN_data_length* is not valid. **User action:** Correct the *output_PAN_data_length* parameter.
DD8 (3544)	The value specified in the *input_PIN_profile_length* or the *reference_PIN_profile_length* parameter is not valid. **User action:** Correct the *input_PIN_profile_length* or the *reference_PIN_profile_length* parameter.
DD9 (3545)	The value specified in the *output_PIN_profile_length* is not valid. **User action:** Correct the *output_PIN_profile_length* parameter.
F9E (3998)	On a call to PCI Interface Callable Service, TKE sent a request to a specific PCI card queue using domain index 0 which is not one of the control domain indices listed in the LPAR activation profile. This occurs when using an older TKE workstation with a newer machine. **User action:** Use the level of TKE workstation that is required when ordering the newer machine or mark domain 0 as a control domain in the LPAR activation profile.
F9F (3999)	On a call to CKDS Key Record Delete or CKDS Key Record Write2, the label refers to a Variable-length Symmetric key token with an unrecognized algorithm or key type in the associated data section. Only key tokens with a recognized algorithm or key type can be managed on this release of ICSF. **User action:** Call CKDS Key Record Delete or CKDS Key Record Write2 on a release of ICSF which recognizes the algorithm and key type of this token.
FA0 (4000)	The encipher and decipher callable services sometime require text (plaintext or ciphertext) to have a length that is an exact multiple of 8 bytes. Padding schemes always create ciphertext with a length that is an exact multiple of 8. If you want to decipher ciphertext that was produced by a padding scheme, and the text length is not an exact multiple of 8, then an error has occurred. The CBC mode of enciphering requires a text length that is an exact multiple of 8. **User action**: Review the requirements of the service you are using. Either adjust the text you are processing or use another process rule. **REASONCODES**: TSS 033 (051)

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
1391 (5009)	The provided TR-31 Optional Block contains data that cannot be used by the HSM. **User action**: Use supported optional block data.
177F (6015)	An ECC curve type is invalid, its usage is inconsistent, or the required hardware level is not available. **User action:** Supply a valid ECC curve type.
1782 (6018)	One or more of the parameters passed to this callable service are in error. **User action:** Refer to the parameter descriptions in this publication under the appropriate callable service to ensure the parameter values specified by your application are valid.
2710 (10000)	A key identifier was passed to a service or token. It is checked in detail to ensure that it is a valid token, and that the fields within it are valid values. There is a token validation value (TVV) in the token, which is a non-cryptographic value. This value was again computed from the rest of the token, and compared to the stored TVV. If these two values are not the same, this reason code is returned. **User action**: The contents of the token have been altered because it was created by ICSF or TSS. Review your program to see how this could have been caused. **REASONCODES**: TSS 0C (12) and 1D (29)
2714 (10004)	A key identifier was passed to a service. The master key verification pattern in the token shows that the key was created with a master key that is neither the current master key nor the old master key. Therefore, it cannot be reenciphered to the current master key. **User action**: Re-import the key from its importable form (if you have it in this form), or repeat the process you used to create the operational key form. If you cannot do one of these, you cannot repeat any previous cryptographic process that you performed with this token. **REASONCODES**: TSS 030 (048)
271C (10012)	A key label was supplied for a key identifier parameter. This label is the label of a key in the in-storage CKDS or PKDS. A key record with that label (and the specific type if required by the ICSF callable service) could not be found. For a retained key label, this error code is also returned if the key is not found in the CCA coprocessor specified in the PKDS record. **User action**: Check with your administrator if you believe that this key should be in the in-storage CKDS or the PKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label. **REASONCODES**: TSS 01E (030)
2720 (10016)	You specified a value for a *key_type* parameter that is not an ICSF-defined name. **User action**: Review the ICSF key types and use the appropriate one. **REASONCODES**: TSS 03D (061)
2724 (10020)	You specified the word TOKEN for a *key_type* parameter, but the corresponding key identifier, which implies the key type to use, has a value that is not valid in the control vector field. Therefore, a valid key type cannot be determined. **User action**: Review the value that you stored in the corresponding key identifier. Check that the value for *key_type* is obtained from the appropriate *key_identifier* parameter. **REASONCODES**: TSS 027 (039)
272C (10028)	One of the following occurred: • Either the *left* half of the control vector in a key identifier (internal or external) equates to a key type that is not valid for the service you are using or the value is not that of any ICSF control vector. For example, an exporter key-encrypting key is not valid in the key import callable service. • An attempt was made to export a non-DATA key to CPACF protected key format. The key may be a CIPHER key which does not have the XPRTCPAC bit set in the control vector. **User action**: Determine which key identifier is in error and use the key identifier that is required by the service. If this is an attempt to export a key to CPACF protected key format, either use a DATA key or a CIPHER key with the XPRTCPAC bit set in the control vector. **REASONCODES**: TSS 027 (039)

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
2730 (10032)	Either the *right* half of the control vector in a key identifier (internal or external) equates to a key type that is not valid for the service you are using, or the value is not that of any ICSF control vector. For example, an exporter key-encrypting key is not valid in the key import callable service. **User action**: Determine which key identifier is in error and use the key identifier that is required by the service. **REASONCODES**: TSS 027 (039)
2734 (10036)	Either the complete control vector (CV) in a key identifier (internal or external) equates to a key type that is not valid for the service you are using, or the value is not that of any ICSF control vector. The difference between this and reason codes 10028 and 10032 is that each half of the control vector is valid, but *as a combination*, the whole is not valid. For example, the left half of the control vector may be the importer key-encrypting key and the right half may be the input PIN-encrypting (IPINENC) key. **User action**: Determine which key identifier is in error and use the key identifier that is required by the service. **REASONCODES**: TSS 027 (039)
2738 (10040)	Key identifiers contain a version number. One of the following situations is possible: • The version number in a supplied key identifier (internal or external) is inconsistent with one or more fields in the key identifier, making the key identifier unusable. • The version number in a supplied key token, or token retrieved by a supplied label, is not consistent or not valid with another parameter you specified. For example, a DES key token (version 0 or 1) is not valid with the rule array keyword AES in the Symmetric Key Encipher callable service. **User action**: Use a token, or the label of a token, containing the required version number. **REASONCODES**: TSS 031 (049)
273C (10044)	A cross-check of the control vector the key type implies has shown that it does not correspond with the control vector present in the supplied internal key identifier. **User action**: Change either the key type or key identifier. **REASONCODES**: TSS 0B7 (183)
2740 (10048)	The *key_type* parameter does not contain one of the valid types for the service or the keyword TOKEN. **User action**: Check the supplied parameter with the ICSF key types. If you supplied the keyword TOKEN, check that you have padded it on the right with blanks. **REASONCODES**: TSS 03D (061)
2744 (10052)	A null key identifier was supplied and the *key_type* parameter contained the word TOKEN. This combination of parameters is not valid. **User action**: Use either a null key identifier or the word TOKEN, not both. **REASONCODES**: TSS 027 (039)
2748 (10056)	You called the key import callable service. The importer key-encrypting key is a NOCV importer and you specified TOKEN for the *key_type* parameter. This combination is not valid. **User action**: Specify a value in the *key_type* parameter for the operational key form.
274C (10060)	You called the key export callable service. A label was supplied in the *key_identifier* parameter for the key to be exported and the *key_type* was TOKEN. This combination is not valid because the service needs a key type in order to retrieve a key from the CKDS. **User action**: Specify the type of key to be exported in the *key_type* parameter. **REASONCODES**: TSS 03D (061)
2754 (10068)	A flag in a key identifier indicates the master key verification pattern (MKVP) is not present in an internal key token. This setting is not valid. **User action**: Use a token containing the required flag values. **REASONCODES**: TSS 02F (047)
2758 (10072)	A flag in a key identifier indicates the encrypted key is not present in an external token. This setting is not valid. **User action**: Use a token containing the required flag values. **REASONCODES**: TSS 02F (047)

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
275C (10076)	A flag in a key identifier indicates the control vector is not present. This setting is not valid. **User action**: Use a token containing the required flag values. **REASONCODES**: TSS 02F (047)
2760 (10080)	An ICSF private flag in a key identifier has been set to a value that is not valid. **User action**: Use a token containing the required flag values. Do not modify ICSF or the reserved flags for your own use.
2768 (10088)	If you supplied a label in the *key_identifier* parameter, a record with the supplied label was found in the CKDS, but the key type (CV) is not valid for the service. If you supplied an internal key token for the *key_identifier* parameter, it contained a key type that is not valid. **User action**: Check with your ICSF administrator if you believe that this key should be in the in-storage CKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label. **REASONCODES**: TSS 027 (039)
2788 (10120)	The internal key token you supplied, or the key token that was retrieved by the label you supplied, contains a flag setting or data encryption algorithm bit that is not valid for this service. **User action**: Ensure that you supply a key token, or label, for a non-ANSI key type.
278C (10124)	The key identifier you supplied cannot be exported because there is a prohibit-export restriction on the key. **User action**: Use the correct key for the service. **REASONCODES**: TSS 027 (039)
2790 (10128)	The keyword you supplied in the *rule_array* parameter is not consistent or not valid with another parameter you specified. For example, the keyword SINGLE is not valid with the key type of EXPORTER in the key token build callable service. **User action**: Correct either the *rule_array* parameter or the other parameter. **REASONCODES**: TSS 09C (156)
2791 (10129)	NOCV KEKs are not permitted in the RKX service.
2AF8 (11000)	The value specified for length parameter for a key token, key, or text field is not valid. This can also occur if either the key type of the key or the algorithm of the key is unsupported by the callable service. **User action**: Correct the appropriate length field parameter. **REASONCODES**: TSS 048 (072)
2AFC (11004)	The hash value (of the secret quantities) in the private key section of the internal token failed validation. The values in the token are corrupted. You cannot use this key. **User action**: Re-create the token using the appropriate combination of the PKA key token build, PKA key generate, and PKA key import callable services. **REASONCODES**: TSS 02F (047)
2B00 (11008)	The public or private key values are not valid (for example, the modulus or an exponent is zero or the exponent is even) or the key could not have created the signature (for example, the modulus value is less than the signature value). In any case, the key cannot be used to verify the signature. **User action**: You might need to re-create the token by using the PKA key token build or PKA key import callable service or regenerate the key values on another platform. **REASONCODES**: TSS 302 (770)
2B04 (11012)	The internal or external private key token contains flags that are not valid. **User action**: You may need to re-create the token using the PKA key token build or PKA key import callable service. **REASONCODES**: TSS 02F (047)

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
2B08 (11016)	The calculated hash of the public information in the PKA token does not match the hash in the private section of the token. The values in the token are corrupted. **User action**: Verify the public key section and the key name section of the token. If the token is still rejected, then you need to re-create the token using the appropriate combination of the PKA key token build, PKA key generate, and PKA key import callable services. **REASONCODES**: TSS 02F (047)
2B0C (11020)	The hash pattern of the master key in the supplied internal PKA private key token does not match the current system's PKA master key. This indicates the master key has changed since the token was created. You cannot use the token. **User action**: Re-create the token using the appropriate combination of the PKA key token build, PKA key generate, and PKA key import callable services. **REASONCODES**: TSS 030 (048)
2B10 (11024)	The PKA tokens have incomplete values, for example, a PKA public key token without modulus. **User action**: Re-create the key. **REASONCODES**: TSS 02F (047)
2B14 (11028)	The modulus of the PKA key is too short for processing the hash or PKCS block. **User action**: Either use a PKA key with a larger modulus size, use a hash algorithm that generates a smaller hash (digital signature services), or specify a shorter DATA key size (symmetric key export, symmetric key generate). **REASONCODES**: TSS 048 (072)
2B18 (11032)	The supplied private key can be used only for digital signature. Key management services are disallowed. **User action**: Supply a key with key management enabled. **REASONCODES**: TSS 040 (064)
2B20 (11040)	The recovered encryption block was not a valid PKCS-1.2 or zero-pad format. (The format is verified according to the recovery method specified in the rule-array.) If the recovery method specified was PKCS-1.2, refer to PKCS-1.2 for the possible error in parsing the encryption block. For the PKCS #11 services CSFPUWK and CSFPSKD, this reason could also indicate a non-RSA encryption block length problem. **User action**: Ensure that the parameters passed to CSNDSYI or CSNFSYI are correct. Possible causes for this error are incorrect values for the RSA private key or incorrect values in the *RSA_enciphered_key* parameter, which must be formatted according to PKCS-1.2 or zero-pad rules when created. **REASONCODES**: TSS 42 (66)
2B24 (11044)	The first section of a supplied PKA token was not a private or public key section. **User action**: Re-create the key. **REASONCODES**: TSS 0B5(181)
2B28 (11048)	The eyecatcher on the PKA internal private token is not valid. **User action**: Reimport the private token using the PKA key import callable service.
2B2C (11052)	An incorrect PKA token was supplied. One of the following situations is possible: • The service requires a private key token of the correct type. • The supplied token may be of a type that is not supported on this system. **User action**: Check that the supplied token is: • a PKA private key token of the correct type. • a type supported by this system.
2B30 (11056)	The input PKA token contains length fields that are not valid. **User action**: Re-create the key token.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
2B38 (11064)	The RSA-OAEP block did not verify when it decomposed. The block type is incorrect (must be X'03'). **User action**: Re-create the RSA-OAEP block. **REASONCODES**: TSS 2CF (719)
2B3C (11068)	The RSA-OAEP block did not verify when it decomposed. The verification code is not correct (must be all zeros). **User action**: Re-create the RSA-OAEP block. **REASONCODES**: TSS 2D1 (721)
2B40 (11072)	The RSA-OAEP block did not verify when it decomposed. The random number I is not correct (must be non-zero with the high-order bit equal to zero). **User action**: Re-create the RSA-OAEP block. **REASONCODES**: TSS 2D0 (720)
2B48 (11080)	The RSA public or private key specified a modulus length that is incorrect for this service. **User action**: Re-invoke the service with an RSA key with the proper modulus length. **REASONCODES**: See reason codes 41 (65) and 2F8 (760)
2B4C (11084)	This service requires an RSA public key and the key identifier specified is not a public key. **User action**: Re-invoke the service with an RSA public key.
2B50 (11088)	This service requires an RSA private key that is for signature use only. **User action**: Re-invoke the service with a supported private key.
2B54 (11092)	There was an invalid subsection in the PKA token. **User action**: Correct the PKA token.
2B58 (11096)	This service requires an RSA private key that is for signature use. The specified key may be used for key management purposes only. **User action**: Re-invoke the service with a supported private key. **REASONCODES**: TSS 040 (064)
2EE0 (12000)	You cannot use the Clear PIN Generate callable services because ICSF is not in special secure mode. **User action**: Contact your ICSF administrator (your administrator can enable the processing mode).
3E80 (16000)	RACF failed your request to use this service or PKCS #11 token. This may be caused by the CSFSERV or CRYPTOZ class. **User action**: Contact your ICSF or RACF administrator if you need this service.
3E84 (16004)	RACF failed your request to use the key label or token. This may be caused by either the CSFKEYS or XCSFKEY class, depending on the setting of the Granular Keylabel Access Controls and the type of token provided. Both key labels and the private-key name in a PKA secure private key are subject to controls implemented using the CSFKEYS class. **User action**: Contact your ICSF or RACF administrator if you need this key.
3E88 (16008)	Clear key generation denied by policy. Secure PKCS #11 services are not available and caller's RACF access to CRYPTOZ class resource CLEARKEY.token-label does not permit the generation of non-secure (clear) PKCS #11 keys. **User action:** Contact your ICSF administrator **ICSF administrator action:** Either configure ICSF for secure PKCS #11 services or have your RACF administrator grant the user authority to use clear keys
3E8C (16012)	You requested the conversion service, but you are not running in an authorized state. **User action**: You must be running in supervisor state to use the conversion service. Contact your ICSF administrator.

Table 604. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
3E90 (16016)	The input/output field contained a valid internal token with the NOCV bit on and processing failed due to one of the following reasons: • The key type was incorrect. • The key type did not match the type of the generated or imported key. • The control vector was not the default control vector. • The generated or imported token would have been wrapped with the WRAPENH3 method. **User action**: Correct the calling application. **REASONCODES**: TSS 027 (039)
3E94 (16020)	You called a service and specified the label of a CKDS system key, which is not allowed. **User action**: Correct the calling application. **REASONCODES**: TSS 0B5 (181)
3E98 (16024)	You called the CKDS key record write callable service, but the key token you supplied is not valid. **User action**: Check with your ICSF administrator if you believe that this key should be in the in-storage CKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label.
3EA0 (16032)	Invalid syntax for CKDS, PKDS or TKDS label name. **User action**: Correct *key_label* syntax. **REASONCODES**: TSS 020 (032)
3EA4 (16036)	The key record create callable service requires that the key created not already exist in the CKDS, PKDS, or TKDS. A key of the same label was found. **User action**: Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer. **REASONCODES**: TSS 02C (044)
3EA8 (16040)	Data in the PKDS record did not match the expected data. This occurs if the record does not contain a null PKA token and CHECK was specified. **User action**: If the record is to be overwritten regardless of its content, specify OVERLAY.
3EAC (16044)	One or more key labels specified as input to the PKA key generate or PKA key import service incorrectly refer to a retained private key. If generating a retained private key, this error may result from one of these conditions: • The private key name of the retained private key being generated is the same as an existing PKDS record, but the PKDS record label was not specified as the input skeleton (source) key identifier. • The label specified in the *generated_key_token* parameter as the target for the retained private key was not the same as the private-key name. If generating or importing a non-retained key, this error occurs when the label specified as the target key specifies a retained private key. The retained private key cannot be over-written. **User action:** Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer.
3EB0 (16048)	Retained keys on the PKDS cannot be deleted or updated using the PKDS key record delete or PKDS key record write callable services, respectively. **User action:** Use the retained key delete callable service to delete retained keys.
Reason code 0, return code 308 (776)	RACF failed your request to use this service. **User action**: Contact your ICSF or RACF administrator if you need this service.
Reason code 1, return code 308 (776)	RACF failed your request to use the key label. **User action**: Contact your ICSF or RACF administrator if you need this key.

Table 604. Reason codes for return code 8 (8) (continued)	
Reason Code Hex (Decimal)	**Description**
06E (110)-PAN, 028 (040)-ser. code, 02A (042)-exp. date, 066 (102)-dec table, 067 (103)-val. table, 06C (198)-pad data	The PAN, expiration date, service code, decimalization table data, validation data, or pad data is not numeric (X'F0' through X'F9'). The parameter must be character representations of numerics or hexadecimal data. **User action**: Review the numeric parameters or fields required in the service that you called and change to the format and values required.

Reason codes for return code C (12)

Table 605 on page 1489 lists reason codes returned from callable services that give return code 12. These reason codes indicate that the call to the callable service was not successful. Either cryptographic processing did not take place, or the last cryptographic processor was switched offline. Therefore, no output parameters were filled.

Note: The higher-order halfword of the reason code field for return code C (12) may contain additional coding. See reason codes 1790, 273C, and 2740 in this table. For example, in the reason code 42738, the 4 is an SVC 99 error code and the 2738 is listed in this table:

Table 605. Reason codes for return code C (12)	
Reason Code Hex (Decimal)	**Description**
0 (0)	ICSF is not available. One of the following situations is possible: • ICSF is not started • ICSF is started, but the DES-MK, AES-MK, or ECC-MK is not defined. • ICSF is started, but the requested function is not available. For instance, an ECC operation was requested but the required hardware is not installed. **User action:** Check the availability of ICSF with your ICSF administrator. OR CKDS Key Record Create2 or CKDS Key Record Write2 was called to add a variable-length key record to a fixed-length CKDS. A variable-length symmetric key token can only be added to a CKDS that supports variable-length records. **User action**: Contact the security administrator or system programmer to activate (refresh) a CKDS that supports variable-length records.
4 (4)	The CKDS or PKDS management service you called is not available because it has been disallowed by the ICSF User Control Functions panel. **User action**: Contact the security administrator or system programmer to determine why the CKDS or PKDS management services have been disallowed.
8 (8)	The service or algorithm is not available on current hardware. Your request cannot be processed. **User action**: Correct the calling program or run on applicable hardware.
C (12)	The service that you called is unavailable because the installation exit for that service had previously failed. **User action**: Contact your ICSF administrator or system programmer.
10 (16)	A requested installation service routine could not be found. Your request was not processed. **User action**: Contact your ICSF administrator or system programmer.
1C (28)	Cryptographic asynchronous processor failed. **User action**: Contact your IBM support center.
28 (40)	The callable service that you called is unsupported for AMODE(64) applications. Your request cannot be processed.
2C (44)	The callable service that you called was linked with the AMODE(64) stub. The application is not running AMODE(64). Your request cannot be processed. **User action**: Link your application with the service stub with the appropriate addressing mode.

Table 605. Reason codes for return code C (12) (continued)	
Reason Code Hex (Decimal)	**Description**
0C5 (197)	I/O error reading or writing to the DASD copy of the CKDS or PKDS in use by ICSF. **User action**: Contact your ICSF security administrator or system programmer. The RPL feedback code will be placed in the high-order halfword of the reason code field.
144 (324)	There was insufficient coprocessor memory available to process your request. This could include the Flash EPROM used to store keys, profiles and other application data. **User action**: Contact your system programmer or the IBM Support Center.
2FC (764)	The master key is not in a valid state. **User action**: Contact your ICSF administrator. **REASONCODES**: ICSF 2B08 (11016)
301 (769)	A cryptographic internal device driver component detected data contained in a cryptographic request that is not valid.
7D6 (2006)	TKE: PCB service error.
7D7 (2007)	TKE: Change type in PCB is not recognized.
7DF (2015)	Domain in CPRB not enabled by EMB mask.
7E1 (2017)	MKVP mismatch on Set MK.
7E5 (2021)	Cryptographic coprocessor adapter disabled.
7E9 (2025)	Enforcement mask error.
7F3 (2035)	Intrusion latch has been tripped. Services disabled.
7F5 (2037)	The domain specified is not valid.
7FB (2043)	OA certificate not found.
819 (2073)	The coprocessor has been disabled on the Support Element. It must be enabled on the Support Element prior to TKE accessing it. **User action**: Permit the selected coprocessor for TKE Commands on the Support Element and then re-open the host on TKE.
835 (2101)	AES flags in the function control vector are not valid.
839 (2105)	The processing for high performance secure keys fails due to a hardware error. **User action**: Contact your IBM support center.
BBD (3005)	The KDS I/O subtask timed out waiting for an exclusive ENQ on the *SYSZxKDS.xKDSdsn* resource, where *x* indicates the KDS type (C for CKDS, P for PKDS, and T for TKDS). A timeout will occur if one or more members of the ICSF sysplex group has not relinquished its ENQ on the resource. The KDS update operation has failed. **User action:** Issue D GRS,RES=(*nnnnn*), where *nnnnn* is the KDS resource name from message CSFM302A, to determine which system or systems hold the resource. Determine if action should be taken to cause the holding system to release its ENQ on the KDS resource.
BBE (3006)	Failure after exhausting retry attempts. IXCMSGO issued from CSFMIOST. **User action**: Contact your system programmer or the IBM Support Center.
BBF (3007)	The CKDS service failed due to unexpected termination of the ICSF Cross-System Services environment. The termination of the ICSF Cross-System Services environment was caused by a failure when ICSF issued the IXCMSGI macro. Message CSFM603 has been issued. **User action**: Report the occurrence of this error to your ICSF system programmer.
BC6 (3014)	There is an I/O error reading or writing to the DASD copy of the TKDS in use by ICSF. **User action**: Report the occurrence of this error to your ICSF system programmer.
BC7 (3015)	A bad header record is detected for the TKDS. **User action**: Report the occurrence of this error to your ICSF system programmer.

Table 605. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
BCF (3023)	The PKCS #11 TKDS is not available for processing. **User action**: Report the occurrence of this error to your ICSF system programmer.
BE6 (3046)	An RSA retained key can no longer be generated with its key-usage flag set to allow key unwrapping (KM-ONLY or KEY-MGMT). Key usage must be SIG-ONLY. **User action**: None required.
BE8 (3048)	The services using encrypted AES keys, encrypted DES, or encrypted ECC keys are not available because the master key is required but not loaded or there is no access to any cryptographic processors. Your request cannot be processed. **User action**: Check the availability of ICSF with your ICSF administrator
C00 (3072)	The serialization subtask terminated for an unexpected reason prior to completing the request. No dynamic CKDS or PKDS update services are possible at this point. **User action**: Contact your system programmer who can investigate the problem and restart the I/O subtask by stopping and restarting ICSF.
C01 (3073)	An error occurred attempting to obtain the system ENQ for a key data set update. **User action:** If the error is common and persistent, contact your system programmer or the IBM Support Center.
C03 (3075)	A symmetric key token was supplied in a key identifier parameter which is wrapped using the enhanced X9.24 key wrapping method. The cryptographic coprocessors available to process the request do not support the enhanced key wrapping. **User action**: Contact system personnel to get coprocessors installed on your system which will support the enhanced X9.24 key wrapping.
C06 (3078)	The CKDS was created with an unsupported LRECL.
C09 (3081)	An attempt was made to load a PKDS that only uses the ECC master key on a pre-HCR7780 release of ICSF. Pre-HCR7780 systems do not support the ECC master key and use of an ECC MK-only PKDS is not allowed. **User Action:** Change the PKDS selected. Specify a PKDS that is empty, uses an RSA master key, or uses both RSA and ECC master keys.
C0A (3082)	A callable service generated or updated a symmetric key token and the X9.24 enhanced wrapping method was used to wrap the key. This key token is not usable on your system and ICSF will not allow the key to be generated. The key was wrapped with the enhanced wrapping method because a CCA Cryptographic coprocessor that is a CEX3C or later has the default wrapping configuration set to enhanced. This was most likely done by TKE changing the configuration. **User Action:** Have the ICSF administrator set the default wrapping configuration to original for the LPAR that this system is running in.
C17 (3095)	While performing a coordinated KDS change master key operation, the sysplex KDS cluster members' new AES master key registers were loaded with different values. All sysplex KDS cluster members' (same active KDS) new AES master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action:** Ensure all sysplex KDS cluster members' new AES master key registers are loaded with the same value or all are empty and retry the function.
C18 (3096)	One or more sysplex KDS cluster members' new DES master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new DES master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action:** Ensure all sysplex KDS cluster members' new DES master key registers are loaded with the same value or all are empty and retry the function.
C19 (3097)	The sysplex KDS cluster members' new DES master key registers were loaded with different values during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new DES master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action:** Ensure all sysplex KDS cluster members' new DES master key registers are loaded with the same value or all are empty and retry the function.

Table 605. Reason codes for return code C (12) (continued)	
Reason Code Hex (Decimal)	**Description**
C1A (3098)	A coordinated KDS change master key was attempted with empty new master key registers. At least one of the new master key registers must be loaded with a value to perform a coordinated KDS change master key. **User action:** Load at least one of the new master key registers on all sysplex KDS cluster members with the same value and retry the function.
C1B (3099)	An ICSF subtask terminated during coordinated KDS refresh or coordinated KDS change master key processing. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C1C (3100)	An error occurred attempting to obtain an ENQ for performing either a coordinated KDS refresh or coordinated KDS change master key. **User action:** The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C1D (3101)	A target system (member of the sysplex KDS cluster) was unable to open the new KDS for either a coordinated KDS refresh or coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C1E (3102)	One or more sysplex KDS cluster members' new AES master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new AES master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action:** Ensure all sysplex KDS cluster members new AES master key registers are loaded with the same value or all are empty and retry the function.
C2B (3115)	Either a coordinated KDS refresh or coordinated KDS change master key was cancelled. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C2C (3116)	A catalog problem occurred during either a coordinated KDS refresh or coordinated KDS change master key. The problem occurred when looking up either the active KDS or new KDS in the catalog. **User action:** Ensure both the active KDS and new KDS are cataloged and retry the function.
C2D (3117)	A coordinated KDS refresh or coordinated KDS change master key was attempted when the cryptographic coprocessor level on the originating system is lower than the cryptographic coprocessor level on one or more of the other sysplex KDS cluster members. **User action:** Perform the coordinated KDS function from the system running the highest cryptographic coprocessor level. The cryptographic coprocessor levels can be checked by issuing the operator command D ICSF,CARDS,SYSPLEX=Y.
C2E (3118)	A coordinated KDS change master key was attempted with the DES new master key register loaded but with no current DES master key set. In order to perform a coordinated KDS change master key to a new DES master key, a valid DES master key must have previously been set. **User action:** Set a valid DES master key and then use the coordinated KDS change master key to change the DES master key.
C2F (3119)	A coordinated KDS change master key was attempted with the AES new master key register loaded but with no current AES master key set. In order to perform a coordinated KDS change master key to a new AES master key, a valid AES master key must have previously been set. **User action:** Set a valid AES master key and then use the coordinated KDS change master key to change the AES master key.
C32 (3122)	A sysplex communication failure occurred during either coordinated KDS refresh or coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated CKDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.

Table 605. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
C33 (3123)	A failure occurred processing KDS updates during a coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C34 (3124)	An internal failure occurred in a coordinated KDS subtask while performing either a coordinated KDS refresh or a coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C35 (3125)	An internal failure occurred in a coordinated KDS subtask while performing either a coordinated KDS refresh or a coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C36 (3126)	An internal failure occurred in the sysplex subtask while performing either a coordinated KDS refresh or coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C37 (3127)	An internal failure occurred in the serialization subtask while performing either a coordinated KDS refresh or coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C38 (3128)	An internal failure occurred in the I/O subtask while performing a coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function may be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C39 (3129)	A coordinated KDS change master key was attempted and at least one ICSF instance in the sysplex reported that it was unable to process the request. The coordinated KDS change master key functions are only available when all ICSF instances in the sysplex, regardless of active KDS, are running at a sufficient level to process the request. **User action:** Remove or upgrade all ICSF instances in the sysplex that are running without sufficient support and retry the function.
C3A (3130)	A target system (member of the sysplex KDS cluster) is not being responsive to a system that is originating either a coordinated KDS refresh or coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C3B (3131)	The active KDS could not be reenciphered to the new KDS during a coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C3E (3134)	A failure occurred either renaming the active KDS to the archive KDS or renaming the new KDS to the active KDS during a coordinated KDS refresh or coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C40 (3136)	A coordinated KDS refresh or coordinated KDS change master key was originated from a system at a lower ICSF FMID release level than one or more of the target systems (sysplex KDS cluster members). The coordinated KDS functions must be originated from a system running the highest ICSF FMID level. **User action:** Retry the function from a sysplex KDS cluster member running the highest ICSF FMID level.

Table 605. Reason codes for return code C (12) (continued)	
Reason Code Hex (Decimal)	**Description**
C41 (3137)	An internal failure occurred during the set master key step of a coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C42 (3138)	A failure occurred trying to back out from a failed rename of the active KDS to the archive KDS or a failed rename of the new KDS to the active KDS during a coordinated KDS refresh or coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C43 (3139)	A failure occurred switching the new KDS to the active KDS during either a coordinated KDS refresh or a coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C44 (3140)	A coordinated KDS refresh or a coordinated KDS change master key failed because one of the target systems (sysplex KDS cluster members) had not finished ICSF initialization. **User action:** Allow all sysplex KDS cluster members to finish ICSF initialization and retry the function.
C45 (3141)	A coordinated KDS change master key was attempted with the RSA new master key register loaded but with no current RSA master key set. In order to perform a coordinated KDS change master key to a new RSA master key, a valid RSA master key must have previously been set. **User action**: Set a valid RSA master key and then use the coordinated KDS change master key to change the RSA master key.
C46 (3142)	A coordinated KDS change master key was attempted with the ECC new master key register loaded but with no current ECC master key set. In order to perform a coordinated KDS change master key to a new ECC master key, a valid ECC master key must have previously been set. **User action**: Set a valid ECC master key and then use the coordinated KDS change master key to change the ECC master key.
C47 (3143)	A coordinated KDS change master key was attempted with the PKCS #11 new master key register loaded but with no current PKCS #11 master key set. In order to perform a coordinated KDS change master key to a new PKCS #11 master key, a valid PKCS #11 master key must have previously been set. **User action**: Set a valid PKCS #11 master key and then use the coordinated KDS change master key to change the PKCS #11 master key.
C48 (3144)	The sysplex KDS cluster members' new RSA master key registers were loaded with different values during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new RSA master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action**: Ensure all sysplex KDS cluster members' new RSA master key registers are loaded with the same value or are all empty, and retry the function.
C49 (3145)	The sysplex KDS cluster members' new ECC master key registers were loaded with different values during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new ECC master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action**: Ensure all sysplex KDS cluster members' new ECC master key registers are loaded with the same value or are all empty, and retry the function.
C4A (3146)	One or more sysplex KDS cluster members' new RSA master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new RSA master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action**: Ensure all sysplex KDS cluster members' new RSA master key registers are loaded with the same value or all are empty and retry the function.

Table 605. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
C4B (3147)	One or more sysplex KDS cluster members' new ECC master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new ECC master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action**: Ensure all sysplex KDS cluster members' new ECC master key registers are loaded with the same value or are all empty and retry the function.
C4C (3148)	The sysplex KDS cluster members' new PKCS #11 master key registers were loaded with different values or missing new PKCS #11 master key values during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new PKCS #11 master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action**: Ensure all sysplex KDS cluster members' new PKCS #11 master key registers are loaded with the same value or are all empty and retry the function.
C4D (3149)	One or more sysplex KDS cluster members' new P11 master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new P11 master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key. **User action**: Ensure all sysplex KDS cluster members' new P11 master key registers are loaded with the same value or are all empty and retry the function.
C59 (3161)	A coordinated KDS change master key was attempted, but one or more of the sysplex members failed to complete processing in the time allotted. **User action:** See *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. If the error is common and persistent, contact your system programmer or the IBM Support Center.
C80 (3200)	Key object's compliance mode is different than current setting of the Enterprise PKCS #11 coprocessors **User action:** Contact your ICSF administrator or system programmer. **ICSF administrator action:** The compliance mode setting on the Enterprise PKCS #11 coprocessors must be set to a value at least as restrictive as the key object that failed. Using the PKCS #11 Token Browser ISPF panels, examine the IBM CARD COMPLIANCE value for the key that failed. Set each Enterprise PKCS #11 coprocessor to this value using TKE.
C82 (3202)	A PKCS #11 Service found an error in DER encoded data returned from the Enterprise PKCS #11 Coprocessor. **User action:** Contact your system programmer or the IBM Support Center.
D21 (3361)	The subtask for the KDS multi-purpose callable service is not active. **User action:** Contact your system operator to stop and then start ICSF.
D22 (3362)	The subtask for the KDS multi-purpose callable service is terminating.
D23 (3363)	An attempt was made to use a KDS which has a KDS cluster identifier that matches an active KDS. This reason code is accompanied by message CSFM663I. **User action:** See the ICSF joblog for message CSFM663I and refer to *z/OS Cryptographic Services ICSF Messages* for information on how to proceed.
D24 (3364)	ICSF encountered continuous ISGQUERY failures while attempting a KDS operation. This reason code is accompanied by message CSFM664I. **User action:** See the ICSF joblog for message CSFM664I and refer to *z/OS Cryptographic Services ICSF Messages* for information on how to proceed.
D27 (3367)	There is no cryptographic coprocessor capable of performing the wrapping operation. **User action:** Check the required hardware section for the callable service being invoked.
D35 (3381)	z/OS UNIX System Services are not available. **User action:** Contact your system programmer. **System programmer action:** z/OS UNIX System Services may be temporarily unavailable. Check for console messages indicating that z/OS UNIX Systems is restarting and is now available. If the problem persists, contact the IBM Support Center.

Table 605. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
D5C (3420)	The master key required for the TR-31 key block is active, but the corresponding CMACZERO verification pattern is not present in the KDS header. **User action:** Contact your ICSF administrator or system programmer to update the KDS with the required CMACZERO verification pattern.
D5D (3421)	A request was made for a function that is not available because ICSF is running without an active CKDS or PKDS defined in the options data set. **User action:** If you wish to perform the function, you must restart ICSF with an active CKDS or PKDS defined in the options data set.
DB0 (3504)	The Options Data Set Refresh function ended. An error was encountered while reading the options data set.
DB3 (3507)	The Options Data Set Refresh function ended abnormally.
DC7 (3527)	ICSF is running with the requested KDS in an older format. The CSFKDU service can only be used when the requested KDS is in KDSR format. **User action:** Contact your system programmer to migrate the KDS to KDSR format.
DD0 (3536)	The operation failed because one or more cryptographic coprocessors were in a compliance mode that disallows the operation. **User action:** Change the operation to a compliant one to achieve the wanted results. If the cryptographic coprocessor is in compliance mode in error, configure the cryptographic coprocessor out of the compliance mode.
DD1 (3537)	The operation failed because the compliance mode required to perform the operation (including attempting to use a compliant-tagged token) is unavailable. **User action:** If this is an attempt to reencipher a CKDS that once held compliant-tagged key tokens but no longer does, perform a refresh of the CKDS to clear the compliant-tagged indicator and retry the request. Otherwise, configure one or more cryptographic coprocessors in the required compliance mode and retry the request.
DD3 (3539)	Unable to export a secure key to a CPACF protected key due to a problem with the internal transport key. This should be a temporary condition and should resolve on its own. **User action:** Retry the request. If the problem persists, contact the IBM Support Center.
DD4 (3540)	The operation failed because there is no cryptographic coprocessor available in migration mode. **User action:** Configure one or more cryptographic coprocessors in migration mode.
DDE (3550)	The Dilithium operation failed because the minimum hardware requirement was not met. **User action:** Ensure that the current hardware environment is IBM z15 or later.
DDF (3551)	A service was called to add or update a record in either the CKDS or PKDS. The LRECL of the KDS is insufficient for the record that would be written. One possible cause is trying to write a Dilithium key token to a PKDS which is not KDSRL format. **User action:** Contact your ICSF administrator.
1779 (6009)	One or more target systems (sysplex KDS cluster members) did not successfully load the new KDS during a coordinated KDS refresh or coordinated KDS change master key. This a common result of an unresponsive target system. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated CKDS administration failure. If the error is common and persistent, contact your system programmer or the IBM Support Center.
1780 (6016)	A DASD IO error was encountered during access of the CKDS, PKDS, or TKDS. **User action:** Contact your ICSF security administrator or system programmer. The SVC 99 error code will be placed in the high-order halfword of the reason code field.
178C (6028)	ESTAE could not be established in common I/O routines. **User action**: Contact your system programmer or the IBM Support Center.

Table 605. Reason codes for return code C (12) (continued)	
Reason Code Hex (Decimal)	**Description**
1790 (6032)	The dynamic allocation of the DASD copy of the CKDS, PKDS, or TKDS in use by ICSF failed. **User action**: Contact your ICSF security administrator or system programmer. The SVC 99 error code will be placed in the high-order halfword of the reason code field.
1794 (6036)	A dynamic deallocation error occurred when closing and deallocating a CKDS, PKDS, or TKDS. **User action**: Contact your security administrator or system programmer. The SVC 99 error code will be placed in the high-order halfword of the reason code field.
1795 (6037)	A failure occurred routing KDS updates to the originating system of a coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
1796 (6038)	The I/O subtask became out of sync with the sysplex KDS cluster during a coordinated KDS change master key. The I/O subtask will be restarted to get back in sync with the sysplex KDS cluster. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
1797 (6039)	ICSF was unable to attach a coordinated KDS subtask for either a coordinated KDS refresh or coordinated KDS change master key. **User action:** Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide* for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.
2724 (10020)	A key retrieved from the in-storage CKDS failed the MAC verification (MACVER) check and is unusable. **User action**: Contact your ICSF administrator.
2728 (10024)	A key retrieved from the in-storage CKDS or a key to be written to the PKDS was rejected for use by the installation exit. **User action**: Contact your ICSF administrator or system programmer.
272C (10028)	You cannot use the secure key import or multiple secure key import callable services because the cryptographic processor is not enabled for processing. The cryptographic coprocessor is not in special secure mode. **User action**: Contact your ICSF administrator (your administrator can enable the processing mode).
2734 (10036)	More than one key with the same label was found in the CKDS or PKDS. This function requires a unique key per label. The probable cause may be the use of an incorrect label pointing to a key type that allows multiple keys per label. **User action**: Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer to verify the contents of the CKDS or PKDS.
273C (10044)	OPEN of the PKDS in use by ICSF failed. **User action**: Contact your ICSF security administrator or system programmer.
2740 (10048)	I/O error reading or writing to the DASD copy of the CKDS or PKDS in use by ICSF. **User action**: Contact your ICSF security administrator or system programmer. The RPL feedback code will be placed in the high-order halfword of the reason code field. **REASONCODES**: TSS 0C5 (197)
274C (10060)	The I/O subtask terminated for an unexpected reason prior to completing the request. No dynamic CKDS, PKDS, or TKDS update services are possible at this point. **User action**: Contact your system programmer who can investigate the problem and restart the I/O subtask by stopping and restarting ICSF.
2B08 (11016)	The master key is not in a valid state. **User action**: Contact your ICSF administrator. **REASONCODES**: TSS 2FC (764)

Table 605. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
2B0C (11020)	The modulus of the public or private key is larger than allowed and configured in the CCC or FCV. You cannot use this key on this system. **User action**: Regenerate the key with a smaller modulus size.
2B10 (11024)	The RSA master key is not active. Possible reasons for this include: • The RSA master key is not loaded. • The master key verification pattern in the PKDS does not match the verification pattern of the RSA current master key. • The system administrator has used the ICSF User Control Functions panel to disable the RSA functions. **User action**: Contact the ICSF administrator to determine the problem and to make the RSA master key active.
2B1C (11036)	A PKDS is not available for processing. **User action**: Contact your ICSF administrator.
2B20 (11040)	The PKDS Control Record hash pattern is not valid. **User action**: Contact your ICSF administrator.
2B24 (11044)	The PKDS could not be accessed. **User action**: Contact your ICSF administrator.
2B28 (11048)	The coprocessor failed. **User action**: Contact your IBM support center.
2B2C (11052)	The specific coprocessor requested for service is temporarily unavailable. PKDS could not be accessed. The specific coprocessor may be attempting some recovery action. If recovery action is successful, the coprocessor will be made available. If the recovery action fails, the coprocessor will be made permanently unavailable. **User action**: Retry the function.
2B30 (11056)	The coprocessor failed. The response from the processor was incomplete. **User action**: Contact your IBM support center.
2B34 (11060)	The service could not be performed because the required coprocessor was not active or did not have a master key set, or the coprocessor did not have the required firmware update. **User action**: If the service required a specific coprocessor, verify that the value specified is correct. Reissue the request when the required coprocessor is available and has the master key set and the required firmware is present.
2B38 (11064)	Service could not be performed because of a hardware error on the coprocessor.
2B40 (11072)	Coprocessor configuration change. A CCA or EP11 coprocessor has been configured as an accelerator. TKE does not recognize coprocessors configured as accelerators.
2B41 (11073)	Coprocessor configuration change. Either a CCA coprocessor has been reconfigured to be a EP11 coprocessor, or a PKCS #11 coprocessor has been reconfigured to be a CCA coprocessor.
8CA2 (36002)	CSFPCI was called to set the RSA master key in any CCA cryptographic coprocessor. This function is disabled because dynamic RSA master key change is enabled and the RSA master key can only be changed from the ICSF TSO Change asymmetric master key utility.
8CB4 (36020)	A refresh of the CKDS failed because the DASD copy of the CKDS is enciphered under the wrong master key. This may have resulted from an automatic refresh during processing of the CKDS key record create callable service. **User action**: Contact your ICSF administrator.
8CE4 (36068)	A failure occurred during a coordinated KDS change master key operation because the DASD copy of the CKDS is enciphered under the wrong master key. **User action**: Contact your ICSF administrator.

Table 605. Reason codes for return code C (12) (continued)	
Reason Code Hex (Decimal)	**Description**
8CE5 (36069)	A failure occurred during a coordinated KDS change master key operation because the DASD copy of the TKDS is enciphered under the wrong master key. **User action**: Contact your ICSF administrator.
8CF4 (36084)	The master keys cannot be changed because ICSF is running in compatibility mode. **User action**: See 'Migration from PCF to z/OS ICSF' in *z/OS Cryptographic Services ICSF System Programmer's Guide* for an explanation of compatibility mode and how to change the master keys. Note that the coordinated change master key utility cannot be used to change master keys when running in compatibility mode.
8D14 (36116)	The PKDS specified for refresh, reencipher or activate has an incorrect dataset attribute. **User action**: Create a larger PKDS. See *z/OS Cryptographic Services ICSF System Programmer's Guide*.
8D3C (36156)	A PKCS #11 service is being requested. The service is disabled due to an ICSF FIPS self test failure. The request is not processed. **User action:** Report the problem to your IBM support center
8D40 (36160)	The attempt to reencipher the CKDS failed because there is an enhanced wrapped token in the CKDS. **User Action:** Reencipher the CKDS on a system that supports the enhanced wrapping method.
8D48 (36168)	A key data set has an LRECL attribute that is not valid. This could be because your release of ICSF does not support the KDS with that LRECL or a supplied KDS does not have the same LRECL as another KDS required for the utility being invoked. **User Action:** Use a KDS with an LRECL supported by the release of ICSF that you are using or supply a KDS with the same LRECL.
8D4D (36173)	A failure occurred during a coordinated KDS change master key operation because the DASD copy of the PKDS is enciphered under the wrong master key. **User Action:** Contact your ICSF administrator.
8D4E (36174)	A failure occurred during a coordinated KDS change master key operation because the DASD copy of the PKDS is enciphered under the wrong master key. **User Action:** Contact your ICSF administrator.
8D56 (36182)	A coprocessor failure was detected during initialization. **User action:** The error is accompanied by the CSFM540I message. Follow instructions associated with that message.
8D5A (36186)	A request was made to reencipher a CKDS. The CKDS specified cannot be reenciphered on this release of ICSF because the CKDS contains Variable-length Symmetric key tokens with an unrecognized algorithm or key type in the associated data section. Only key tokens with a recognized algorithm or key type can be managed on this release of ICSF. **User action:** Perform the reencipher operation on a release of ICSF which recognizes the algorithm and key type of all tokens in the specified CKDS.
8D5D (36189)	The TKDS has an incorrect dataset attribute. **User action:** Create a TKDS with valid dataset attributes. See *z/OS Cryptographic Services ICSF System Programmer's Guide*
8D73 (36211)	A request was made to process a key data set (CKDS, PKDS, or TKDS) which has an LRECL that is not supported for this operation at this release of ICSF. **User Action:** Retry the operation on a release that supports the LRECL.

Reason codes for return code 10 (16)

Table 606 on page 1500 lists reason codes returned from callable services that give return code 16.

Table 606. Reason codes for return code 10 (16)	
Reason Code Hex (Decimal)	**Description**
4 (4)	ICSF: Your call to an ICSF callable service resulted in an abnormal ending. **User action**: Contact your system programmer or the IBM Support Center.
150 (336)	An error occurred in the cryptographic hardware component. **User action**: Contact your system programmer or the IBM Support Center. **REASONCODES**: ICSF 4 (4)
22C (556)	The request parameter block failed consistency checking. **User action**: Contact your system programmer or the IBM Support Center. **REASONCODES**: ICSF 4 (4)
2C4 (708)	Inconsistent data was returned from the cryptographic engine. **User action**: Contact your system programmer or the IBM Support Center. **REASONCODES**: ICSF 4 (4)
2C5 (709)	Cryptographic engine internal error; could not access the master key data. **User action**: Contact your system programmer or the IBM Support Center. **REASONCODES**: ICSF 4 (4)
2C8 (712)	An unexpected error occurred in the Master Key manager. **User action**: Contact your system programmer or the IBM Support Center. **REASONCODES**: ICSF 4 (4)
BFA (3066)	Service terminated because ICSF is going down. **User action**: Contact your system programmer to restart ICSF.

Appendix B. Key token formats

For debugging purposes, this topic provides the formats for all CCA key tokens and trusted blocks. A key token is a data structure that contains information about a key and usually contains a key or keys. A trusted block is an extension of CCA PKA key tokens.

- "AES internal fixed-length key token" on page 1502
- "DES fixed-length key token" on page 1503
- "External RKX DES key token" on page 1508
- "Variable-length symmetric key token" on page 1509
- "Variable-length symmetric null key token" on page 1537
- "X9.143 (TR-31) key block header and optional block data" on page 1538
- "PKA key tokens" on page 1546
- "Trusted blocks" on page 1588

Master key verification pattern (MKVP)

A master key verification pattern (MKVP) exists within an internal AES, DES, or HMAC key token that has an encrypted key present. An MKVP also exists within an internal PKA key token that has an encrypted RSA or ECC private key present or within an active internal CCA trusted block. An MKVP enables the cryptographic coprocessor to detect whether the key within the token is enciphered by an available master key.

Note: A fixed-length symmetric key token that is stored in a non-KDSR CKDS does not have an MKVP. Before such a key token is used, the MKVP is copied from the CKDS header record and placed in the token.

Null key tokens

With some callable services, a null key token can be used instead of an internal or an external key token. A service generally accepts a null key token as a signal to use a key token with default values. A null key token always has a value of X'00' as its first byte.

Null AES fixed-length key token
A null AES key-token consisting of 64 bytes of X'00'.

Null DES fixed-length key token
A null DES key-token consisting of 64 bytes of X'00'.

The CSNBKIM callable service accepts input with offset zero valued to X'00'. In this special case, the service treats information that starts at offset 16 as an enciphered, single-length or double-length key.

Null variable-length symmetric key token
A null variable-length symmetric key token is an 8-byte structure. For more information, see Table 625 on page 1537. The null key token is used in CKDS records for tokens that do not have key material.

Null PKA key token
A null PKA key token is an 8-byte structure. For more information, see Table 632 on page 1549. The null key token is used in PKDS records for tokens that do not have key material.

Symmetric key tokens

Token validation value (fixed-length symmetric tokens)

ICSF uses the *token validation value (TVV)* to verify that a token is valid. The TVV prevents a key token that is not valid or that is overlaid from being accepted by ICSF. It provides a checksum to detect a corruption in the key token.

When an ICSF callable service generates a key token, it generates a TVV and stores the TVV in bytes 60-63 of the key token. When an application program passes a key token to a callable service, ICSF checks the TVV. To generate the TVV, ICSF performs a twos complement ADD operation (ignoring carries and overflow) on the key token, operating on four bytes at a time, starting with bytes 0-3 and ending with bytes 56-59.

Note: A fixed-length symmetric key token that is stored in a non-KDSR CKDS does not have an MKVP or TVV. Before such a key token is used, the MKVP is copied from the CKDS header record and the TVV is calculated and placed in the token.

AES internal fixed-length key token

Fixed-length AES key tokens are 64 bytes and consist of an internal key token identifier and a token version number, reserved fields, a flag byte containing various flag bits, and a token validation value.

Depending on the flag byte, the key token either contains an encrypted key, a clear key, or the key is absent. An encrypted key is encrypted under an AES master key that is identified by a master-key verification pattern (MKVP) in the key token. The key token contains a two-byte integer that specifies the length of the clear-key value in bits, valued to 0, 128, 192, or 256, and a two-byte integer that specifies the length of the encrypted-key value in bytes, valued to 0 or 32. An LRC checksum byte of the clear-key value is also in the key token.

All keys in fixed-length AES key tokens are DATA keys. If the flag byte indicates that a control vector (CV) is present, it must be all binary zeros. An all-zero CV represents the CV value of an AES DATA key. If a key is present without a control vector in a key token, that is accepted and the key is interpreted as an AES DATA key.

The AES internal key token is the structure that is used to hold AES keys that are either encrypted with the AES master-key or in clear text format.

Table 607 on page 1502 shows the format for an AES internal key token.

Table 607. AES internal fixed-length key token format

Offset (Dec)	Length of field (Bytes)	Description
00	1	X'01' (flag indicating that this is an internal key token)
01	3	Implementation-dependent bytes (X'000000' for ICSF)
04	1	Key token version number (X'04')
05	1	Reserved - must be set to X'00'

Table 607. AES internal fixed-length key token format (continued)

Offset (Dec)	Length of field (Bytes)	Description
06	1	Flag byte **Bit** **Meaning When Set On** **0** Encrypted key and master key verification pattern (MKVP) are present. Off for a clear key token. On for an encrypted key token. **1** Control vector (CV) value in this token has been applied to the key. **2** No key is present or the AES MKVP is not present if the key is encrypted. **3- 7** Reserved. Must be set to 0.
07	1	1-byte LRC checksum of clear key value.
08	8	Master key verification pattern (MKVP). (For a clear AES key token, this value is hex zeros.)
16	32	Key value, if present. Contains either: • A 256-bit encrypted-key value. The clear key value is padded on the right with binary zeros, and the entire 256-bit value is encrypted under the AES master-key using AES CBC mode with an initialization vector of binary zeros. • A 128-bit, 192-bit, or 256-bit clear-key value left-aligned and padded on the right with binary zeros for the entire 256-bit field.
48	8	8-byte control vector. (For a clear AES key token, this value is hex zeros.)
56	2	2-byte integer that specifies the length in bits of the clear key value.
58	2	2-byte integer that specifies the length in bytes of the encrypted key value. (For a clear AES key token, this value is hex zeros.)
60	4	Token validation value (TVV).

DES fixed-length key token

Fixed-length DES key tokens are 64 bytes and consist of a DES-enciphered key, a control vector, various flag bits, a token identifier and version number, reserved fields, and a token-validation value. An internal key-token also includes a master-key verification pattern.

If an internal fixed-length DES key-token has a key present, it contains a key enciphered by a DES master key. If an external fixed-length DES key-token has a key present, it contains a key enciphered by a key-encrypting key.

Version X'00' tokens are single-length, double-length, and triple-length keys for all key types. DATA key tokens with zero control vectors are version X'00' for single-length keys and version X'01' for double-length and triple-length keys.

Table 608 on page 1504 shows the format for a DES internal key token.

Table 608. DES internal fixed-length key token format

Offset (Dec)	Length of field (Bytes)	Description
00	1	X'01' (flag indicating this is an internal key token).
01	3	Implementation-dependent bytes (X'000000' for ICSF).
04	1	Key token version number (X'00' or X'01').
05	1	Reserved (X'00').
06	1	Flag byte **Bit** **Meaning When Set On** **0** Encrypted key and master key verification pattern (MKVP) are present. **1** Control vector (CV) value is present. **2** Key is used for no control vector (NOCV) processing. Valid for transport keys only. **3-6** Reserved. **7** Export prohibited.
07	1	Flag byte **Bit** **Meaning When Set On** **0-2** Key value encryption method. • 000 - The key is encrypted by using the original CCA method (ECB). • 001 - The encrypted key is wrapped using the enhanced method and SHA-1 (WRAP-ENH). • 010 - The encrypted key is wrapped using the enhanced method and SHA-256 (WRAPENH2). Requires CV bit ENH-ONLY to be enabled. Only valid with version X’00’ tokens. • 011 - The encrypted key wrapped using the enhanced method and SHA-256 and TDES-CMAC authentication code (WRAPENH3). Requires CV bit ENH-ONLY to be enabled. Only valid with version X’00’ tokens. These bits are ignored if the token contains a clear key. **3-7** Reserved.

Table 608. DES internal fixed-length key token format (continued)

Offset (Dec)	Length of field (Bytes)	Description
08	8	When the compliant-tag bit is off (bit 58 in the CV): Master Key Verification Pattern (MKVP). When the compliant-tag bit is on (bit 58 in the CV): **Offset | Length** **Description** **0 | 5** Truncated MKVP. **5 | 2** Reserved. **7 | 1** Key Derivation Function (KDF). When KDF is X'01' or X'02', the token is not considered compliant-tagged. Throughout the publications, they are referred to as DES KDF 01 or KDF 02 tokens. Only key tokens with a KDF higher than X'02' are referred to as compliant-tagged.
16	8	A single-length key, the left half of a double-length key, or Part A of a triple-length key. The value is encrypted under the master key when flag bit 0 is on. Otherwise, it is in the clear.
24	8	X'0000000000000000' if a single-length key, or the right half of a double-length key, or Part B of a triple-length key. The right half of the double-length key or Part B of the triple-length key is encrypted under the master key when flag bit 0 is on. Otherwise, it is in the clear. For WRAPENH3, this field will always hold ciphertext in order to obfuscate the length of the key.
32	8	The control vector (CV) for a single-length key or the left half of the control vector for a double-length key. For WRAPENH3, this field will contain the control vector with key form bits (40-42) indicating a triple-length key.
40	8	X'0000000000000000' if a single-length key or the right half of the control vector for a double-length operational key. For WRAPENH3, this field will hold the 8-byte TDES-CMAC over the entire key token, with this field set to hex zero before the calculation of the TDES-CMAC.
48	8	X'0000000000000000' if a single-length key or double-length key, or Part C of a triple-length key. Part C of a triple-length key is encrypted under the master key when flag bit 0 is on. Otherwise, it is in the clear. For WRAPENH3, this field will always hold ciphertext in order to obfuscate the length of the key.
56	3	Reserved (X'000000').

Table 608. DES internal fixed-length key token format (continued)		
Offset (Dec)	**Length of field (Bytes)**	**Description**
59	1	Key length for zero CV DATA keys: **Value** **Meaning** **B'00000000'** Single-length key (version 0 only). **B'00010000'** Double-length key (version 1 only). **B'00100000'** Triple-length key (version 1 only). All other values are reserved and undefined.
60	4	Token validation value.

Note: A fixed-length key token that is stored in a non-KDSR CKDS will not have an MKVP or TVV. Before such a key token is used, the MKVP is copied from the CKDS header record, and the TVV is calculated and placed in the token.

Table 609 on page 1506 shows the format for a DES external fixed-length key token.

Table 609. DES external fixed-length key token format		
Offset (Dec)	**Length of field (Bytes)**	**Description**
00	1	X'02' (flag indicating an external key token).
01	3	Implementation-dependent bytes (X'000000' for ICSF).
04	1	Key token version number (X'00' or X'01').
05	1	Reserved (X'00').
06	1	Flag byte. **Bit** **Meaning When Set On** **0** Encrypted key and master key verification pattern (MKVP) are present. **1** Control vector (CV) value is present. **2** Key is used for no control vector (NOCV) processing. Valid for transport keys only. Other bits are reserved and are binary zeros.

Table 609. DES external fixed-length key token format (continued)

Offset (Dec)	Length of field (Bytes)	Description
07	1	Flag byte. **Bit** **Meaning When Set On** **0-2** Key value encryption method. • 000 - The key is encrypted by using the original CCA method (ECB). • 001 - The encrypted key is wrapped using the enhanced method and SHA-1 (WRAP-ENH). • 010 - The encrypted key is wrapped using the enhanced method and SHA-256 (WRAPENH2). Requires CV bit ENH-ONLY to be enabled. Only valid with version X'00' tokens. • 011 - The encrypted key wrapped using the enhanced method and SHA-256 and TDES-CMAC authentication code (WRAPENH3). Requires CV bit ENH-ONLY to be enabled. Only valid with version X'00' tokens. These bits are ignored if the token contains a clear key. **3-7** Reserved.
08	8	Reserved (X'0000000000000000').
16	8	Single-length key or left half of a double-length key, or Part A of a triple-length key. The value is encrypted under a transport key-encrypting key when flag bit 0 is on. Otherwise, it is in the clear.
24	8	X'0000000000000000' if a single-length key or right half of a double-length key, or Part B of a triple-length key. The right half of a double-length key or Part B of a triple-length key is encrypted under a transport key-encrypting key when flag bit 0 is on. Otherwise, it is in the clear. For WRAPENH3, this field will always hold ciphertext in order to obfuscate the length of the key.
32	8	Control vector (CV) for single-length key or left half of CV for double-length key. For WRAPENH3, this field will contain the control vector with key form bits (40-42) indicating a triple-length key.
40	8	X'0000000000000000' if single-length key or right half of CV for double-length key. For WRAPENH3, this field will hold the 8-byte TDES-CMAC over the entire key token, with this field set to hex zero before the calculation of the TDES-CMAC.
48	8	X'0000000000000000' if a single-length key, double-length key, or Part C of a triple-length key. This key part is encrypted under a transport key-encrypting key when flag bit 0 is on. Otherwise, it is in the clear. For WRAPENH3, this field will always hold ciphertext in order to obfuscate the length of the key.
56-58	4	Reserved (X'000000').

Table 609. DES external fixed-length key token format (continued)		
Offset (Dec)	**Length of field (Bytes)**	**Description**
59	1	Key length for zero CV DATA keys. **Value** **Meaning** **B'00000000'** Single-length key (version 0 only). **B'00010000'** Double-length key (version 1 only). **B'00100000'** Triple-length key (version 1 only). All other values are reserved and undefined.
60-63	4	Token validation value.

External RKX DES key token

Table 610 on page 1508 defines an external DES key token that is called an *RKX key-token*. An RKX key-token is a special token that is used exclusively by the CSNDRKX callable service and the DES key-storage callable services (for example, CSNBKRW). No other callable services use or reference an RKX key-token or key-token record. For more information about the use of RKX key tokens, see "Remote key loading" on page 44.

Note: Callable services other than CSNDRKX and the DES key-storage callable services do not support RKX key-tokens or RKX key-token records.

RKX key-tokens are 64 bytes and have a token identifier flag (X'02'), a token version number (X'10'), and room for encrypted keys like normal CCA DES key tokens. Unlike normal CCA DES key-tokens, RKX key-tokens do not have a control vector, flag bits, or a token-validation value. In addition, they have a confounder value, a MAC value, and room for a third encrypted key.

Table 610. External RKX DES key-token format, version X'10'		
Offset	**Length**	**Meaning**
00	1	X'02' (a token identifier flag that indicates an external key-token).
01	3	Reserved, binary zero.
04	1	The token version number (X'10').
05	2	Reserved, binary zero.
07	1	Key length in bytes, including confounder.
08	8	Confounder.
16	8	Key left.
24	8	Key middle (binary zero if not used).
32	8	Key right (binary zero if not used).

Table 610. External RKX DES key-token format, version X'10' (continued)		
Offset	**Length**	**Meaning**
40	8	Rule ID. The trusted block rule identifier that is used to create this key token. A subsequent call to CSNDRKX can use this token with a trusted block rule that references the rule ID that must have been used to create this token. The trusted block rule can be compared with this rule ID for verification purposes. The Rule ID is an 8-byte string of ASCII characters, left-align, and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.
48	8	Reserved, binary zero.
56	8	MAC value. ISO 16609 TDES CBC-mode MAC, computed over the 56 bytes starting at offset 0 and including the encrypted key value and the rule ID by using the same MAC key that is used to protect the trusted block itself. This MAC value ensures that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. This MAC value must verify with the same trusted block that is used to create the key, thus binding the key structure to that specific trusted block.

Notes:

1. A fixed, randomly derived variant is exclusive-ORed with the MAC key before it is used to encipher the generated or exported key and confounder.
2. The MAC key is located within a trusted block (internal format) and can be recovered by decipherment under a variant of the PKA master key.
3. The trusted block is originally created in external form by the Trusted Block Create callable service and then converted to internal form by the PKA Key Import callable service prior to the Remote Key Export call.

Variable-length symmetric key token formats

Variable-length symmetric key token

The following table presents the format for a variable-length symmetric key token. The length of the token depends on the key type and algorithm.

Table 611. Variable-length symmetric key token		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
		Header

Table 611. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
0	1	Token flag **X'00'** for null tokens **X'01'** for internal tokens **X'02'** for external tokens
1	1	Reserved (X'00')
2	2	Length of the token in bytes
4	1	Token version number X'05' (May be X'00' for null tokens)
5	3	Reserved (X'000000')
		Wrapping information
8	1	Key material state. **X'00'** no key present (internal or external) **X'01'** key is clear (internal) **X'02'** key is encrypted under a key-encrypting key (external) **X'03'** key is encrypted under the master key (internal)
9	1	Key verification pattern (KVP) type. **X'00'** No KVP **X'01'** AES master key verification pattern **X'02'** key-encrypting key verification pattern **X'03'** Truncated AES master key verification pattern with compliance information
10	16	Non-compliant tagged token: Verification pattern of the key used to wrap the payload. Compliant-tagged token: 5 bytes of the AES MKVP followed by 3 bytes of internal compliance information. Values are left justified.

Table 611. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
26	1	Wrapping method - This value indicates the wrapping method used to protect the data in the encrypted section. **X'00'** key is in the clear **X'02'** AESKW **X'03'** PKOAEP2
27	1	Hash algorithm used in wrapping algorithm. • For wrapping method X'00' **X'00'** None. For clear key tokens. • For wrapping method X'02' **X'02'** SHA-256 • For wrapping method X'03' **X'01'** SHA-1 **X'02'** SHA-256 **X'04'** SHA-384 **X'08'** SHA-512
28	1	Payload version **X'00'** Variable-length payload **X'01'** Fixed-length payload All other values are reserved and must not be used.
29	1	Reserved (X'00')
		Associated data section
30	1	Associated data version (X'01')
31	1	Reserved (X'00')
32	2	Length of the associated data in bytes: *adl*
34	1	Length of the key name in bytes: *kl*
35	1	Length of the IBM extended associated data in bytes: *iead*
36	1	Length of the installation-definable associated data in bytes: *uad*
37	1	Reserved (X'00')

Table 611. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
38	2	Length of the payload in bits: *pl*
40	1	Reserved (X'00')
41	1	Type of algorithm for which the key can be used **X'01'** DES **X'02'** AES **X'03'** HMAC
42	2	Key type: For algorithm AES: **X'0001'** CIPHER **X'0002'** MAC **X'0003'** EXPORTER **X'0004'** IMPORTER **X'0005'** PINPROT **X'0006'** PINCALC **X'0007'** PINPRW **X'0009'** DKYGENKY **X'000A'** SECMSG **X'000B'** KDKGENKY For algorithm HMAC: **X'0002'** MAC For algorithm DES: **X'0008'** DESUSECV
44	1	Key-usage field count (*kuf*) - (1 byte) Key-usage field information defines restrictions on the use of the key.

Table 611. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	*kuf* * 2	Key-usage fields (*kuf* * 2 bytes) • For HMAC algorithm keys, refer to Table 613 on page 1514. • For AES algorithm Key-Encrypting keys (Exporter or Importer), refer to Table 620 on page 1527. • For AES algorithm CIPHER keys, refer to Table 621 on page 1529. • For AES algorithm MAC keys, refer to Table 614 on page 1516. • For AES algorithm PINCALC keys, refer to Table 615 on page 1517. • For AES algorithm PINPROT keys, refer to Table 616 on page 1518. • For AES algorithm PINPRW keys, refer to Table 617 on page 1520. • For AES algorithm DKYGENKY keys, refer to Table 618 on page 1522. • For AES algorithm SECMSG keys, refer to Table 619 on page 1526. • For AES algorithm KDKGENKY keys, refer to Table 624 on page 1536. • For DESUSECV keys, refer to Table 612 on page 1514.
45 + *kuf* * 2	1	Key-management field count (*kmf*) - (1 byte): • For AES and HMAC keys: 2 (no pedigree information) or 3 (has pedigree information) • For DESUSECV keys: 1 Key-management field information describes how the data is to be managed or helps with management of the key material.
46 + *kuf* * 2	*kmf* * 2	Key-management fields (kmf * 2 bytes): • For AES and HMAC algorithm keys, refer to Table 622 on page 1531. • For DESUSECV keys, refer to Table 623 on page 1536.
46 + *kuf* * 2 + *kmf* * 2	*kl*	Key name
46 + *kuf* * 2 + *kmf* * 2 + *kl*	*iead*	IBM extended associated data
46 + *kuf* * 2 + *kmf* * 2 + *kl* + *iead*	*uad*	Installation-defined associated data
		Clear key or encrypted payload

Table 611. Variable-length symmetric key token (continued)		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
30 + *adl*	*(pl+7)/8*	**Encrypted AESKW payload (internal keys)**: The encrypted AESKW payload is created from the unencrypted AESKW payload which is made up of the ICV/pad length/ hash options and hash length/hash options/hash of the associated data/key material/ padding. See unencrypted AESKW payload. **Encrypted PKOAEP2 payload (external keys)**: The encrypted PKOAEP2 payload is created using the PKCS #1 v1.2 encoding method for a given hash algorithm. The message (M) inside the encoding contains: [2 bytes: bit length of key] || [clear HMAC key]. M is encoded using OAEP and then encrypted with an RSA public key according to the standard. **Clear key payload**: When the key is clear, only the key material will be in the payload padded to the nearest byte with binary zeros.

Table 612. DESUSECV key-usage fields		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
44	1	Key-usage field count (*kuf*): 1
45	2	Key-usage field 1 High-order byte: **B'0000 0000'** Reserved All unused bits are reserved and must be zero. Low-order byte: **B'0000 0000'** Reserved All unused bits are reserved and must be zero.

Table 613. HMAC algorithm key-usage fields		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
44	1	Key-usage field count (*kuf*): 2

Table 613. HMAC algorithm key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	Key-usage field 1 High-order byte: **1xxx xxxx** Key can be used for generate. **x1xx xxxx** Key can be used for verify. All unused bits are reserved and must be zero. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where uuu are UDX-defined bits. All unused bits are reserved and must be zero.
47	2	Key-usage field 2 High-order byte: **1xxx xxxx** SHA-1 hash method is allowed for the key. **x1xx xxxx** SHA-224 hash method is allowed for the key. **xx1x xxxx** SHA-256 hash method is allowed for the key. **xxx1 xxxx** SHA-384 hash method is allowed for the key. **xxxx 1xxx** SHA-512 hash method is allowed for the key. All unused bits are reserved and must be zero. Low-order byte: All bits are reserved and must be zero.

Table 614. AES algorithm MAC key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (kuf): 2 – 3 Count is based on whether the key is DK enabled or not: ***kuf*** **DK enabled** **2** No **3** Yes
45	2	Key-usage field 1 High-order byte: **B'00xx xxxx'** Undefined. **B'01xx xxxx'** Key cannot be used for generate; key can be used for verify. **B'10xx xxxx'** Key can be used for generate; key cannot be used for verify. **B'11xx xxx*'** Key can be used for generate and verify. Not valid if offset 50 is X'01'. All unused bits are reserved and must be zero. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where *uuu* are UDX-defined bits. All unused bits are reserved and must be zero.
47	2	Key-usage field 2 High-order byte: **X'01'** CMAC mode. All unused bits are reserved and must be zero. Low-order byte: **B'0xxx xxxx'** Key cannot be used by CSNBPTR2 to verify authentication data using NIST SP 800-38B CMAC for ISO-4 to ISO-4 PAN change. Only valid with key usage VERIFY. **B'1xxx xxxx'** Key can be used by CSNBPTR2 to verify authentication data using NIST SP 800-38B CMAC for ISO-4 to ISO-4 PAN change. Only valid with key usage VERIFY. All bits are reserved and must be zero.

Table 614. AES algorithm MAC key associated data (continued)		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
49	2	Key-usage field 3 High-order byte when DK enabled: **X'01'** PIN_OP (DKPINOP) **X'03'** PIN_ADMIN1 (DKPINAD1) **X'04'** PIN_ADMIN2 (DKPINAD2) All unused values are reserved and must not be used. Low-order byte: **X'01'** DK enabled. All unused values are reserved and must not be used.

Table 615. AES algorithm PINCALC key associated data		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
44	1	Key-usage field count (*kuf*): 3
45	2	Key-usage field 1 High-order byte: **B'00xx xxxx'** Undefined. **B'10xx xxxx'** Key can be used for generate; key cannot be used for verify. All unused bits are reserved and must be zero. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where *uuu* are UDX-defined bits. All unused bits are reserved and must be zero.

Table 615. AES algorithm PINCALC key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
47	2	Key-usage field 2 High-order byte: **X'00'** Key can be used for Cipher Block Chaining (CBC). All unused values are reserved and must not be used. Low-order byte: All bits are reserved and must be zero.
49	2	Key-usage field 3 High-order byte when DK enabled: **X'01'** PIN_OP (DKPINOP) All unused values are reserved and must not be used. Low-order byte: **X'01'** DK enabled. All unused values are reserved and must not be used.

Table 616. AES algorithm PINPROT key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key usage fields count (kuf): 3 if value at offset 50 = X'01' (DK enabled), or 4 if value at offset 50 = X'00' (no field format specification).
45	2	Key-usage field 1 High-order byte: **B'00xx xxxx'** Undefined. **B'01xx xxxx'** Key cannot be used for encryption; key can be used for decryption. This is an inbound PIN protection key. **B'10xx xxxx'** Key can be used for encryption; key cannot be used for decryption. This is an outbound PIN protection key. **B'11xx xxxx'** Undefined. All unused bits are reserved and must be zero. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where *uuu* are UDX-defined bits. All unused bits are reserved and must be zero.

Table 616. AES algorithm PINPROT key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
47	2	Key-usage field 2 High-order byte: **X'00'** Key can be used for Cipher Block Chaining (CBC). All unused values are reserved and undefined.
		Key-usage field 2 Low-order byte: **Inbound key (value at offset 45 is B'01xx xxxx')** **B'xxx1 xxxx'** Key can be used to verify an encrypted PIN (EPINVER). **B'xxx0 xxxx'** Key cannot be used to verify an encrypted PIN. **B'xxxx 1xxx'** Key can be used to generate an alternate clear PIN (CPINGENA). **B'xxxx 0xxx'** Key cannot be used to generate an alternate clear PIN. **B'xxxx x1xx'** Key can be used to translate an encrypted PIN (PINXLATE). **B'xxxx x0xx'** Key cannot be used to translate an encrypted PIN. **B'xxxx xx1x'** Key can be used to reformat an encrypted PIN (REFORMAT). **B'xxxx xx0x'** Key cannot be used to reformat an encrypted PIN. **B'xxxx xxx1'** Key can be used to restrictively reformat an ISO-4 encrypted PIN to an ISO-1 encrypted PIN (RFMT4TO1). **B'xxxx xxx0'** Key cannot be used to reformat an ISO-4 encrypted PIN to an ISO-1 encrypted PIN. All unused bits are reserved and must be zero. **Outbound key (value at offset 45 is B'10xx xxxx')** **B'xx1x xxxx'** Key can be used to encrypt a clear PIN (CPINENC). **B'xx0x xxxx'** Key cannot be used to encrypt a clear PIN. **B'xxx1 xxxx'** Key can be used to generate an encrypted PIN (EPINGEN). **B'xxx0 xxxx'** Key cannot be used to generate an encrypted PIN. **B'xxxx x1xx'** Key can be used to translate an encrypted PIN (PINXLATE). **B'xxxx x0xx'** Key cannot be used to translate an encrypted PIN. **B'xxxx xx1x'** Key can be used to reformat an encrypted PIN (REFORMAT). **B'xxxx xx0x'** Key cannot be used to reformat an encrypted PIN. **B'xxxx xxx1'** Key can be used to restrictively reformat an ISO-1 encrypted PIN to an ISO-4 encrypted PIN (RFMT1TO4). **B'xxxx xxx0'** Key cannot be used to restrictively reformat an ISO-1 encrypted PIN to an ISO-4 encrypted PIN. All unused bits are reserved and must be zero.

Table 616. AES algorithm PINPROT key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
49	2	Key-usage field 3, high-order byte No field format specification (value at offset 50 is X'00') **Value** **Meaning** **X'00'** No field format specification (NOFLDFMT) All unused values are reserved and undefined. DK enabled (value at offset 50 is X'01') **X'01'** PIN_OP (DKPINOP) **X'02'** PIN_OPP (DKPINOPP) **X'03'** PIN_ADMIN1 (DKPINAD1) All unused values are reserved and undefined.
		Key-usage field 3, low-order byte Field format identifier: **X'00'** No field format specification (NOFLDFMT) **X'01'** DK enabled (DKPINOP, DKPINOPP, DKPINAD1) All unused values are reserved and undefined.
51	2	Key-usage field 4, high-order byte PIN block format usage: **B'xxxx xxx1'** Allow ISO-4 All undefined bits are reserved and must be zero.
		Key-usage field 4, low-order byte All bits are reserved and must be zero.

Table 617. AES algorithm PINPRW key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (kuf): 3

Table 617. AES algorithm PINPRW key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	Key-usage field 1 High-order byte: **B'00xx xxxx'** Undefined. **B'01xx xxxx'** Key cannot be used for generate; key can be used for verify. **B'10xx xxxx'** Key can be used for generate; key cannot be used for verify. **B'11xx xxxx'** Undefined. All unused bits are reserved and must be zero. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where *uuu* are UDX-defined bits. All unused bits are reserved and must be zero.
47	2	Key-usage field 2 High-order byte: **X'01'** CMAC mode All unused values are reserved and must not be used. Low-order byte: All bits are reserved and must be zero.
49	2	Key-usage field 3 High-order byte when DK enabled: **X'01'** PIN_OP (DKPINOP) All unused values are reserved and must not be used. Low-order byte: **X'01'** DK enabled. All unused values are reserved and must not be used.

Table 618. AES algorithm DKYGENKY key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (kuf): 2, 4-51 Count is based on the type of key to diversify (value of offset 45): **Value at offset 45** **Type of key to diversify / *kuf* count** **X'00'** D-ALL / *kuf* count: 2 **X'01'** D-CIPHER / *kuf* count: 4 **X'02'** D-MAC / *kuf* count: 4 (not DK enabled) or 5 (DK enabled) **X'03'** D-EXP / *kuf* count: 6 **X'04'** D-IMP / *kuf* count: 6 **X'05'** D-PPROT / *kuf* count: 5 **X'06'** D-PCALC / *kuf* count: 5 **X'07'** D-PPRW / *kuf* count: 5 **X'08'** D-SECMSG / *kuf* count: 4 **X'09'** D-KDKGKY / *kuf* count: 13, 25, 37, 49 Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.

Table 618. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	Key-usage field 1 High-order byte: Defines the key type to be generated. **X'00'** Any type listed below (D-ALL) **X'01'** CIPHER (D-CIPHER) **X'02'** MAC (D-MAC) **X'03'** EXPORTER (D-EXP) **X'04'** IMPORTER (D-IMP) **X'05'** PINPROT (D-PPROT) **X'06'** PINCALC (D-PCALC) **X'07'** PINPRW (D-PPRW) **X'08'** SECMSG (D-SECMSG) **X'09'** KDKGENKY (D-KDKGKY) All other values are reserved and undefined. Low-order byte: **1xxx xxxx** The key can be used as the base derivation key in the AES DUKPT key derivation algorithm. **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where *uuu* are UDX-defined bits. All unused bits are reserved and must be zero.

Table 618. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
47	2	Key-usage field 2: Indicates the key usage. High-order byte (key-usage field level of control): **B'1xxx xxxx'** The key usage fields of the key to be generated must be equal (KUF-MBE) to the related generated key usage fields that start with key usage field 3 below. **B'0xxx xxxx'** The key usage fields of the key identifier to be generated must be permitted (KUF-MBP) based on the related generated-key usage fields that start with key usage field 3 below. A key to be diversified is not permitted to have a higher level of usage than the related key usage fields permit. The key to be diversified is only permitted to have key usage that is less than or equal to the related key usage fields. The UDX-ONLY bit of the related key usage fields must always be equal in both the generating key and the generated key. Undefined when the value at offset 45 = X'00' (D-ALL). All other values are reserved and undefined. Low-order byte (key-derivation sequence level): **X'00'** DKYL0. Generate a key based on the key usage byte at offset 45. **X'01'** DKYL1. Generate a level 0 diversified key with key type DKYGENKY. **X'02'** DKYL2. Generate a level 1 diversified key with key type DKYGENKY. All other values are reserved and undefined.

Table 618. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
49 (if defined)	2	Key-usage field 3 (related generated key usage fields): These values determine allowable key usage of key to be generated. Meaning depends on value of offset 45: **X'01'** Same as key-usage field 1 of AES CIPHER key. **X'02'** Same as key-usage field 1 of AES MAC key. **X'03'** Same as key-usage field 1 of AES EXPORTER key. **X'04'** Same as key-usage field 1 of AES IMPORTER key. **X'05'** Same as key-usage field 1 of AES PINPROT key. **X'06'** Same as key-usage field 1 of AES PINCALC key. **X'07'** Same as key-usage field 1 of AES PINPRW key. **X'08'** Same as key-usage field 1 of AES SECMSG key. **X'09'** Same as key-usage field 1 of AES KDKGENKY key.
51 (if defined)	2	Key-usage field 4 (related generated key usage fields): These values determine allowable key usage of key to be generated. Meaning depends on value of offset 45: **X'01'** Same as key-usage field 2 of AES CIPHER key. **X'02'** Same as key-usage field 2 of AES MAC key. **X'03'** Same as key-usage field 2 of AES EXPORTER key. **X'04'** Same as key-usage field 2 of AES IMPORTER key. **X'05'** Same as key-usage field 2 of AES PINPROT key. **X'06'** Same as key-usage field 2 of AES PINCALC key. **X'07'** Same as key-usage field 2 of AES PINPRW key. **X'08'** Same as key-usage field 2 of AES SECMSG key. **X'09'** Same as key-usage field 2 of AES KDKGENKY key (1st KUF of required 1st active/passive related key-usage field blocks).

Table 618. AES algorithm DKYGENKY key associated data (continued)		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
53 (if defined)	2	Key-usage field 5 (related generated key usage fields): These values determine allowable key usage of key to be generated. Meaning depends on value of offset 45: **X'02'** Same as key-usage field 3 of AES MAC key. **X'03'** Same as key-usage field 3 of AES EXPORTER key. **X'04'** Same as key-usage field 3 of AES IMPORTER key. **X'05'** Same as key-usage field 3 of AES PINPROT key. **X'06'** Same as key-usage field 3 of AES PINCALC key. **X'07'** Same as key-usage field 3 of AES PINPRW key. **X'09'** Same as key-usage field 3 of AES KDKGENKY key (1st KUF of required 1st active/passive related key-usage field blocks).
55 (if defined)	2	Key-usage field 6 (related generated key usage fields): These values determine allowable key usage of key to be generated. Meaning depends on value of offset 45: **X'03'** Same as key-usage field 4 of AES EXPORTER key. **X'04'** Same as key-usage field 4 of AES IMPORTER key. **X'05'** Same as key-usage field 4 of AES PINPROT key. **X'09'** Same as key-usage field 4 of AES KDKGENKY key (1st KUF of required 1st active/passive related key-usage field blocks).

Table 619. AES algorithm SECMSG key associated data		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
44	1	Key-usage field count (*kuf*): 2

Table 619. AES algorithm SECMSG key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	Key-usage field 1 High-order byte: Secure message encryption enablement: **Value** **Meaning** **X'00'** Enable the encryption of PINs in an EMV secure message (SMPIN). All other values are reserved and undefined. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx x*uuu*** Reserved for UDXs, where *uuu* are UDX-defined bits. All unused bits are reserved and must be zero.
47	2	Key-usage field 2: Indicates the key usage. High-order byte: Service restriction: **Value** **Meaning** **X'00'** Any verb can use this key (ANY-USE). **X'01'** Only CSNBDPC can use this key (DPC-ONLY). All other values are reserved and undefined. Low-order byte (reserved). All unused bits are reserved and must be zero

Table 620. AES algorithm KEK key-usage fields

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (*kuf*): 4

Table 620. AES algorithm KEK key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	Key-usage field 1, high-order byte EXPORTER: **1xxx xxx0** Key can be used for EXPORT. **x1xx xxx0** Key can be used for TRANSLAT. **xx1x xxx0** Key can be used for GEN-OPEX. **xxx1 xxx0** Key can be used for GEN-IMEX. **xxxx 1xx0** Key can be used for GEN-EXEX. **xxxx x1x0** Key can be used for GEN-PUB. **0000 0001** Key can wrap an AES or DES key using the Key Block Binding key wrapping method as defined in ISO/DIS 20038 (EXPTT31D). All unused bits are reserved and must be zero. IMPORTER: **1xxx xxx0** Key can be used for IMPORT. **x1xx xxx0** Key can be used for TRANSLAT. **xx1x xxx0** Key can be used for GEN-OPIM. **xxx1 xxx0** Key can be used for GEN-IMEX. **xxxx 1xx0** Key can be used for GEN-IMIM. **xxxx x1x0** Key can be used for GEN-PUB. **0000 0001** Key can unwrap an AES or DES key using the Key Block Binding key wrapping method as defined in ISO/DIS 20038 (IMPTT31D). All unused bits are reserved and must be zero.
		Key-usage field 1, low-order byte UDX control: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where uuu are UDX-defined bits. All unused bits are reserved and must be zero.
47	2	Key-usage field 2 High-order byte: **0000 0001** Key can wrap a TR-31 key block version “D” (VARDRV-D). Only valid if value at offset 45 is B’0000 0001’ (EXPTT31D or IMPTT31D). **1xxx xxx0** Key can wrap a TR-31 key block. Not valid if value at offset 45 is B’0000 0001’ (EXPTT31D or IMPTT31D). All unused bits are reserved and must be zero. Low-order byte: **xxxx xxx1** This KEK can export a key in RAW format. All unused bits are reserved and must be zero.

Table 620. AES algorithm KEK key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
49	2	Key-usage field 3 High-order byte: **1xxx xxxx** Key can wrap DES keys. **x1xx xxxx** Key can wrap AES keys. **xx1x xxxx** Key can wrap HMAC keys. **xxx1 xxxx** Key can wrap RSA keys. **xxxx 1xxx** Key can wrap ECC keys. **xxxx x1xx** Key can wrap QSA keys. All unused bits are reserved and must be zero. Low-order byte: All bits are reserved and must be zero.
51	2	Key-usage field 4 High-order byte: **1xxx xxxx** Key can wrap DATA class keys. **x1xx xxxx** Key can wrap KEK class keys. **xx1x xxxx** Key can wrap PIN class keys. **xxx1 xxxx** Key can wrap DERIVATION class keys. **xxxx 1xxx** Key can wrap CARD class keys. **xxxx x1xx** Key can wrap CVAR class keys. All unused bits are reserved and must be zero. Low-order byte: All bits are reserved and must be zero.

Table 621. AES algorithm CIPHER key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (*kuf*): 2

Table 621. AES algorithm CIPHER key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	Key-usage field 1 High-order byte: **1xxx xxxx** Key can be used for encryption. **x1xx xxxx** Key can be used for decryption. **xx1x xxxx** Key can only be used for data translate. All unused bits are reserved and must be zero. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where uuu are UDX-defined bits. All unused bits are reserved and must be zero.

Table 621. AES algorithm CIPHER key associated data (continued)		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
47	2	Key-usage field 2 High-order byte: **X'00'** Key can be used for Cipher Block Chaining (CBC). **X'01'** Key can be used for Electronic Code Book (ECB). **X'02'** Key can be used for Cipher Feedback (CFB). **X'03'** Key can be used for Output Feedback (OFB). **X'04'** Key can be used for Galois/Counter Mode (GCM) **X'05'** Key can be used for XEX-based Tweaked CodeBook Mode with CipherText Stealing (XTS) **X'06'** Key can be used for Format Preserving method FF1. **X'07'** Key can be used for Format Preserving method FF2. **X'08'** Key can be used for Format Preserving method FF2.1. **X'FF'** Key can be used for any mode of encryption All unused values are reserved and must not be used. Low-order byte: All bits are reserved and must be zero.

Table 622. AES and HMAC algorithm key-management fields		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
49	1	Key-management field count (kmf): 2 or 3.

Table 622. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
50	2	Key-management field 1. High-order byte: **1xxx xxxx** Allow export using symmetric key. **x1xx xxxx** Allow export using unauthenticated asymmetric key. **xx1x xxxx** Allow export using authenticated asymmetric key. **xxx1 xxxx** Allow export in RAW format. **xxxx 1xxx** Allow export to CPACF protected key format. **xxxx xxx1** Compliant-tagged key. Applies to AES only. All other bits are reserved and must be zero. Low-order byte: --symmetric-- **1xxx xxxx** Prohibit export using DES key. **x1xx xxxx** Prohibit export using AES key. --asymmetric-- **xxxx 1xxx** Prohibit export using RSA key. All other bits are reserved and must be zero.

Table 622. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
48 + *kuf* * 2	2	Key-management field 2. High-order byte: **11xx xxxx** Key, if present, is incomplete. Key requires at least 2 more parts. **10xx xxxx** Key, if present, is incomplete. Key requires at least 1 more part. **01xx xxxx** Key, if present, is incomplete. Key can be completed or have more parts added. **00xx xxxx** Key, if present, is complete. No more parts can be added. All other bits are reserved and must be zero. Low-order byte (Security History): **xxx1 xxxx** Key was encrypted with an untrusted KEK. **xxxx 1xxx** Key was in a format without type/usage attributes. **xxxx x1xx** Key was encrypted with key weaker than itself. **xxxx xx1x** Key was in a non-CCA format. **xxxx xxx1** Key was encrypted in ECB mode. All other bits are reserved and must be zero.

Table 622. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
50 + *kuf* * 2	2	Key-management field 3 - Pedigree (this field may or may not be present). Indicates how key was originally created and how it got into the current system. High-order byte: Pedigree Original. **X'00'** Unknown (Key Token Build2, Key Translate2). **X'01'** Other - method other than those defined here, probably used in UDX. **X'02'** Randomly Generated (Key Generate2). **X'03'** Established by key agreement (ECC Diffie-Hellman). **X'04'** Created from cleartext key components (Key Part Import2). **X'05'** Entered as a cleartext key value (Key Part Import2, Secure Key Import2). **X'06'** Derived from another key. **X'07'** Cleartext keys or key parts that were entered at TKE and secured from there to the target card (operational key load). All unused values are reserved and undefined.

Table 622. AES and HMAC algorithm key-management fields (continued)		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
50 + *kuf* * 2 (cont'd)	2 (cont'd)	Low-order byte: Pedigree Current. **X'00'** Unknown (Key Token Build2). **X'01'** Other - method other than those defined here, probably used in UDX. **X'02'** Randomly Generated (Key Generate2). **X'03'** Established by key agreement (ECC Diffie-Hellman). **X'04'** Created from cleartext key components (Key Part Import2). **X'05'** Entered as a cleartext key value (Key Part Import2, Secure Key Import2). **X'06'** Derived from another key. **X'07'** Imported from a CCA 05 variable length token with pedigree field (Symmetric Key Import2). **X'08'** Imported from a CCA 05 variable length token with no pedigree field (Symmetric Key Import2). **X'09'** Imported from a CCA token that had a CV. **X'0A'** Imported from a CCA token that had no CV or a zero CV. **X'0B'** Imported from a TR-31 key block that contained a CCA CV (ATTR-CV option) (TR-31 Import). **X'0C'** Imported from a TR-31 key block that did not contain a CCA CV (TR-31 Import). **X'0D'** Imported using PKCS 1.2 RSA encryption (Symmetric Key Import2). **X'0E'** Imported using PKCS OAEP encryption (Symmetric Key Import2). **X'0F'** Imported using PKA92 RSA encryption (Symmetric Key Import2). **X'10'** Imported using RSA ZERO-PAD encryption (Symmetric Key Import2). **X'11'** Converted from a CCA token that had a CV (Key Translate2). **X'12'** Converted from a CCA token that had no CV or a zero CV (Key Translate2). **X'13'** Cleartext keys or key parts that were entered at TKE and secured from there to the target card (operational key load).

Table 622. AES and HMAC algorithm key-management fields (continued)		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
50 + *kuf* * 2 (cont'd)	2 (cont'd)	Low-order byte: Pedigree Current. **X'14'** Exported from a CCA 05 variable length token with pedigree field (Symmetric Key Export). **X'15'** Exported from a CCA 05 variable length token with no pedigree field (Symmetric Key Export). **X'16'** Exported using PKCS OAEP encryption (Symmetric Key Export). All unused values are reserved and undefined.

Table 623. DESUSECV key-management fields		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
47	1	Key-management field count (*kmf*): 1
48	2	Key-management field 1 High-order byte: **B'0000 0000'** Reserved All unused bits are reserved and must be zero. Low-order byte: **B'0000 0000'** Reserved All unused bits are reserved and must be zero.

Table 624. AES algorithm KDKGENKY key-usage fields		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
44	1	Key-usage field count (*kuf*): 13, 25, 37, or 49. Each key-usage field is two bytes in length.

Table 624. AES algorithm KDKGENKY key-usage fields (continued)		
Offset (Dec)	**Length of Field (Bytes)**	**Description**
45	2	Key-usage field 1, high-order byte Key diversification type: **X'00'** Entity type A (KDKTYPEA) **X'01'** Entity type B (KDKTYPEB) All other values are reserved and undefined.
		Key-usage field 1, low-order byte User-defined extension control: **B'xxxx 1xxx'** Key can only be used in UDXs (used in KGN, KIM, KEX). **B'xxxx 0xxx'** Key can be used in UDXs and CCA. **B'xxxx xuuu'** Reserved for UDXs, where *uuu* are UDX-defined bits. All unused bits are reserved and must be zero.
47	*amb* + *acb* + *apb* + *awp*	Key-usage field 2 (active/passive key-usage field block) For the format of an active/passive key-usage field block, see the 'AES DKYGENKY and AES KDKGENKY active/passive related key-usage field block' table in *z/OS Cryptographic Services ICSF Application Programmer's Guide*. At least one active/passive related KUF block is required and a maximum of four blocks is allowed. The sequence of the blocks is in ascending numerical order, based on the numeric value of the key type of the key to be derived or generated, found at offset 0 of the block header. This is the required order of the blocks: 1. Active AES MAC/passive AES MAC related KUF block (*amb* bytes, where *amb* = 0 or 24). 2. Active AES CIPHER/passive AES CIPHER related KUF block (*acb* bytes, where *acb* = 0 or 24). 3. Active AES PINPROT/passive AES PINPROT related KUF block (*apb* bytes, where *apb* = 0 or 24). 4. Active AES EXPORTER/passive AES EXPORTER related KUF block (*awp* bytes, where *awp* = 0 or 24). where *amb + acb + apb + awp* = 24, 48, 72, or 96.

Variable-length symmetric null key token

The following table shows the format for a variable-length symmetric null key token.

Table 625. Variable-length symmetric null token	
Bytes	**Description**
0	X'00' Token identifier (indicates that this is a null key token).

Table 625. Variable-length symmetric null token (continued)	
Bytes	**Description**
1	Version, X'00'.
2-3	X'0008' Length of the key token structure.
4-7	Ignored (zero).

X9.143 (TR-31) key block header and optional block data

X9.143 (TR-31) key blocks are supported by ICSF. This topic describes the format of the X9.143 key block header and the header values that ICSF supports. It also describes the TR-31 optional blocks that can be used by ICSF. See ANSI X9.143 Retail Financial Services Interoperable Secure Key Block Specification for the definition of a X9.143 key block.

X9.143 key block header

A X9.143 key block always begins with a key block header consisting of a required 16-byte fixed-length portion followed by from 0 - 99 variable-length optional blocks. Table 626 on page 1538 shows the format of the required fixed-length portion of the header and only shows those values supported by ICSF.

Table 626. Format and supported values of the required header for a X9.143 key block		
Offset (bytes)	**Length (bytes)**	**X9.143 key block header field name**
0	1	Key block version ID. Identifies the version of the key block, which defines the method by which it is cryptographically protected and the content and layout of the block. The following key block version ID values defined by X9.143 are the only ones supported by ICSF: **ASCII value** **Meaning** **"A"** Key block protected by the Key Variant Binding Method 2005 Edition (maps to TR-31 Translate rule-array keyword VARXOR-A). This method is deprecated and should not be used for any new development. **"B"** Key block protected by the Key Derivation Binding Method 2010 Edition (maps TR-31 Translate rule-array keyword VARDRV-B). **"C"** Key block protected by the Key Variant Binding Method 2010 Edition (maps to TR-31 Translate rule-array keyword VARXOR-C). **"D"** Key block protected by the Key Derivation Binding Method 2017 Edition (maps to TR-31 Translate rule-array keyword VARDRV-D).
1	4	Key block length. Provides the key-block length after encoding. Length includes the entire block (header + encrypted confidential data + MAC).

Table 626. Format and supported values of the required header for a X9.143 key block (continued)

Offset (bytes)	Length (bytes)	X9.143 key block header field name
5	2	Key usage. Provides information about the intended function of the protected key/ sensitive data. The following key usage values defined by TR-31 are the only ones fully supported by ICSF: • "B0", "B1", "B3" • "C0" • "D0", "D3" • "E0", "E1", "E2", "E3", "E4", "E5" • "F0", "F1", "F2", "F3", "F4" • "I0" • "K0", "K1", "K4" • "M0", "M1", "M3", "M6", "M7" • "P0" • "V0", "V1", "V2" See Table 627 on page 1541 for descriptions of the key usage values.
7	1	Algorithm. The approved algorithm for which the key may be used. The following algorithm values defined by TR-31 are the only ones supported by ICSF: **ASCII value** **Meaning** **"A"** Advanced Encryption Standard (AES) (maps to TR-31 Translate AES key in the source key-token). **"D"** Data Encryption Algorithm (DEA) (maps to TR-31 Translate single-length DES key in the source key-token). **"H" (ASC X9 TR 31-2018)** Hash-based message authentication code (HMAC) (specify the underlying hash algorithm in optional field with a block ID of "HM") (maps to TR-31 Translate HMAC key in the source key-token). **"H" (ISO 20038)** HMAC-SHA-1 (maps to TR-31 Translate HMAC key in the source key-token). **"I" (ISO 20038)** HMAC-SHA-2 (maps to TR-31 Translate HMAC key in the source key-token). **"T"** Triple Data Encryption Algorithm (TDEA) (maps to TR-31 Translate double-length or triple-length Triple-DES key in the source key-token).

Table 626. Format and supported values of the required header for a X9.143 key block (continued)

Offset (bytes)	Length (bytes)	X9.143 key block header field name
8	1	Mode of use. Defines the operation the protected key can perform. The following mode of use values defined by TR-31 are the only ones supported by ICSF: **ASCII value** **Meaning** **"B"** Both encrypt and decrypt data, wrap and unwrap keys (maps to TR-31 Translate rule-array keyword ENCDEC). **"C"** Both generate and verify of check/PIN values (maps to TR-31 Translate rule-array keyword GENVER). **"D"** Decrypt data, unwrap keys only (maps to TR-31 Translate rule-array keyword DECONLY). **"E"** Encrypt data, wrap keys only (maps to TR-31 Translate rule-array keyword ENCONLY). **"G"** Generate of check/PIN values only (maps to TR-31 Translate rule-array keyword GEN-ONLY). **"N"** No special restrictions (other than restrictions implied by the key usage) (maps to TR-31 Translate rule-array keyword ANY). **"V"** Verify of check/PIN values only (maps to TR-31 Translate rule-array keyword VERONLY). **"X"** Key used to derive other key or keys (maps to TR-31 Translate rule-array keyword DERIVE). **"1"** IBM proprietary mode for keys (maps to TR-31 Translate rule-array keyword ATTR-CV).
9	2	Key version number. Version number used to indicate that contents of the key block is a component (partial key) or to prevent reinjection of old keys.

Table 626. Format and supported values of the required header for a X9.143 key block (continued)		
Offset (bytes)	**Length (bytes)**	**X9.143 key block header field name**
11	1	Exportability. Defines whether the protected key may be transferred outside the cryptographic domain in which the key is found. The following exportability values defined by TR-31 are the only ones supported by CCA: **ASCII value** **Meaning** **"E"** Extra sensitive: Key exportable under a key-encrypting key meeting the requirements of X9.24 Parts 1 or 2; no ECB mode KEK allowed (maps to TR-31 Translate rule-array keyword EXP-TRST). **"N"** Non-exportable (maps to TR-31 Translate rule-array keyword EXP-NONE). **"S"** Sensitive: Key exportable under any key-encrypting key not necessarily meeting the requirements of X9.24 Parts 1 or 2 (maps to TR-31 Translate rule-array keyword EXP-ANY).
12	2	Number of optional blocks. Defines the number of optional blocks included in the key block. The minimum value is zero and the maximum is 99.
14	1	Key Context. Defines whether the key block is in a key exchange context (wrapped by a transport key) or in a storage context (for example, wrapped by the master key). The key context value does not require a certain exportability setting. **ASCII value** **Meaning** **"0"** This is the equivalent to external CCA key tokens. **"1"** This is the equivalent to internal CCA key tokens. **"2"** This is the equivalent to external CCA key tokens.
15	1	Reserved.

Note: Information specific to X9-143 is taken from ANSI X9.143 Retail Financial Services Interoperable Secure Key Block Specification.

Table 627. Key usage values and meanings	
ASCII value	**Meaning**
"B0"	BDK base derivation key (maps to TR-31 Translate rule array keyword "BDK"). This key is used to derive the initial PIN encryption key (IPEK) in the derived unique key per transaction (DUKPT) process defined in X9.24-3.

Table 627. Key usage values and meanings (continued)	
ASCII value	**Meaning**
"B1"	Initial DUKPT key (maps to TR31 Key Create rule array keyword "DUKPT"). This key is used as the initial PIN encryption key (IPEK) in the derived unique key per transaction (DUKPT) process as defined in X9.24-3. Only valid with Key Usage 'X'.
"B3"	Key Derivation key (maps to TR31 Key Create rule array keyword "KDK"). This key is used as an input to an irreversible key derivation function to derive other keys. Only valid with Key Usage 'X'.
"C0"	CVK card verification key (maps to TR-31 Translate rule array keyword "CVK"). These are the keys used for computing or verifying (against a supplied value) a card verification code with the CVV, CVC, CVC2, and CVV2 algorithms.
"D0"	Symmetric data encryption key (maps to TR31 Key Create rule array keyword "ENC").
"D3"	Symmetric data encryption key for sensitive data (maps to TR31 Key Create rule array keyword "ENCSENS").
"E0"	Derivation key for an EMV/chip issuer master key: Application cryptograms (maps to TR31 Key Create rule array keyword "EMVAC-E").
"E1"	Derivation key for an EMV/chip issuer master key: Secure messaging for confidentiality (maps to TR31 Key Create rule array keyword "EMVSC-E").
"E2"	Derivation key for an EMV/chip issuer master key: Secure messaging for integrity (maps to TR31 Key Create rule array keyword "EMVSI-E").
"E3"	Derivation key for an EMV/chip issuer master key: Data authentication code (maps to TR31 Key Create rule array keyword "EMVDA-E").
"E4"	Derivation key for an EMV/chip issuer master key: Dynamic numbers (maps to TR31 Key Create rule array keyword "EMVDN-E").
"E5"	Derivation key for an EMV/chip issuer master key: Card personalization (maps to TR31 Key Create rule array keyword "EMVCP-E").
"F0"	EMV/chip issuer master key: Application cryptograms (maps to TR31 Create rule array keyword "EMVAC-F").
"F1"	EMV/chip issuer master key: Secure messaging for confidentiality (maps to TR31 Create rule array keyword "EMVSC-F").
"F2"	EMV/chip issuer master key: Secure messaging for integrity (maps to TR31 Create rule array keyword "EMVSI-F").
"F3"	EMV/chip issuer master key: Data authentication code (maps to TR31 Create rule array keyword "EMVDA-F").
"F4"	EMV/chip issuer master key: Dynamic numbers (maps to TR31 Create rule array keyword "EMVDN-F").
"I0"	Initialization vector (maps to TR31 Export rule array keyword "INITVEC").

Table 627. Key usage values and meanings (continued)	
ASCII value	**Meaning**
"K0"	Key encryption or wrapping key (maps to TR-31 Translate rule array keyword "KEK").
"K1"	TR-31 key block protection key (maps TR-31 Translate rule array keyword "KEKWRAP"). The note for key usage "K0" also applies to this key usage.
"K4"	ISO 20038 key block protection key (maps to TR-31 Translate rule array keyword "KEK-WRK4"). The note for key usage "K0" also applies to this key usage.
"M0"	ISO 16609 MAC algorithm 1 (using TDEA) key (maps to TR-31 Translate rule array keyword "ISOMAC0").
"M1"	ISO 9797-1 MAC algorithm 1 (maps to TR-31 Translate rule array keyword "ISOMAC1"). The note for key usage "M0" also applies to this key usage.
"M3"	ISO 9797-1 MAC algorithm 3 (maps to TR-31 Translate rule array keyword "ISOMAC3"). The note for key usage "M0" also applies to this key usage.
"M6"	ISO 9797-1:2011 MAC algorithm 5/CMAC (maps to TR-31 Translate rule array keyword "ISOMAC6", Release 5.4 or later). These keys are used to compute or verify a code for message authentication.
"M7"	HMAC (maps to TR-31 Translate rule array keyword "HMAC", Release 5.6 or later). These keys are used to compute or verify a code for message authentication.
"P0"	PIN encryption key (maps to TR-31 Translate rule array keyword "PINENC"). These keys are used to protect PIN blocks.
"P2"	PIN generation key (maps to TR-31 Create rule array keyword "PINGEN"). These keys are used to generate PINs.
"V0"	PIN verification, KPV, other algorithm key (maps to TR31 Key Create rule array keyword "PINVO").
"V1"	PIN verification, IBM 3624 key (maps to TR31 Key Create rule array keyword "PINV3624").
"V2"	PIN verification, Visa PVV key (maps to TR31 Key Create rule array keyword "VISAPVV").

X9.143 (TR-31) optional block data defined by IBM

As defined by ANSI X9.143 and ASC X9 TR-31, a TR-31 key block can contain one or more optional blocks. A TR-31 key block contains at least one optional block when bytes 13 - 14 is a value other than ASCII "00".

The data of an IBM-defined optional block contains ASCII string "10" in the first two bytes and contains ASCII string "IBMC" beginning at offset 4 of the data. ICSF treats an optional block with these characteristics as a proprietary container. See Table 628 on page 1544 for details. An optional block with different characteristics is ignored.

If a TR-31 key block contains an optional block as defined by Table 628 on page 1544, the data contains either a tag-length-value (TLV) ID of '01' or '02'.

For TLV ID '01'

A copy of the 8-byte or 16-byte DES control vector that was in the CCA key-token of the key being exported. The copied control vector is in hex-ASCII format ("0"-"9", "A"-"F").

The control vector is only copied from the CCA key-token when the user of the TR-31 Translate service specifies a control vector transport control keyword (INCL-CV or ATTR-CV):

- If the optional block contains a control vector as the result of specifying the INCL-CV keyword during export, the key usage and mode of use fields indicate the key attributes, and these attributes are verified during export to be compatible with the ones in the included control vector.
- If the optional block contains a control vector as the result of specifying the ATTR-CV keyword during export, the key usage field (byte number 5-6 of the TR-31 key block) is set to the proprietary value "10", and the mode of use field (byte number 8) is set to the proprietary value "1". These proprietary values indicate that the key attributes are specified in the included control vector.

For TLV ID '02'

The data contains the IBM Internal X9-SWKB controls.

For additional information on how CCA uses an IBM-defined optional block in a TR-31 key block, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Table 628. IBM optional block data in a TR-31 key block, control vector (ID "10")

Offset (bytes)	Length (bytes)	Description
00	02	Proprietary ID of TR-31 optional block (alphanumeric-ASCII). **ASCII value** **Meaning** **"10"** IBM proprietary optional block.
02	02	Length of optional block (hex-ASCII). For TLV valued to "01": **ASCII value** **Meaning** **"1C"** 8-byte (single-length) control vector. **"2C"** 16-byte (double-length) control vector. For TLV valued to "02": **ASCII value** **Meaning** **"24"** 12-byte IBM internal controls.
Beginning of optional block data		

Table 628. IBM optional block data in a TR-31 key block, control vector (ID "10") (continued)

Offset (bytes)	Length (bytes)	Description
04	04	"Magic" value (alphanumeric-ASCII). **ASCII value** **Meaning** **"IBMC"** A constant value used to reduce ambiguity and the chance for false interpretation of the proprietary optional blocks of non-IBM vendors. An optional block that uses the same proprietary ID, but does not include this magic value will be ignored.
08	02	Tag-length-value (TLV) ID (numeric-ASCII). **ASCII value** **Meaning** **"01"** IBM CCA control vector. **"02"** IBM Internal X9-SWKB controls.
10	02	Length of TLV (hex-ASCII). For TLV valued to "01": **ASCII value** **Meaning** **"14"** Length of TLV for 8-byte control vector (decimal 20). **"24"** Length of TLV for 16-byte control vector (decimal 36). For TLV valued to "02": **ASCII value** **Meaning** **"1C"** Length of TLV for 12-byte IBM internal controls (decimal 28).
12	16 or 32	**For TLV ID "01":** This should be 16 or 32 bytes long and contain the control vector (hex-ASCII).
	24	**For TLV ID "02":** This should be 24 bytes long and contain the IBM Internal X9-SWKB controls (hex-ASCII). Table 629 on page 1546 documents the 12 bytes of controls after conversion to binary: • Four bytes (12-15) are controls that apply to all key usages. • Four bytes (16-19) are type specific and only used for certain Key Usages. Unused bytes must be set to zero. • Four bytes (20-23) are reserved and must be set to zero.

Table 629. IBM internal X9-SWKB controls (TLV ID '02') after conversion to binary		
Offset (bytes)	**Length (bytes)**	**Description**
12	1	General flag byte applicable for key usages. **Bit** **Meaning when set On** **0-4** Reserved. **5** Compliance Tagged key block. **6** CPACF export allowed. **7** Only usable in DK services, DKPINOP/KUF equivalent.
13	2	Reserved.
15	1	KDF Indicator. **X'00'** No KDF / comp-tag indicator. **X'04'** Comp-tag KDF for TR-31.
16	1	Type specific flags. **Bit** **Meaning when set On** **0** All use cases are allowed. Bits 1-7 must be zero. **1-7** Reserved.
17-19	3	Reserved for future use.
20-23	4	Reserved.

PKA key tokens

PKA key tokens, which hold RSA, ECC, and QSA keys, contain various fields, some of which are optional, and some of which can be present in different forms. The token is composed of concatenated sections that must occur in the prescribed order.

As with other CCA key tokens, both internal and external forms are defined:

- An internal PKA key token contains a private key that is protected by encrypting the private key information using an object protection key (OPK) that is encrypted by the RSA or ECC master key. The internal key token also contains the modulus and the public-key exponent. A master key verification pattern is also included to enable determination that the proper master key is available to process the protected private key.

  Note: The format and content of an internal key token is local to a specific node and product implementation and does not represent an interchange format.
- An external PKA key token contains the public key components of the key. Also, the external key token optionally contains the private key. If the private key is present, it is either in the clear or it might be protected by encryption using an AES or DES transport key. An external key token is an inter-product interchange data structure.

An RSA private key can be represented in one of two forms:

- By a modulus and the private key exponent.
- By a set of numbers used in the Chinese Remainder Theorem (CRT). ICSF always generates a CRT key with p > q. If you import a CRT key from another RSA implementation with q > p, the key is usable within the coprocessor, but your application may encounter a performance degradation with each use of the key.

PKA key token sections

A PKA key token is the concatenation of an ordered set of sections. Table 630 on page 1547 describes the key token section data structures and the references that provide details.

Table 630. PKA key token section data structures

Section	Reference	Usage
Header	Table 631 on page 1549	PKA key token header.
X'02'	Table 633 on page 1550	RSA private key, 1024-bit Modulus-Exponent format. External format. Note: The internal format is deprecated.
X'04'	Table 639 on page 1565	RSA public key.
X'06'	Table 634 on page 1551	RSA private key, 1024-bit Modulus-Exponent format with OPK. Internal format.
X'08'	Table 637 on page 1559	RSA private key, 4096-bit Chinese-Remainder Theorem format with OPK. Internal and external format.
X'09'	Table 636 on page 1557	RSA private key, 4096-bit Modulus-Exponent format. Generated for external format for clear keys or for keys encrypted by a DES key-encrypting key.
X'10'	Table 640 on page 1566	Private-key name for RSA and QSA keys.
X'20'	Table 645 on page 1567	ECC private key format with OPK. Internal and external format.
X'21'	Table 646 on page 1572	ECC public key.
X'23'	Table 648 on page 1574	ECC key-derivation.
X'30'	Table 635 on page 1552	RSA private key, 4096-bit Modulus-Exponent format with AES-encrypted OPK. Internal and external format.
X'31'	Table 638 on page 1561	RSA private key, 4096-bit Chinese-Remainder Theorem format with AES-encrypted OPK. Internal and external format.
X'50'	Table 650 on page 1575	QSA Private key with OPK.
X'51'	Table 651 on page 1581	QSA Public key.

PKA key tokens can be built with the PKA Key Token Build service (CSNDPKB and CSNFPKB).

An RSA key token is the concatenation of these sections:

- A token header:
  - An external header (first byte X'1E').
  - An internal header (first byte X'1F').
- An optional private key section in one of these formats:
  - Section identifier X'02' for a Modulus-Exponent format key 512 - 1024 bits, either in an external key token in the clear or wrapped by a DES key-encrypting key, or in an internal key token wrapped by the RSA master key.

    Note: The use of the internal format is deprecated.
  - Section identifier X'06' for a Modulus-Exponent format key 512 - 1024 bits, in an internal key token wrapped with an OPK which is wrapped by the RSA master key.
  - Section identifier X'30' for a Modulus-Exponent format key 512 - 4096 bits, either in an external key token in the clear or wrapped with an OPK which is wrapped by an AES key-encrypting key, or in an internal key token wrapped with an OPK which is wrapped by the ECC master key.
  - Section identifier X'09' for a Modulus-Exponent format key 512 - 4096 bits, in an external key token in the clear or wrapped by a DES key-encrypting key.
  - Section identifier X'08' for a Chinese-Remainder Theorem format key 512 - 4096 bits, either in an external key token in the clear or wrapped by a DES key-encrypting key, or in an internal key token wrapped by the RSA master key.
  - Section identifier X'31' for a Chinese-Remainder Theorem format key 512 - 4096 bits, either in an external key token in the clear or wrapped with an OPK which is wrapped by an AES key-encrypting key, or in an internal key token wrapped with an OPK which is wrapped by the ECC master-key.
- A public key section (section identifier X'04').
- An optional private key name section (section identifier X'10').

Special note for CRT keys: ICSF always generates a CRT key with p > q. If you import a CRT key from another RSA implementation with q > p, the key is usable within the coprocessor or accelerator, but your application encounters a performance degradation with each use of the key.

An ECC key token is the concatenation of these sections:

- A token header:
  - An external header (first byte X'1E').
  - An internal header (first byte X'1F').
- An optional private key section (section identifier X'20').
- A public key section (section identifier X'21').
- An optional key-derivation section (section identifier X'23').

Integrity of PKA private key sections containing an encrypted RSA key

With the exception of PKA key tokens containing an AES-encrypted OPK (sections X'30' and X'31'), if a PKA key token contains information for an encrypted RSA private key, then the integrity of the information within the token can be verified by computing and comparing the SHA-1 message digest values that are found at offsets 4 and 30 within the private key section.

The SHA-1 message digest at offset 4 requires access to the cleartext values of the private-key components. The cryptographic engine verifies this hash quantity whenever it retrieves the secret key for productive use.

A second SHA-1 message digest, located at offset 30 (excluding sections X'30' and X'31'), is computed on optional, designated key token information following the public key section. The value of this SHA-1 message digest is included in the computation of the message digest at offset 4. As with the offset 4

value, the message digest at offset 30 is validated whenever a private key is recovered from the token for productive use.

PKA private key sections X'30' and X'31' can contain an AESKW-wrapped (ANS X9.102) RSA key and an AES-encrypted OPK. When the RSA key is wrapped, a message digest is calculated over the associated data section contained in the private key section. The calculated message digest becomes part of the payload before it is wrapped. A user is not able to retrieve this value to validate it.

In addition to the hash checks, various token-format and content checks are performed to validate the key values.

The optional private key name section can be used by access monitor systems (for example, Security Server RACF) to ensure that the application program is entitled to employ the particular private key.

Number representation in PKA key tokens

- All length fields are in binary.
- All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address); the value is right-justified and padded with zeros to the left.
- Values in the offset and length columns are decimal. Values in the description column are decimal unless otherwise noted.

Table 631. PKA key token header

Offset (bytes)	Length (bytes)	Description
00	01	Token identifier (a flag that indicates token type): **X'00'** Null token. **X'1E'** External token. The optional private-key is either in cleartext or enciphered by a transport key-encrypting key. **X'1F'** Internal token. The private key is enciphered by the master key.
01	01	Token version number: X'00'.
02	02	Length in bytes of the token structure.
04	04	Reserved, binary zero.

Table 632. Null PKA key token format

Offset (bytes)	Length (bytes)	Description
00	01	Token identifier: **X'00'** Null key token.
01	01	Token version number: X'00'.
02	02	**X'0008'** Length indicates a null PKA key token.
04	04	Reserved, binary zero.

Table 633. RSA private key, 1024-bit Modulus-Exponent format section (X'02')		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'02'** RSA private key, modulus-exponent format (RSA-PRIV).
001	001	Section version number: X'00'.
002	002	Section length in bytes: 364.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved, binary zero.
028	001	Key format and security flag: External token: **X'00'** Unencrypted ME RSA private key subsection identifier. **X'82'** Encrypted ME RSA private key subsection identifier.
029	001	Reserved, binary zero.
030	020	SHA-1 hash of all optional sections that follow the public-key section, if any. Otherwise, 20 bytes of binary zero.
050	004	Key-usage and translation control flag: Key usage: **B'11xx xxxx'** Only key unwrapping (KM-ONLY). **B'10xx xxxx'** Both signature generation and key unwrapping (KEY-MGMT). **B'01xx xxxx'** Undefined. **B'00xx xxxx'** Only signature generation (SIG-ONLY). Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). All other bits are reserved and must be zero.
054	006	Reserved, binary zero.
060	024	Reserved, binary zero.
		Start of the optionally-encrypted secure subsection.
084	024	Random number, confounder.
108	128	Private-key exponent, d. $d=e^{-1} \bmod((p-1)(q-1))$, and $1<d<n$ where *e* is the public exponent.

Table 633. RSA private key, 1024-bit Modulus-Exponent format section (X'02') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
		End of the optionally-encrypted subsection. The confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.
236	128	Modulus, n. n=pq where p and q are prime and $1<n<2^{1024}$.

Table 634. RSA private key, 1024-bit Modulus-Exponent format with OPK section (X'06')		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'06'** RSA private key, modulus-exponent format (RSA-PRIV).
001	001	Section version number: X'00'.
002	002	Section length in bytes: 408.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to and including the modulus at offset 236.
024	004	Reserved, binary zero.
028	001	Key format and securityflag: Internal token: **X'02'** Encrypted ME RSA private key with OPK subsection identifier.
029	001	Key source flag: **X'21'** External private key was specified in the clear. **X'22'** External private key was encrypted. **X'23'** Private key was generated using regeneration data. **X'24'** Private key was randomly generated. All other values are reserved and undefined.
030	020	SHA-1 hash of all optional sections that follow the public-key section, if any. Otherwise, 20 bytes of binary zero.

Table 634. RSA private key, 1024-bit Modulus-Exponent format with OPK section (X'06') (continued)

Offset (bytes)	Length (bytes)	Description
050	004	Key-usage and translation control flag: Key usage: **B'11xx xxxx'** Only key unwrapping (KM-ONLY). **B'10xx xxxx'** Both signature generation and key unwrapping (KEY-MGMT). **B'01xx xxxx'** Undefined. **B'00xx xxxx'** Only signature generation (SIG-ONLY). Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). All other bits are reserved and must be zero.
054	006	Reserved, binary zero.
060	048	Object Protection Key (OPK) data. External key-token: Reserved, binary zero. Internal key-token: The OPK consists of a 8-byte confounder, three 8-byte DES keys, and two 8-byte initialization vector values. Encrypted under the RSA-MK using the ede3 algorithm.
108	128	Private key exponent d, encrypted under the OPK using the ede5 algorithm. $d=e^{-1} \bmod ((p-1)(q-1))$, and $1<d<n$ where e is the public exponent.
236	128	Modulus, n. $n=pq$ where p and q are prime and $2^{512}<n<2^{1024}$.
364	016	RSA master key verification pattern.
380	020	SHA-1 hash value of the subsection cleartext, offset 400 to the end of the section.
400	002	Reserved, binary zero.
402	002	Reserved, binary zero.
404	002	Reserved, binary zero.
406	002	Reserved, binary zero.

Table 635. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30')

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'30'** RSA private key, ME format with AES encrypted OPK.

Table 635. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') (continued)

Offset (bytes)	Length (bytes)	Description
001	001	Section version number: X'00'.
002	002	Section length in bytes: 122 + *nnn* + *ppp*.
004	002	Length of associated data section: 46.
006	002	Length of payload data: *ppp*.
008	002	Reserved, binary zero.
		Start of associated data section
010	001	Associated data section version: X'02' or X'04'.
011	001	Key format and security flag: External token: **X'00'** Unencrypted ME RSA private-key subsection identifier. **X'82'** Encrypted ME RSA private-key subsection identifier. Internal token: **X'02'** Encrypted ME RSA private-key subsection identifier. All other values are reserved and undefined.
012	001	Key source flag: External token: Reserved, binary zero. Internal token: **X'21'** External private key was specified in the clear. **X'22'** External private key was encrypted. **X'23'** Private key was generated using regeneration data. **X'24'** Private key was randomly generated. All other values are reserved and undefined.

Table 635. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') (continued)

Offset (bytes)	Length (bytes)	Description
013	001	When associated data section version is X'02': Reserved, binary zero. When associated data section version is X'04': Compliance and export control byte. **Bit** **Meaning** **B'1xxx xxxx'** Compliant-tagged key. **B'0xxx xxxx'** Non-compliant-tagged key. **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). All other bits are reserved and must be zero.
014	001	Hash type: **X'00'** Clear key. **X'02'** SHA-256.
015	032	When associated data section version is X'02': SHA-256 hash of all optional sections that follow the public key section, if any. Otherwise, 32 bytes of binary zero. When associated data section version is X'04': Hash value of: 1. The public key section (section identifier X'04') and 2. All optional sections that follow the public key section, if any. If there are no optional sections, the hash covers only the public keys section.
047	001	Reserved, binary zero.

Table 635. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') (continued)

Offset (bytes)	Length (bytes)	Description
048	002	When associated data section version is X'02': Reserved, binary zero. When associated data section version is X'04': Usage bytes: • Offset 48: **Bit** **Meaning** **B'1xxx xxxx'** Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D. **B'x1xx xxxx'** Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV. **B'xx1x xxxx'** Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD. **B'xxx1 xxxx'** Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD. **B'xxxx 1xxx'** Key agreement usage is allowed (U-KEYAGR). **B'xxxx x1xx'** keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV. **B'xxxx xx1x'** Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV. **B'xxxx xxx1'** Only encipher operations are allowed during key agreement (U-ENCONL). • Offset 49: **Bit** **Meaning** **B'1xxx xxxx'** Only decipher operations are allowed during key agreement (U-DECONL).

Table 635. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') (continued)

Offset (bytes)	Length (bytes)	Description
050	001	When associated data section version is X'02': Key-usage and translation control flag: Key-usage: **B'11xx xxxx'** Only key unwrapping (KM-ONLY). **B'10xx xxxx'** Both signature generation and key unwrapping (KEY-MGMT). **B'01xx xxxx'** Undefined. **B'00xx xxxx'** Only signature generation (SIG-ONLY). All other values are undefined. Translation control bit: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). All other bits are reserved and must be zero. When associated data section version is X'04': Reserved, binary zero.
051	001	Format restriction byte for digital-signature hash formatting method. Value: **B'0000 0000'** No format restriction. **B'0000 0001'** ISO-9796 only. **B'0000 0010'** PKCS-1.0 only. **B'0000 0011'** PKCS-1.1 only. **B'0000 0100'** PKCS-PSS only. **B'0000 0101'** X9.31 only. **B'0000 0110'** ZERO-PAD only. All other values are reserved and undefined.
052	002	Length of modulus: *nnn* bytes.
054	002	Length of private exponent: *ddd* bytes.
		End of associated data section

Table 635. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') (continued)

Offset (bytes)	Length (bytes)	Description
056	048	Object Protection Key (OPK) Data: The OPK consists of a 16 byte confounder and a 256-bit AES key. External token: The OPK data is wrapped with an AES key-encrypting key using the AESKW (ANS X9.102) algorithm. Internal token: The OPK data is wrapped with an ECC master key using the AESKW algorithm.
104	016	Key verification pattern: External token: • For an encrypted private key, KEK verification pattern (KVP). • For a clear private key, binary zero. • For a skeleton, binary zero. Internal token: • For an encrypted private key, – When a non-compliant-tagged token (bit 0 at offset 13 is not set), the ECC master-key verification pattern (MKVP). – When a compliant-tagged token (bit 0 at offset 13 is set), 5 bytes of the ECC MKVP followed by 3 bytes of internal compliance information. • For a skeleton, binary zero.
120	002	Reserved, binary zero.
122	*nnn*	Modulus, n. n=pq where p and q are prime and 2^{512}<n<2^{4096}.
122+nnn	ppp	Payload starts here and includes: When this section is unencrypted: Clear private exponent d, length *ppp*. When this section is encrypted: Private exponent d in the payload wrapped by the OPK using the AESKW algorithm.

Table 636. RSA private key, 4096-bit Modulus-Exponent format section (X'09')

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'09'** RSA private key, modulus-exponent format (RSAMEVAR).
001	001	Section version number: X'00'.
002	002	Section length in bytes: 132 + *ddd* + *nnn* + *xxx*.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	002	Length of the encrypted private key section 8 + *ddd* + *xxx*.
026	002	Reserved, binary zero.

Table 636. RSA private key, 4096-bit Modulus-Exponent format section (X'09') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
028	001	Key format and security flag: **X'00'** Unencrypted ME RSA private key subsection identifier. **X'82'** Encrypted ME RSA private key subsection identifier. All other values are reserved and undefined.
029	001	Reserved, binary zero.
030	020	SHA-1 hash of all optional sections that follow the public key section, if any. Otherwise, 20 bytes of binary zero.
050	001	Key-usage and translation control flag: Key usage: **B'11xx xxxx'** Only key unwrapping (KM-ONLY). **B'10xx xxxx'** Both signature generation and key unwrapping (KEY-MGMT). **B'01xx xxxx'** Undefined. **B'00xx xxxx'** Only signature generation (SIG-ONLY). Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). All other bits are reserved and must be zero.
051	001	Reserved, binary zero.
052	048	Reserved, binary zero.
100	016	Reserved, binary zero.
116	002	Length of private exponent, d, in bytes: *ddd*.
118	002	Length of modulus, n, in bytes: *nnn*.
120	002	Length of padding field, in bytes: *xxx*.
122	002	Reserved, binary zero.
		Start of the optionally-encrypted subsection.
124	008	Random number, confounder.
132	*ddd*	Private-key exponent, d. $d=e^{-1} \bmod((p-1)(q-1))$, and $1<d<n$ where *e* is the public exponent.
132 + *ddd*	*xxx*	X'00' padding of length *xxx* bytes such that the length from the start of the random number to the end of the padding field is a multiple of eight bytes.

Table 636. RSA private key, 4096-bit Modulus-Exponent format section (X'09') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
		End of the optionally-encrypted subsection. The confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.
132 + *ddd* + *xxx*	*nnn*	Modulus, n. n=pq where p and q are prime and $1<n<2^{4096}$.

Table 637. RSA private key, Chinese-Remainder Theorem format with OPK section (X'08')		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'08'** RSA private key, CRT format (RSA-CRT).
001	001	Section version number: X'00'.
002	002	Section length in bytes: 132 + *ppp* + *qqq* + *rrr* + *sss* + *uuu* + *xxx* + *nnn*.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the end of the modulus.
024	004	Reserved, binary zero.
028	001	Key format and security flag: External token: **X'40'** Unencrypted CRT RSA private-key subsection identifier. **X'42'** Encrypted CRT RSA private-key subsection identifier. Internal token: **X'08'** Encrypted CRT RSA private-key subsection identifier. All other values are reserved and undefined.
029	001	Key source flag: External token: Reserved, binary zero. Internal token: **X'21'** External private key was specified in the clear. **X'22'** External private key was encrypted. **X'23'** Private key was generated using regeneration data. **X'24'** Private key was randomly generated. All other values are reserved and undefined.

Table 637. RSA private key, Chinese-Remainder Theorem format with OPK section (X'08') (continued)

Offset (bytes)	Length (bytes)	Description
030	020	SHA-1 hash of all optional sections that follow the public key section, if any. Otherwise, 20 bytes of binary zero.
050	004	Key-usage and translation control flag: Key usage: **B'11xx xxxx'** Only key unwrapping (KM-ONLY). **B'10xx xxxx'** Both signature generation and key unwrapping (KEY-MGMT). **B'01xx xxxx'** Undefined. **B'00xx xxxx'** Only signature generation (SIG-ONLY). Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). All other bits are reserved and must be zero.
054	002	Length of prime number, p, in bytes: *ppp*.
056	002	Length of prime number, q, in bytes: *qqq*.
058	002	Length of dp, in bytes: *rrr*.
060	002	Length of dq, in bytes: *sss*.
062	002	Length of U, in bytes: *uuu*.
064	002	Length of modulus, n, in bytes: *nnn*.
066	002	Reserved, binary zero.
068	002	Reserved, binary zero.
070	002	Length of padding field, in bytes: *xxx*.
072	004	Reserved, binary zero.
076	016	RSA master key verification pattern.
092	032	Object Protection Key (OPK) data: External token: Reserved, binary zero. Internal token: The OPK consists of a 8-byte confounder and three 8-byte DES keys. The OPK is encrypted under the RSA master key using the Triple-DES CBC process.

Table 637. RSA private key, Chinese-Remainder Theorem format with OPK section (X'08') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
124		**Start of the optionally encrypted subsection.** External token: • When offset 028 is X'40', the subsection is not encrypted. • When offset 028 is X'42', the subsection is encrypted by the double-length transport key using the Triple-DES CBC process. Internal token: • When offset 028 is X'08', the subsection is encrypted by the OPK using the Triple-DES CBC process.
124	008	Random number, confounder.
132	*ppp*	Prime number, p.
132 + *ppp*	*qqq*	Prime number, q.
132 + *ppp* + *qqq*	*rrr*	dp = d mod(p - 1)
132 + *ppp* + *qqq* + *rrr*	*sss*	dq = d mod(q - 1)
132 + *ppp* + *qqq* + *rrr* + *sss*	*uuu*	$U = q^{-1} \bmod(p)$.
132 + *ppp* + *qqq* + *rrr* + *sss* + *uuu*	*xxx*	X'00' padding of length *xxx* bytes such that the length from the start of the confounder at offset 124 to the end of the padding field is a multiple of eight bytes.
		End of the optionally encrypted subsection. External token: All of the fields starting with the confounder field and ending with the variable length pad field are enciphered for key confidentiality when the key format-and-security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the TDES (CBC outer chaining) algorithm. Internal token: All of the fields starting with the confounder field and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) for key confidentiality.
132 + *ppp* + *qqq* + *rrr* + *sss* + *uuu* + *xxx*	*nnn*	Modulus, n. n = pq where p and q are prime and $2^{512} < n < 2^{4096}$.

Table 638. RSA private key, 4096-bit Chinese-Remainder Theorem format with AES-encrypted OPK section (X'31')		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'31'** RSA private key, CRT format with AES encrypted OPK.
001	001	Section version number: X'00'.
002	002	Section length: 134 + *nnn* + *xxx*.

Table 638. RSA private key, 4096-bit Chinese-Remainder Theorem format with AES-encrypted OPK section (X'31') (continued)

Offset (bytes)	Length (bytes)	Description
004	002	Length of associated data section.
006	002	Length of payload data: *xxx*.
008	002	Reserved, binary zero.
		Start of associated data section.
010	001	Associated data section version: X'03' or X'05'.
011	001	Key format and security flag: External token: **X'40'** Unencrypted CRT RSA private-key subsection identifier. **X'42'** Encrypted CRT RSA private-key subsection identifier. Internal token: **X'08'** Encrypted CRT RSA private-key subsection identifier. All other values are reserved and undefined.
012	001	Key source flag: External token: Reserved, binary zero. Internal key-token: **X'21'** External private key was specified in the clear. **X'22'** External private key was encrypted. **X'23'** Private key was generated using regeneration data. **X'24'** Private key was randomly generated. All other values are reserved and undefined.
013	001	When associated data section version is X'03': Reserved, binary zero. When associated data section version is X'05': Compliance and export control byte. **Bit** **Meaning** **B'1xxx xxxx'** Compliant-tagged key. **B'0xxx xxxx'** Non-compliant-tagged key. **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). All other bits are reserved and must be zero.

Table 638. RSA private key, 4096-bit Chinese-Remainder Theorem format with AES-encrypted OPK section (X'31') (continued)

Offset (bytes)	Length (bytes)	Description
014	001	Hash type: **X'00'** Clear key. **X'02'** SHA-256.
015	032	When associated data section version is X'03': SHA-256 hash of all optional sections that follow the public key section, if any. Otherwise, 32 bytes of binary zero. When associated data section version is X'05': Hash value of: 1. The public key section (section identifier X'04') and 2. All optional sections that follow the public key section, if any. If there are no optional sections, the hash covers only the public keys section.
047	001	Reserved, binary zero.
048	002	When associated data section version is X'03': Reserved, binary zero. When associated data section version is X'05': Usage bytes: • Offset 48: **Bit** **Meaning** **B'1xxx xxxx'** Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D. **B'x1xx xxxx'** Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV. **B'xx1x xxxx'** Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD. **B'xxx1 xxxx'** Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD. **B'xxxx 1xxx'** Key agreement usage is allowed (U-KEYAGR). **B'xxxx x1xx'** keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV. **B'xxxx xx1x'** Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV. **B'xxxx xxx1'** Only encipher operations are allowed during key agreement (U-ENCONL). • Offset 49: **Bit** **Meaning** **B'1xxx xxxx'** Only decipher operations are allowed during key agreement (U-DECONL).

Table 638. RSA private key, 4096-bit Chinese-Remainder Theorem format with AES-encrypted OPK section (X'31') (continued)

Offset (bytes)	Length (bytes)	Description
050	001	When associated data section version is X'03': Key-usage and translation control flag. Key-usage flag: **B'11xx xxxx'** Only key unwrapping (KM-ONLY). **B'10xx xxxx'** Both signature generation and key unwrapping (KEY-MGMT). **B'01xx xxxx'** Undefined. **B'00xx xxxx'** Only signature generation (SIG-ONLY). All other values are undefined. Translation control bit: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). All other values are reserved and undefined. When associated data section version is X'05': Reserved, binary zero.
051	001	Format restriction byte for digital-signature hash formatting method. Value: **B'0000 0000'** No format restriction. **B'0000 0001'** ISO-9796 only. **B'0000 0010'** PKCS-1.0 only. **B'0000 0011'** PKCS-1.1 only. **B'0000 0100'** PKCS-PSS only. **B'0000 0101'** X9.31 only. **B'0000 0110'** ZERO-PAD only. All other values are reserved and undefined.
052	002	Length of the prime number, p, in bytes: *ppp*.
054	002	Length of the prime number, q, in bytes: *qqq*.
056	002	Length of dp: *rrr*.
058	002	Length of dq: *sss*.
060	002	Length of U: *uuu*.
062	002	Length of modulus, *nnn*.
064	004	Reserved, binary zero.
		End of associated data section.

Table 638. RSA private key, 4096-bit Chinese-Remainder Theorem format with AES-encrypted OPK section (X'31') (continued)

Offset (bytes)	Length (bytes)	Description
068	048	Object Protection Key (OPK) data: The OPK consists of a 16-byte confounder and a 256-bit AES key. External token: Encrypted with an AES KEK. Internal token: Encrypted with the ECC master key.
116	016	Key verification pattern: External token: • For an encrypted private key, KEK verification pattern (KVP). • For a clear private key, binary zero. • For a skeleton, binary zero. Internal token: • For an encrypted private key, – When a non-compliant-tagged token (bit 0 at offset 13 is not set), the ECC master-key verification pattern (MKVP). – When a compliant-tagged token (bit 0 at offset 13 is set), 5 bytes of the ECC MKVP followed by 3 bytes of internal compliance information. • For a skeleton, binary zero.
132	002	Reserved, binary zero.
134	*nnn*	Modulus, n. n = pq where p and q are prime and $2^{512}<n<2^{4096}$.
134 + *nnn*	*xxx*	Payload starts here and includes: When this section is unencrypted: • Clear prime number p. • Clear prime number q. • Clear dp. • Clear dq. • Clear U. • Length *xxx* bytes: *ppp* + *qqq* + *rrr* + *sss* + *uuu* When this section is encrypted: • Private key values in the payload are wrapped by the OPK using the AESKW algorithm.

Table 639. RSA public-key section (X'04')

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'04'** RSA public key.
001	001	Section version number: X'00'.
002	002	Section length, 12 + *xxx* + *yyy*.
004	002	Reserved, binary zero.
006	002	RSA public key exponent field length in bytes, *xxx*.
008	002	Public key modulus length in bits.

Table 639. RSA public-key section (X'04') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
010	002	RSA public key modulus field length in bytes, *yyy*.
012	*xxx*	Public key exponent (this is generally a 1-byte, 3-byte, or 64-byte to 512-byte quantity), e. e must be odd and 1<e<n. (Frequently, the value of e is $2^{16}+1$)
12 + *xxx*	*yyy*	Modulus, n. n = *pq* where *p* and *q* are prime and $2^{512}<n<2^{4096}$. This field is absent when the modulus is contained in the private-key section. If present, the field length is 64 - 512 bytes.

Table 640. PKA private-key name section for RSA and QSA keys (X'10')		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'10'** Private-key name.
001	001	Section version number: X'00'.
002	002	Section length in bytes: 68.
004	064	Private-key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.

The types of supported elliptic curves when generating an ECC key are:

EC curve type
Corresponding table of supported elliptic curves with object identifier (OID).

Brainpool
Table 641 on page 1566

NIST Prime
Table 642 on page 1567

Edwards
Table 643 on page 1567

Koblitz
Table 644 on page 1567

Table 641. ECC supported Brainpool elliptic curves by size, name, and object identifier		
Size of prime *p* in bits (key length)	**OID in dot notation**	**Brainpool elliptic curve ID**
160	1.3.36.3.3.2.8.1.1.1	brainpoolP160r1
192	1.3.36.3.3.2.8.1.1.3	brainpoolP192r1
224	1.3.36.3.3.2.8.1.1.5	brainpoolP224r1
256	1.3.36.3.3.2.8.1.1.7	brainpoolP256r1
320	1.3.36.3.3.2.8.1.1.9	brainpoolP320r1
384	1.3.36.3.3.2.8.1.1.11	brainpoolP384r1
512	1.3.36.3.3.2.8.1.1.13	brainpoolP512r1

Table 642. ECC supported Prime elliptic curves by size, name, and object identifier				
Size of prime *p* in bits (key length)	**OID in dot notation**	**ANS X9.62 ECDSA prime curve ID**	**NIST-recommended elliptic curve ID**	**SEC 2 recommended elliptic curve domain parameter**
192	1.2.840.10045.3.1.1	prime192v1	P-192	secp192r1
224	1.3.132.0.33	N/A	P-224	secp224r1
256	1.2.840.10045.3.1.7	prime256v1	P-256	secp256r1
384	1.3.132.0.34	N/A	P-384	secp384r1
521	1.3.132.0.35	N/A	P-521	secp521r1

Table 643. EC supported Edwards elliptic curves			
Size of prime *p* in bits (key length)	**OID in dot notation**	**Edwards-coordinate signature system**	**Elliptic curve**
255	1.3.101.112	id-Ed25519	Curve25519
224	1.3.101.113	id-Ed448	Curve448

Table 644. ECC supported Koblitz elliptic curves by size, name, and object identifier			
Size of prime p in bits (key length)	**OID in dot notation**	**ANS X9.62 curve name**	**SEC 2 recommended elliptic curve domain parameter**
256	1.3.132.0.1	Koblitz curve over 256-bit Prime field	secp256k1

An ECC key token is the concatenation of these sections:

- A token header:
  - An external header (first byte X'1E').
  - An internal header (first byte X'1F').
- An optional private key section (section identifier X'20').
- A public key section (section identifier X'21').
- An optional key-derivation section (section identifier X'23').

Table 645. ECC private-key section (X'20')		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'20'** ECC private key.
001	001	Section version number: **X'00'** or **X'01'**
002	002	Section length in bytes: 76 + *asdl* + *fsl*.

Table 645. ECC private-key section (X'20') (continued)

Offset (bytes)	Length (bytes)	Description
004	001	Encrypted section wrapping method: **X'00'** Clear - section is unencrypted. **X'01'** AESKW. **X'02'** CBC Wrap - Other.
005	001	Hash used for wrapping: **X'01'** SHA224. **X'02'** SHA256.
006	002	Reserved, binary zero.
008	001	Key-usage and translation control flag: Management of symmetric keys and generation of digital signatures: **B'11xx xxxx'** Only key establishment (KM-ONLY). **B'10xx xxxx'** Both signature generation and key establishment (KEY-MGMT). **B'01xx xxxx'** Undefined. **B'00xx xxxx'** Only signature generation (SIG-ONLY). AESKW export control: **B'xxxx x1xx'** Private key export under AES key is allowed (AES1ECOK). **B'xxxx x0xx'** Private key export under AES key is not allowed (NOAES1EC). Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). Key management: **B'xxxx xxx1'** Private key CPACF-export is allowed (XPRTCPAC). **B'xxxx xxx0'** Private key CPACF-export is not allowed (NOEXCPAC). All other bits are reserved and must be zero.

Table 645. ECC private-key section (X'20') (continued)

Offset (bytes)	Length (bytes)	Description
009	001	Curve type: **X'00'** Prime curve. **X'01'** Brainpool curve. **X'02'** Edwards curve. **X'03'** Koblitz curve.
010	001	Key format and security flag: External token: **X'40'** Unencrypted ECC private key subsection identifier. **X'42'** Encrypted ECC private key subsection identifier. Internal Token: **X'08'** Encrypted ECC private key subsection identifier. All other values are reserved and undefined.
011	001	Pedigree / Key source flag byte: **Version '00'** Reserved, binary zero. **Version '01'** External key-token: **X'00'** None / Clear **X'24'** Randomly generated Internal key-token: **X'00'** None / Clear **X'21'** Imported from cleartext **X'22'** Imported from ciphertext **X'24'** Randomly generated

Table 645. ECC private-key section (X'20') (continued)

Offset (bytes)	Length (bytes)	Description
012	002	Length of *p* in bits: **X'00C0'** Prime P-192. **X'00E0'** Prime P-224. **X'0100'** Prime P-256. **X'0180'** Prime P-384. **X'0209'** Prime P-521. **X'00A0'** Brainpool p-160. **X'00C0'** Brainpool P-192. **X'00E0'** Brainpool P-224. **X'0100'** Brainpool P-256. **X'0140'** Brainpool P-320. **X'0180'** Brainpool P-384. **X'0200'** Brainpool P-512. **X'00FF'** Edwards 25519. **X'01C0'** Edwards 448. **X'0100'** Koblitz P-256.
014	002	Length of the IBM associated data section, 0 or *iadl* = 16 + *pknl* + *ieadl*.
016	008	Key verification pattern: External token: • For an encrypted private key, KEK verification pattern (KVP). • For a clear private key, binary zero. • For a skeleton, binary zero. Internal token: • For encrypted private key, master-key verification pattern (MKVP). • For a skeleton, binary zero.

Table 645. ECC private-key section (X'20') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
024	048	Object Protection Key (OPK) data: External key-token: Reserved, binary zero. Internal key-token: The OPK consists of an 8-byte integrity check value (ICV) and length indicators, an 8-byte confounder, and a 256-bit AES key used with the AESKW algorithm to encrypt the ECC private key contained in an AESKW formatted section. The OPK is encrypted by the ECC master key using the AESKW algorithm.
072	002	Length of the associated data section, 0 or *asdl* = *iadl* + *uadl*.
074	002	Length of formatted section in bytes, or *fsl*.
		Start of the associated data section.
		Start of the IBM associated data section.
076	001	Associated data section version (**X'00'** or **X'01'**). Includes IBM associated data and user-definable associated data.
077	001	Length in bytes of the optional private-key name: *pknl* (0-64).
078	002	Length in bytes of the IBM associated data, including key label and IBM extended associated data (≥ 16).
080	002	Length in bytes of the IBM extended associated data. Associated data section version: **'00'** 0 **'01'** 36
082	001	Length in bytes of the user-definable associated data: *uad* (0 - 155).
083	001	Curve type (see offset 009).
084	002	Length of *p* in bits (see offset 012).
086	001	Key-usage flag byte (see offset 008).
087	001	Key format and security flag byte (see offset 010).
088	004	Depends on associated data section version at offset 76: **'00'** Reserved, binary zero. **'01'** Pedigree / Key source flag byte (see offset 011).
092	*pknl*	Optional private-key name. Private key name (in ASCII), left-justified, padded with space characters (X'20').
092 + *pknl*	*ieadl*	Optional IBM extended associated data.
		End of the IBM associated data section.
092+ *pknl* + *ieadl*	*uadl*	Optional user-definable associated data.

Table 645. ECC private-key section (X'20') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
		End of associated data section.
092+ *pknl* + *ieadl* + *uadl*	*fsl*	Formatted section (payload), which includes private key d: • Clear-key section contains d. • Encrypted-key section contains d within the AESKW-wrapped payload.

Table 646. ECC public-key section (X'21')		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'21'** ECC public key.
001	001	Section version number: X'00'.
002	002	Section length in bytes: 14 + *ccc*
004	004	Reserved, binary zero.
008	001	Curve type: **X'00'** Prime curve. **X'01'** Brainpool curve. **X'02'** Edwards curve. **X'03'** Koblitz curve.
009	001	Reserved, binary zero.

Table 646. ECC public-key section (X'21') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
010	002	Length of *p* in bits: **X'00C0'** Prime P-192. **X'00E0'** Prime P-224. **X'0100'** Prime P-256. **X'0180'** Prime P-384. **X'0209'** Prime P-521. **X'00A0'** Brainpool P-160. **X'00C0'** Brainpool P-192. **X'00E0'** Brainpool P-224. **X'0100'** Brainpool P-256. **X'0140'** Brainpool P-320. **X'0180'** Brainpool P-384. **X'0200'** Brainpool P-512. **X'00FF'** Edwards 25519. **X'01C0'** Edwards 448. **X'0100'** Koblitz P-256.
012	002	The length of the public key *q* in bytes (*ccc*).
014	*ccc*	Public key, *q* field. For Prime, Brainpool, or Koblitz, this must be an uncompressed point (prefixed with 0x04). For Edwards, this is the 'public key A', as described in RFC 8032.

IBM extended associated data section

The IBM extended associated data section consists of a series of TLV objects. The format of the TLV objects is shown in Table 647 on page 1574.

Tag 'X'60' object contains the 32 byte SHA-256 hash of all of the optional sections concatenated to an ECC key-token. The object is described in Table 647 on page 1574.

Table 647. IBM extended associated data section TLV object		
Offset	**Length (bytes)**	**Description**
000	001	Tag identifier: **X'60'** Optional section hash TLV object.
001	001	TLV object version number (X'00').
002	002	TLV object length in bytes (X'0024').
004	032	SHA-256 hash of all the optional sections that follow the public-key section, if any. Otherwise, binary 0.

Table 648. ECC key-derivation section (X'23')		
Offset	**Length (bytes)**	**Description**
000	001	Section identifier: **X'23'** Key-derivation information.
001	001	Section version number: **X'00'**.
002	002	Section length in bytes: 8
004	001	Algorithm of the key to be derived: **X'01'** DES **X'02'** AES
005	001	Key type of the key to be derived: **X'01'** DATA **X'02'** EXPORTER **X'03'** IMPORTER **X'04'** CIPHER **X'05'** DECIPHER **X'06'** ENCIPHER **X'07'** CIPHERXI **X'08'** CIPHERXL **X'09'** CIPHERXO
006	002	Key bit length: 64, 128, 192, 256.

Table 649. Supported CRYSTALS-Dilithium strengths		
Size of matrix (n x n)	**Size of t1 in bits**	**OID in dot notation**
(6 x 5)	13824	1.3.6.1.4.1.2.267.1.6.5

A QSA key token is the concatenation of these sections:

- A token header:
  - An external header (first byte X'1E').
  - An internal header (first byte X'1F').
- An optional private key section (section identifier X’50’).
- A public key section (section identifier X’51’).
- An optional private key name section (section identifier X’10’).

Table 650. QSA Private Key section with OPK (X'50')

Offset (bytes)	**Length (bytes)**	**Description**			
000	001	Section Identifier: **X'50'** QSA Private Key.			
001	001	Section version number: **X'00'**.			
002	002	Section length in bytes: 128 + *ppp*.			
		When 'Key format' (offset 12) is:			
		X'00' (Clear key)			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	*ppp* (in bytes)	Section length (in bytes)
		X'01'	X'0605'	3824	3952
		X'01'	X'0807'	5104	5232
		X'02'	X'1024'	1600	1728
		X'03'	X'0605'	3968	4096
		X'03'	X'0807'	4832	4960
		X'01' (Encrypted key)			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	*ppp* (in bytes)	Section length (in bytes)
		X'01'	X'0605'	3872	4000
		X'01'	X'0807'	5152	5280
		X'02'	X'1024'	1648	1776
		X'03'	X'0605'	4016	4144
		X'03'	X'0807'	4880	5008
004	002	Length of associated data section in bytes: 54.			
006	002	Reserved, binary zero.			

Table 650. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description
008	001	Associated data section version number: X'01'.
009	001	Algorithm identifier: **X'00'** No algorithm. **X'01'** CRYSTALS-Dilithium Round 2. **X'02'** CRYSTALS-Kyber Round 2. **X'03'** CRYSTALS-Dilithium Round 3.
010	002	Algorithm parameters: When 'Algorithm identifier' is X'01', allowed values are: **X'0605'** CRYSTALS-Dilithium (6,5). **X'0807'** CRYSTALS-Dilithium (8,7). When 'Algorithm identifier' is X'02', allowed values are: **X'1024'** CRYSTALS-Kyber (1024). When 'Algorithm identifier' is X'03', allowed values are: **X'0605'** CRYSTALS-Dilithium (6,5). **X'0807'** CRYSTALS-Dilithium (8,7).
012	001	Key format: External token: **X'00'** Unencrypted QSA private key subsection identifier. **X'01'** Encrypted QSA private key subsection identifier. Internal token: **X'01'** Encrypted QSA private key subsection identifier. All other values are reserved and undefined.

Table 650. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description
013	001	Key source flag byte: External token: **X'00'** No Key. **X'24'** Randomly generated. Internal token: **X'00'** No Key. **X'21'** Imported from cleartext. **X'22'** Imported from ciphertext. **X'24'** Randomly generated. All other values are reserved and undefined.
014	001	Compliance and export control byte: **Bit** **Meaning** **B'xxxx x1xx'** Private key export under AES key is allowed (AES1ECOK). **B'xxxx x0xx'** Private key export under AES key is not allowed (NOAES1EC). **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK). **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE). All other values are reserved and undefined.
015	001	Hash type: **X'00'** Clear key. **X'02'** SHA-256.

Table 650. QSA Private Key section with OPK (X'50') (continued)

<table>
<tr><th>Offset (bytes)</th><th>Length (bytes)</th><th>Description</th></tr>
<tr><td>016</td><td>002</td><td>Usage bytes:
Byte 16:
Bit
Meaning
B'1xxx xxxx'
Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV.
B'xx1x xxxx'
Key Encipherment usage is allowed (U-KEYENC).
B'xxx1 xxxx'
Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKD, CSNDPKE.
All other values are reserved and undefined.
Byte 17: All values are reserved and undefined.</td></tr>
<tr><td>018</td><td>032</td><td>SHA-256 hash value of:
1. The public key section (section identifier X'51') and
2. All optional sections that follow the public key section, if any.
If there are no optional sections, the hash will cover only the public key section.</td></tr>
<tr><td>050</td><td>002</td><td>'aaa' = QSA payload component 1 length.
<table>
<tr><th>Algorithm identifier (offset 9)</th><th>Algorithm parameter (offset 10)</th><th>aaa item description</th><th>aaa value</th></tr>
<tr><td>X'01'</td><td>X'0605'</td><td>length of key D, also called 'seed'</td><td>32</td></tr>
<tr><td>X'01'</td><td>X'0807'</td><td>length of key D, also called 'seed'</td><td>32</td></tr>
<tr><td>X'02'</td><td>X'1024'</td><td>length of secret polynomial vector</td><td>1536</td></tr>
<tr><td>X'03'</td><td>X'0605'</td><td>length of key D, also called 'seed'</td><td>32</td></tr>
<tr><td>X'03'</td><td>X'0807'</td><td>length of key D, also called 'seed'</td><td>32</td></tr>
</table></td></tr>
</table>

Table 650. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description			
052	002	'*bbb*' = QSA payload component 2 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	*bbb* item description	*bbb* value
		X'01'	X'0605'	length of tr T : prf output	48
		X'01'	X'0807'	length of tr T : prf output	48
		X'02'	X'1024'	length of public key integrity check	32
		X'03'	X'0605'	length of tr T : prf output	32
		X'03'	X'0807'	length of tr T : prf output	32
054	002	'*ccc*' = QSA payload component 3 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	*ccc* item description	*ccc* value
		X'01'	X'0605'	length of vector 's1', of L elements	480
		X'01'	X'0807'	length of vector 's1', of L elements	672
		X'02'	X'1024'	length of random data	32
		X'03'	X'0605'	length of vector 's1', of L elements	640
		X'03'	X'0807'	length of vector 's1', of L elements	672

Table 650. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description			
056	002	'*ddd*' = QSA payload component 4 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	*ddd* item description	*ddd* value
		X'01'	X'0605'	length of vector 's2', of K elements	576
		X'01'	X'0807'	length of vector 's2', of K elements	768
		X'02'	X'1024'	not used	0
		X'03'	X'0605'	length of vector 's2', of K elements	768
		X'03'	X'0807'	length of vector 's2', of K elements	768
058	002	'*eee*' = QSA payload component 5 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	*eee* item description	*eee* value
		X'01'	X'0605'	length of 't0', K * high bits (vector)	2688
		X'01'	X'0807'	length of 't0', K * high bits (vector)	3584
		X'02'	X'1024'	not used	0
		X'03'	X'0605'	length of 't0', K * high bits (vector)	2496
		X'03'	X'0807'	length of 't0', K * high bits (vector)	3328
060	002	Reserved, binary zero.			
062	056	Object Protection key (OPK) data: The OPK consists of a 16-byte confounder and a 256-bit AES key, followed by 8 bytes of AESKW overhead. External token: • Encrypted: Encrypted by AES KEK. • Clear: All bytes binary zero. Internal token • Encrypted: Encrypted with the ECC master key.			

Table 650. QSA Private Key section with OPK (X'50') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
118	008	Key verification pattern (KVP): When 'Key format' (offset 12) is: **X'00' (Clear key)** Binary zero. **X'01' (Encrypted key)** Bytes 0 – 7: KVP • When internal, this is the ECC master key verification pattern (MKVP). • When external, this is the KVP for the KEK.
126	002	Reserved, binary zero
128	*ppp*	Payload starts here and includes: When this section is encrypted: Private exponent *d* in the payload wrapped by the OPK using the AESKW algorithm.

Table 651. QSA Public Key section (X'51')		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section Identifier: **X'51'** QSA Public Key.
001	001	Section version number: **X'00'**.
002	002	Section length: 24 + *aaa* + *bbb*.
004	001	Key format: **X'00'** No MAC over public key subsection.
005	001	Algorithm identifier: **X'00'** No algorithm. **X'01'** CRYSTALS-Dilithium Round 2. **X'02'** CRYSTALS-Kyber Round 2. **X'03'** CRYSTALS-Dilithium Round 3.

Table 651. QSA Public Key section (X'51') (continued)

Offset (bytes)	Length (bytes)	Description
006	002	Algorithm parameters: When 'Algorithm identifier' is X'01', allowed values are: **X'0605'** CRYSTALS-Dilithium (6,5). **X'0807'** CRYSTALS-Dilithium (8,7). When 'Algorithm identifier' is X'02', allowed values are: **X'1024'** CRYSTALS-Kyber (1024). When 'Algorithm identifier' is X'03', allowed values are: **X'0605'** CRYSTALS-Dilithium (6,5). **X'0807'** CRYSTALS-Dilithium (8,7).
008	002	Usage bytes: Byte 8: **Bit** **Meaning** **B'1xxx xxxx'** Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSV. **B'xx1x xxxx'** Key Encipherment usage is allowed (U-KEYENC). **B'xxx1 xxxx'** Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE. All other values are reserved and undefined. Byte 9: All values are reserved and undefined.

Table 651. QSA Public Key section (X'51') (continued)

Offset (bytes)	Length (bytes)	Description			
010	002	'*aaa*' = QSA public component 1 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	*aaa* item description	*aaa* value
		X'01'	X'0605'	length of 'rho', also called 'nonce'	32
		X'01'	X'0807'	length of 'rho', also called 'nonce'	32
		X'02'	X'1024'	length of public polynomial vector	1536
		X'03'	X'0605'	length of 'rho', also called 'nonce'	32
		X'03'	X'0807'	length of 'rho', also called 'nonce'	32
012	002	'*bbb*' = QSA public component 2 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	*bbb* item description	*bbb* value
		X'01'	X'0605'	length of 't1' : K*low bits (vector)	1728
		X'01'	X'0807'	length of 't1' : K*low bits (vector)	2304
		X'02'	X'1024'	length of public seed	32
		X'03'	X'0605'	length of 't1' : K*low bits (vector)	1920
		X'03'	X'0807'	length of 't1' : K*low bits (vector)	2560
014	010	Reserved, binary zero.			
024	*aaa*	QSA public component 1 When 'Algorithm identifier' is X'01' or X'03' (CRYSTALS-Dilithium): • 'rho' , also called 'nonce'. When 'Algorithm identifier' is X'02' (CRYSTALS-Kyber): • Public polynomial vector.			

Table 651. QSA Public Key section (X'51') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
024 + *aaa*	*bbb*	QSA public component 2 When 'Algorithm identifier' is X'01' or X'03' (CRYSTALS-Dilithium): • 't1' : K * low bits (vector). When 'Algorithm identifier' is X'02' (CRYSTALS-Kyber): • Public seed.

AESKW external format

AES Key Wrap (AESKW) is a symmetric encryption algorithm designed to encapsulate cryptographic key material. CCA, ECC, CRYSTALS-Dilithium, and CRYSTALS-Kyber private key tokens can be exported to AESKW external format wrapped with an AES key-encrypting-key with the PKA Key Translate callable service. CCA private key tokens exported to AESKW external format cannot be written to the PKDS.

Table 652 on page 1584 outlines the structure of the AESKW external format:

Table 652. AESKW external format structure		
Offset	**Size**	**Field**
Start of Associated Data		
	Sub-section: Header	
0	1	Primary Identifier: **X'53'** ASCII 'S'.
1	1	Version: **X'00'** AESKW wrapping method.
2	2	Structure length (SL): Length in bytes of the total structure (big endian).
	Sub-section: Key data	
4	1	Algorithm type: **Value** **Meaning** **X'81'** ECC key (private). **X'82'** QSA CRYSTALS-Dilithium Round 2 (private). **X'83'** QSA CRYSTALS-Kyber Round 2 (private). **X'84'** QSA CRYSTALS-Dilithium Round 3 (private).

Table 652. AESKW external format structure (continued)

Offset	Size	Field
5	2	Key Type: Values in this field depend on the input token and the value at offset X'04', Algorithm type. When input token is an ECC private key: **Encoding:** 1. Upper nibble details curve type. 2. Lower three nibbles details, length of private key ‘p’ in bits. **Values** 1. X'0209' ECC PRIME 521. When input token is a QSA private key: **Encoding:** 1. Values correspond to the algorithm parameter field of the QSA token. **Values** 1. X'0605' CRYSTALS-Dilithium (6,5). 2. X'0807' CRYSTALS-Dilithium (8,7). 3. X'1024' CRYSTALS-Kyber (1024).
7	1	Kuf count: Key usage fields count 0 - 4. Key usage field information defines restrictions on the use of the key. ECC private key token: Kuf count = 2. QSA CRYSTALS-Dilithium: kuf count = 2. QSA CRYSTALS-Kyber: kuf count = 2. **Notes:** • Each *key-usage* field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow. • There are 8 bytes of Associated Data after this field. Each of these remaining 8 bytes of Associated Data is either X'00' or a Kuf byte, as indicated by this kuf count field. This section is not variable size. Unused bytes have an X'00' value. **Examples:** **kuf_count = 1** 2 bytes of key usage information follow, this is 1 kuf field. The remaining 6 bytes of Associated Data are 0x00 bytes. **kuf_count = 2** 4 bytes of key usage information follow, this is 2 kuf fields. The remaining 4 bytes of Associated Data are 0x00 bytes.

Table 652. AESKW external format structure (continued)

Offset	Size	Field
8	1	Bit value meanings: **Bit** **Meaning** **B'1xxx xxxx'** digitalSignature. **B'x1xx xxxx'** nonrepudiation or contentCommitment. **B'xx1x xxxx'** keyEncipherment. **B'xxx1 xxxx'** dataEncipherment. **B'xxxx 1xxx'** keyAgreement. **B'xxxx x1xx'** keyCertSign. **B'xxxx xx1x'** cRLSign. **B'xxxx xxx1'** encipherOnly (requires keyAgreement).
9	1	Bit value meanings: **Bit** **Meaning** **B'1xxx xxxx'** decipherOnly. Requires keyAgreement bit at offset 8. Cannot be combined with encipherOnly. All other values are reserved and undefined.
10	1	This field is reserved and must be X'00' byte.
11	1	This field is reserved and must be X'00' byte.
12	1	This field is reserved and must be X'00' byte.
13	1	This field is reserved and must be X'00' byte.
14	1	This field is reserved and must be X'00' byte.
15	1	This field is reserved and must be X'00' byte.
End of Associated Data		
Start of AESKW wrapped payload		
16	6	Integrity constant: byte array. **Value:** X'A6A6A6A6A6A6'
22	1	PbL: Zero padding bit length. This padding is after the key, at the end of the payload.
23	1	ADLen: Associated Data byte length, in hex. **Value:** X'10'

Table 652. AESKW external format structure (continued)

Offset	Size	Field
24	16	Copy of Associated Data. The Associated Data is copied here. After decryption, this must match the clear data shown above in the 'Associated Data' section.
40	KL	Keydata Format of keydata: **When algorithm type is ECC (X'81'):** The maximum size key, P521, will have KL=66 bytes. **When algorithm type is QSA CRYSTALS-Dilithium Round 2 (X'82'), type (X'0605'):** • 32 B for component 1: key D. • 48 B for component 2: tr T. • 480 B for component 3: vector 's1'. • 576 B component 4: vector 's2'. • 2688 B component 5: 't0'. **When algorithm type is QSA CRYSTALS-Dilithium Round 2 (X'82'), type (X'0807'):** • 32 B for component 1: key D. • 48 B for component 2: tr T. • 672 B for component 3: vector 's1'. • 768 B component 4: vector 's2'. • 3584 B component 5: 't0' . **When algorithm type is QSA CRYSTALS-Kyber Round 2 (X'83'), type (X'1024'):** • 1536 B for component 1: secret polynomial vector. • 32 B for component 2: public key integrity check. • 32 B for component 3: random data. **When algorithm type is QSA CRYSTALS-Dilithium Round 3 (X'84'), type (X'0605'):** • 32 B for component 1: key D. • 32 B for component 2: tr T. • 640 B for component 3: vector 's1'. • 768 B component 4: vector 's2'. • 2496 B component 5: 't0'. **When algorithm type is QSA CRYSTALS-Dilithium Round 3 (X'84'), type (X'0807'):** • 32 B for component 1: key D. • 32 B for component 2: tr T. • 672 B for component 3: vector 's1'. • 768 B component 4: vector 's2'. • 3328 B component 5: 't0'.
KL+40	PbL/ 8	Padding data: PbL count of 0b0 bits.
End of AESKW wrapped payload		

Table 652. AESKW external format structure (continued)		
Offset	**Size**	**Field**
KL+40+ (PbL/8)		Final size (FS). **Algorithm** **Size** **ECC:** 521-bit P521: KL = 66 bytes; PbL = 48 bits; FS = 112 bytes (This is the max for ECC). **QSA CRYSTALS-Dilithium Round 2 (X'82'), type (X'0605'):** KL = 3824; PbL = 0; FS = 3864 bytes. **QSA CRYSTALS-Dilithium Round 2 (X'82'), type (X'0807'):** KL = 5104; PbL = 0; FS = 5144 bytes. **QSA CRYSTALS-Kyber Round 2 (X'83'), type (X'1024'):** KL = 1600; PbL = 0; FS = 1640 bytes. **QSA CRYSTALS-Dilithium Round 3 (X'84'), type (X'0605'):** KL = 3968; PbL = 0; FS = 4008 bytes. **QSA CRYSTALS-Dilithium Round 3 (X'84'), type (X'0807'):** KL = 4832; PbL = 0; FS = 4872 bytes.

Trusted blocks

A trusted block is an extension of CCA PKA key tokens using new section identifiers. They are an integral part of a remote key-loading process.

Trusted blocks contain various items, some of which are optional, and some of which can be present in different forms. Tokens are composed of concatenated sections that, unlike CCA PKA key tokens, occur in no prescribed order.

As with other CCA key-tokens, both internal and external forms are defined:

- An external trusted block contains a randomly generated confounder and a triple-length MAC key enciphered under a DES IMP-PKA transport key. The MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. An external trusted block is created by the Trusted Block Create callable service. This service can:
  1. Create an inactive external trusted block.
  2. Change an external trusted block from inactive to active.
- An internal trusted block contains a confounder and triple-length MAC key enciphered under a variant of the PKA master key. The MAC key is used to calculate a TDES MAC of the trusted block contents. A PKA master key verification pattern is also included to enable determination that the proper master key is available to process the key. The Remote Key Export service only operates on trusted blocks that are internal. An internal trusted block must be imported from an external trusted block that is active using the PKA Key Import service.

  Note: Trusted blocks do not contain a private key section.

Trusted block sections

A trusted block is a concatenation of a header followed by an unordered set of sections. The data structures of these sections are summarized in the following table:

Table 653. Trusted block sections

Section	Reference	Usage
Header	Table 654 on page 1590	Trusted block token header
X'11'	Table 655 on page 1591	Trusted block public key

Table 653. Trusted block sections (continued)

Section	Reference	Usage
X'12'	Table 656 on page 1592	Trusted block rule
X'13'	Table 663 on page 1599	Trusted block name (key label)
X'14'	Table 664 on page 1599	Trusted block information
X'15'	Table 668 on page 1601	Trusted block application-defined data

Every trusted block starts with a token header. The first byte of the token header determines the key form:

- An external header (first byte X'1E'), created by the Trusted Block Create callable service.
- An internal header (first byte X'1F'), imported from an active external trusted block by the PKA Key Import callable service.

Following the token header of a trusted block is an unordered set of sections. A trusted block is formed by concatenating these sections to a trusted block header:

- An optional public-key section (trusted block section identifier X'11')

  The trusted block trusted RSA public-key section includes the key itself in addition to a key-usage flag. No multiple sections are allowed.
- An optional rule section (trusted block section identifier X'12')

  A trusted block may have zero or more rule sections.

  1. A trusted block with no rule sections can be used by the PKA Key Token Change and PKA Key Import callable services. A trusted block with no rule sections can also be used by the Digital Signature Verify callable service, provided there is an RSA public-key section that has its key-usage flag bits set to allow digital signature operations.
  2. At least one rule section is required when the Remote Key Export callable service is used to:
     - Generate an RKX key-token
     - Export an RKX key-token
     - Export a CCA DES key-token
     - Encrypt the clear generated or exported key using the provided vendor certificate
  3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.
- An optional name (key label) section (trusted block section identifier X'13')

  The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block. No multiple sections are allowed.
- A required information section (trusted block section identifier X'14')

  The trusted block information section contains control and security information related to the trusted block. The information section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system. No multiple sections are allowed.
- An optional application-defined data section (trusted block section identifier X'15')

  The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application. CCA does not examine or use this data in any way. No multiple sections are allowed.

Trusted block integrity

An enciphered confounder and triple-length MAC key contained within the required information section of the trusted block is used to protect the integrity of the trusted block. The randomly generated MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. Together, the MAC key and MAC value provide a way to verify that the trusted block originated from an authorized source, and binds it to the local system.

An external trusted block has its MAC key enciphered under an IMP-PKA key-encrypting key. An internal trusted block has its MAC key enciphered under a variant of the PKA master key, and the master key verification pattern is stored in the information section.

Number representation in trusted blocks

- All length fields are in binary.
- All binary fields (exponents, lengths, and so forth) are stored with the high-order byte first; thus the least significant bits are to the right and preceded with zero-bits to the width of a field.
- In variable-length binary fields that have an associated field-length value, leading bytes that would otherwise contain X'00' can be dropped and the field can be shortened to contain only the significant bits.

Format of trusted block sections

At the beginning of every trusted block is a trusted block header. The header contains the following information:

- A token identifier, which specifies if the token contains an external or internal key-token
- A token version number to allow for future changes
- A length in bytes of the trusted block, including the length of the header

The trusted block header is defined in the following table:

Table 654. Trusted block header

Offset (bytes)	Length (bytes)	Description
000	001	Token identifier (a flag that indicates token type) **X'1E'** External trusted block token **X'1F'** Internal trusted block token
001	001	Token version number (X'00').
002	002	Length of the key-token structure in bytes.
004	004	Reserved, binary zero.

Note: See “Number representation in trusted blocks” on page 1590.

Following the header, in no particular order, are trusted block sections. There are five different sections defined, each identified by a one-byte section identifier (X'11' - X'15'). Two of the five sections have subsections defined. A subsection is a tag-length-value (TLV) object, identified by a two-byte subsection tag.

Only sections X'12' and X'14' have subsections defined; the other sections do not. A section and its subsections, if any, are one contiguous unit of data. The subsections are concatenated to the related section, but are otherwise in no particular order. Section X'12' has five subsections defined (X'0001' - X'0005'), and section X'14' has two (X'0001' and X'0002'). Of all the subsections, only subsection X'0001' of section X'14' is required. Section X'14' is also required.

The trusted block sections and subsections are described in detail in the following sections.

Note: See “Number representation in trusted blocks” on page 1590.

Trusted block section X'11'

Trusted block section X'11' contains the trusted RSA public key in addition to a key-usage flag indicating whether the public key is usable in key-management operations, digital signature operations, or both.

Section X'11' is optional. No multiple sections are allowed. It has no subsections defined.

This section is defined in the following table:

Table 655. Trusted block trusted RSA public-key section (X'11')

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'11'** Trusted block trusted RSA public key
001	001	Section version number (X'00').
002	002	Section length (16+*xxx*+*yyy*).
004	002	Reserved, must be binary zero.
006	002	RSA public-key exponent field length in bytes, *xxx*.
008	002	RSA public-key modulus length in bits.
010	002	RSA public-key modulus field length in bytes, *yyy*.
012	xxx	Public-key exponent, *e* (this field length is typically 1, 3, or 64 - 512 bytes). *e* must be odd and $1 \le e < n$. (*e* is frequently valued to 3 or $2^{16}+1$ (=65537), otherwise *e* is of the same order of magnitude as the modulus). **Note:** Although the current product implementation does not generate such a public key, you can import an RSA public key having an exponent valued to two (2). Such a public key (a Rabin key) can correctly validate an ISO 9796-1 digital signature.
012+*xxx*	*yyy*	RSA public-key modulus, *n*. $n=pq$, where *p* and *q* are prime and $2^{512} \le n < 2^{4096}$. The field length is 64 - 512 bytes.
012 +*xxx*+*yyy*	004	Flags: **X'00000000'** Trusted block public key can be used in digital signature operations only **X'80000000'** Trusted block public key can be used in both digital signature and key management operations **X'C0000000'** Trusted block public key can be used in key management operations only

Note: See “Number representation in trusted blocks” on page 1590.

Trusted block section X'12'

Trusted block section X'12' contains information that defines a rule. A trusted block may have zero or more rule sections.

1. A trusted block with no rule sections can be used by the PKA Key Token Change and PKA Key Import callable services. A trusted block with no rule sections can be used by the Digital Signature Verify callable service, provided there is an RSA public-key section that has its key-usage flag set to allow digital signature operations.
2. At least one rule section is required when the Remote Key Export callable service is used to:

- Generate an RKX key-token
- Export an RKX key-token
- Export a CCA DES key-token
- Generate or export a key encrypted by a public key. The public key is contained in a vendor certificate (section X'11'), and is the root certification key for the ATM vendor. It is used to verify the digital signature on public-key certificates for specific individual ATMs.

3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.

Section X'12' is the only section that is allowed to have multiple sections. Section X'12' is optional. Multiple sections are allowed.

Note: The overall length of the trusted block may not exceed its maximum size of 3500 bytes.

Five subsections (TLV objects) are defined.

This section is defined in the following table:

Table 656. Trusted block rule section (X'12')

Offset (bytes)	**Length (bytes)**	**Description**
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'12'** Trusted block rule
001	001	Section version number (X'00').
002	002	Section length in bytes (20+*yyy*).
004	008	Rule ID (in ASCII). An 8-byte character string that uniquely identifies the rule within the trusted block. Valid ASCII characters are: A...Z, a...z, 0...9, - (hyphen), and _ (underscore), left justified and padded on the right with space characters.
012	004	Flags (undefined flag bits are reserved and must be zero). **X'00000000'** Generate new key. **X'00000001'** Export existing key.
016	001	Generated key length. Length in bytes of key to be generated when flags value (offset 012) is set to generate a new key; otherwise, ignore this value. Valid values are 8, 16, or 24; return an error if not valid.
017	001	Key-check algorithm identifier (all others are reserved and must not be used): **Value** **Meaning** **X'00'** Do not compute key-check value. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to zero. **X'01'** Encrypt an 8-byte block of binary zeros with the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 8. **X'02'** Compute the MDC-2 hash of the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 16.

Table 656. Trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
018	001	Symmetric encrypted output key format flag (all other values are reserved and must not be used). Return the indicated symmetric key-token using the *sym_encrypted_key_identifier* parameter. **Value** **Meaning** **X'00'** Return an RKX key-token encrypted under a variant of the MAC key. **Note:** This key format is permitted when the flags value (offset 012) is set to either: • Generate a new key. • Export an existing key. **X'01'** Return a CCA fixed-length DES key-token encrypted under a transport key. **Note:** This key format is not permitted if the flags value (offset 012) is set to generate a new key. It is only permitted when exporting an existing key.
019	001	Asymmetric encrypted output key format flag (all other values are reserved and must not be used). Return the indicated asymmetric key-token in the *asym_encrypted_key* variable. **Value** **Meaning** **X'00'** Do not return an asymmetric key. Set the *asym_encrypted_key_length* variable to zero. **X'01'** Output in PKCS1.2 format. **X'02'** Output in RSAOAEP format by using SHA-1. **X'04'** Output in RSAOAEP format by using SHA-256.
020	*yyy*	Rule section subsections (tag-length-value objects). A series of 0 - 5 objects in TLV format.

Note: See "Number representation in trusted blocks" on page 1590.

Section X'12' has five rule subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 657. Summary of trusted block rule subsection

Rule subsection tag	TLV object	Optional or required	Comments
X'0001'	Transport key variant	Optional	Contains variant to be exclusive-ORed into the cleartext transport key.
X'0002'	Transport key rule reference	Optional; required to use an RKX key-token as a transport key.	Contains the rule ID for the rule that must have been used to create the transport key.
X'0003'	Common export key parameters	Optional for key generation; required for key export of an existing key	Contains the export key and source key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key to control usage of the key.
X'0004'	Source key reference	Optional; required if the source key is an RKX key-token	Contains the rule ID for the rule that is used to create the source key. **Note:** Include all rules that will ever be needed when a trusted block is created. A rule cannot be added to a trusted block after it has been created.

Table 657. Summary of trusted block rule subsection (continued)			
Rule subsection tag	**TLV object**	**Optional or required**	**Comments**
X'0005'	Export key CCA token parameters	Optional; used for export of CCA DES key tokens only	Contains mask length, mask, and CV template to limit the usage of the exported key. Also contains the template length and template, which defines which source key labels are allowed. The key type of a source key input parameter can be "filtered" by using the export key CV limit mask (offset 005) and limit template (offset 005+*yyy*) in this subsection.

Note: See "Number representation in trusted blocks" on page 1590.

Trusted block section X'12' subsection X'0001': Subsection X'0001' of the trusted block rule section (X'12') is the transport key variant TLV object. This subsection is optional. It contains a variant to be exclusive-ORed into the cleartext transport key.

This subsection is defined in the following table:

Table 658. Transport key variant subsection (X'0001') of trusted block rule section (X'12')		
Offset (bytes)	**Length (bytes)**	**Description**
000	002	Subsection tag: **X'0001'** Transport key variant TLV object
002	002	Subsection length in bytes (8+*nnn*).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Length of variant field in bytes (*nnn*). This length must be greater than or equal to the length of the transport key that is identified by the *transport_key_identifier* parameter. If the variant is longer than the key, truncate it on the right to the length of the key before use.
008	*nnn*	Transport key variant. Exclusive-OR this variant into the cleartext transport key, provided: (1) the length of the variant field value (offset 007) is not zero, and (2) the symmetric encrypted output key format flag (offset 018 in section X'12') is X'01'. **Note:** A transport key is not used when the symmetric encrypted output key is in RKX key-token format.

Note: See "Number representation in trusted blocks" on page 1590.

Trusted block section X'12' subsection X'0002': Subsection X'0002' of the trusted block rule section (X'12') is the transport key rule reference TLV object. This subsection is optional. It contains the rule ID for the rule that must have been used to create the transport key. This subsection must be present to use an RKX key-token as a transport key.

This subsection is defined in the following table:

Table 659. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12')		
Offset (bytes)	**Length (bytes)**	**Description**
000	002	Subsection tag: **X'0002'** Transport key rule reference TLV object.
002	002	Subsection length in bytes (14).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.

Table 659. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
006	008	Rule ID. Contains the rule identifier for the rule that must have been used to create the RKX key-token used as the transport key. The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.

Trusted block section (X'12') subsection X'0003': Subsection X'0003' of the trusted block rule section (X'12') is the common export key parameters TLV object. This subsection is optional, but is required for the key export of an existing source key (identified by the *source_key_identifier* parameter) in either RKX key-token format or CCA DES key-token format. For new key generation, this subsection applies the output key variant to the cleartext generated key, if such an option is wanted. It contains the input source key and output export key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key.

This subsection is defined in the following table:

Table 660. Common export key parameters subsection (X'0003') of trusted block rule section (X'12')		
Offset (bytes)	**Length (bytes)**	**Description**
000	002	Subsection tag: **X'0003'** Common export key parameters TLV object.
002	002	Subsection length in bytes (12+*xxx*+*yyy*).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Flags (must be set to binary zero).
008	001	Export key minimum length in bytes. Length must be 8, 16, or 24. Also applies to the source key.
009	001	Export key maximum length in bytes (*yyy*). Length must be 8, 16, or 24. Also applies to the source key. The export key maximum length must be less than or equal to the CV length with one exception: When the export key maximum length is 24, then the CV length must be equal to 16. The CV must be all binary zeroes or zeroes with the keypart bit and parity turned on.
010	001	Output key variant length in bytes (*xxx*). Valid values are 0 or 8 - 255. If greater than 0, the length must be at least as long as the longest key ever to be exported using this rule. If the variant is longer than the key, truncate it on the right to the length of the key before use. **Note:** The output key variant (offset 011) is not used if this length is zero.
011	*xxx*	Output key variant. The variant can be any value. Exclusive-OR this variant into the cleartext value of the output.

Table 660. Common export key parameters subsection (X'0003') of trusted block rule section (X'12') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
011+*xxx*	001	CV length in bytes (*yyy*). • If the length is not 0, 8, or 16, return an error. • If the length is 0, and if the source key is a CCA DES key-token, preserve the CV in the symmetric encrypted output if the output is to be in the form of a CCA DES key-token. • If a non-zero length is less than the length of the key that is identified by the *source_key_identifier* parameter, return an error. • If the length is 16, and if the CV (offset 012+*xxx*) is valued to 16 bytes of X'00' (ignoring the key-part bit), then: 1. Ignore all CV bit definitions 2. If CCA DES key-token format, set the flag byte of the symmetric encrypted output key to indicate that a CV value is present. 3. If the source key is 8 bytes in length, do not replicate the key to 16 bytes.
012+*xxx*	*yyy*	CV. Place this CV into the output exported key-token if the symmetric encrypted output key format selected (offset 018 in rule section) is CCA DES key-token. • If the symmetric encrypted output key format flag (offset 018 in section X'12') indicates return an RKX key-token (X'00'), then ignore this CV. Otherwise, exclusive-OR this CV into the cleartext transport key. • Exclusive-OR the CV of the source key into the cleartext transport key if the CV length (offset 011+*xxx*) is set to 0. If a transport key to encrypt a source key has equal left and right key halves, return an error. Replicate the key halves of the key that is identified by the *source_key_identifier* parameter whenever all of these conditions are met: 1. The Replicate Key command (offset X'00DB') is enabled in the active role. 2. The CV length (offset 011+*xxx*) is 16, and both CV halves are non-zero. 3. The *source_key_identifier* parameter (contained in either a CCA DES key-token or RKX key-token) identifies an 8-byte key. 4. The key-form bits (40 - 42) of this CV do not indicate a single-length key (are not set to zero). 5. Key-form bit 40 of this CV does not indicate that the key is to have guaranteed unique halves (is not set to 1). **Note:** A transport key is not used when the symmetric encrypted output key is in RKX key-token format.

Note: See "Number representation in trusted blocks" on page 1590.

Trusted block section X'12' subsection X'0004': Subsection X'0004' of the trusted block rule section (X'12') is the source key rule reference TLV object. This subsection is optional, but is required if using an RKX key-token as a source key (identified by *source_key_identifier* parameter). It contains the rule ID for the rule that is used to create the export key. If this subsection is not present, an RKX key-token format source key will not be accepted for use.

This subsection is defined in the following table:

Table 661. Source key rule reference subsection (X'0004' of trusted block rule section (X'12')		
Offset (bytes)	**Length (bytes)**	**Description**
000	002	Subsection tag: **X'0004'** Source key rule reference TLV object.
002	002	Subsection length in bytes (14).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.

Table 661. Source key rule reference subsection (X'0004' of trusted block rule section (X'12') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
006	008	Rule ID. Rule identifier for the rule that must have been used to create the source key. The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.

Note: See "Number representation in trusted blocks" on page 1590.

Trusted block section X'12' subsection X'0005': Subsection X'0005' of the trusted block rule section (X'12') is the export key CCA token parameters TLV object. This subsection is optional. It contains a mask length, mask, and template for the export key CV limit. It also contains the template length and template for the source key label. When using a CCA DES key-token as a source key input parameter, its key type can be "filtered" by using the export key CV limit mask (offset 005) and limit template (offset 005+*yyy*) in this subsection.

This subsection is defined in the following table:

Table 662. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12')		
Offset (bytes)	**Length (bytes)**	**Description**
000	002	Subsection tag: **X'0005'** Export key CCA token parameters TLV object.
002	002	Subsection length in bytes (10+*yyy*+*yyy*+*zzz*).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Flags (must be set to binary zero).
008	001	Export key CV limit mask length in bytes (*yyy*). Do not use CV limits if this CV limit mask length (*yyy*) is zero. Use CV limits if *yyy* is non-zero, in which case *yyy*: • Must be 8 or 16. • Must not be less than the export key minimum length (offset 008 in subsection X'0003'). • Must be equal in length to the actual source key length of the key. **Example**: An export key minimum length of 16 and an export key CV limit mask length of 8 returns an error.
009	*yyy*	Export key CV limit mask (does not exist if *yyy*=0). Indicates which CV bits to check against the source key CV limit template (offset 009+*yyy*). **Examples**: A mask of X'FF' means check all bits in a byte. A mask of X'FE' ignores the parity bit in a byte.

Table 662. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12') (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
009+*yyy*	*yyy*	Export key CV limit template (does not exist if *yyy*=0). Specifies the required values for those CV bits that are checked based on the export key CV limit mask (offset 009). The export key CV limit mask and template have the same length, *yyy*. This is because these two variables work together to restrict the acceptable CVs for CCA DES key tokens to be exported. The checks work as follows: 1. If the length of the key to be exported is less than *yyy*, return an error. 2. Logical AND the CV for the key to be exported with the export key CV limit mask. 3. Compare the result to the export key CV limit template. 4. Return an error if the comparison is not equal. **Examples**: An export key CV limit mask of X'FF' for CV byte 1 (key type) along with an export key CV limit template of X'3F' (key type CVARENC) for byte 1 filters out all key types except CVARENC keys. **Note:** Using the mask and template to permit multiple key types is possible, but cannot consistently be achieved with one rule section. For example, setting bit 10 to 1 in the mask and the template permits PIN processing keys and cryptographic variable encrypting keys, and only those keys. However, a mask to permit PIN-processing keys and key-encrypting keys, and only those keys, is not possible. In this case, multiple rule sections are required, one to permit PIN-processing keys and the other to permit key-encrypting keys.
009+ ***yyy+ yyy***	001	Source key label template length in bytes (*zzz*). Valid values are 0 and 64. Return an error if the length is 64 and a source key label is not provided.
010+ ***yyy+ yyy***	*zzz*	Source key label template (does not exist if *zzz*=0). If a key label is identified by the *source_key_identifier* parameter, verify that the key label name matches this template. If the comparison fails, return an error. The source key label template must conform to the following rules: • The key label template must be 64 bytes in length. • The first character cannot be in the range X'00' - X'1F', nor can it be X'FF'. • The first character cannot be numeric (X'30' - X'39'). • A key label name is terminated by a space character (X'20') on the right and must be padded on the right with space characters. • The only special characters that are permitted are #, $, @, and * (X'23', X'24', X'40', and X'2A'). • The wildcard X'2A' (*) is only permitted as the first character, the last character, or the only character in the template. • Only alphanumeric characters (a...z, A...Z, 0...9), the four special characters (X'23', X'24', X'40', and X'2A'), and the space character (X'20') are allowed.

Note: See “Number representation in trusted blocks” on page 1590.

Trusted block section X'13'

Trusted block section X'13' contains the name (key label). The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block.

Section X'13' is optional. No multiple sections are allowed. It has no subsections defined. This section is defined in the following table:

Table 663. Trusted block key label (name) section X'13'		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'13'** Trusted block name (key label)
001	001	Section version number (X'00').
002	002	Section length in bytes (68).
004	064	Name (key label).

Note: See "Number representation in trusted blocks" on page 1590.

Trusted block section X'14'

Trusted block section X'14' contains control and security information related to the trusted block. This information section is separate from the public key and other sections because this section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system.

Section X'14' is required. No multiple sections are allowed. Two subsections are defined. This section is defined in the following table:

Table 664. Trusted block information section X'14'		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'14'** Trusted block information
001	001	Section version number (X'00').
002	002	Section length in bytes (10+*xxx*).
004	002	Reserved, binary zero.
006	004	Flags: **X'00000000'** Trusted block is in the inactive state **X'00000001'** Trusted block is in the active state
010	*xxx*	Information section subsections (tag-length-value objects). One or two objects in TLV format.

Note: See "Number representation in trusted blocks" on page 1590.

Section X'14' has two information subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 665. Summary of trusted block information subsections			
Rule subsection tag	**TLV object**	**Optional or required**	**Comments**
X'0001'	Protection information	Required	Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO 16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).

Table 665. Summary of trusted block information subsections (continued)			
Rule subsection tag	**TLV object**	**Optional or required**	**Comments**
X'0002'	Activation and expiration dates	Optional	Contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.

Note: See “Number representation in trusted blocks” on page 1590.

Trusted block section X'14' subsection X'0001': Subsection X'0001' of the trusted block information section (X'14') is the protection information TLV object. This subsection is required. It contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO-16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).

This subsection is defined in the following table:

Table 666. Protection information subsection (X'0001') of trusted block information section (X'14')		
Offset (bytes)	**Length (bytes)**	**Description**
000	002	Subsection tag: **X'0001'** Trusted block information TLV object
002	002	Subsection length in bytes (62).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	032	Encrypted MAC key. Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key in the following format: **Offset** **Description** **00 - 07** Confounder **08 - 15** Left key **16 - 23** Middle key **24 - 31** Right key
038	008	MAC. Contains the ISO-16609 TDES CBC message authentication code value.
046	016	MKVP. Contains the PKA master key verification pattern, computed using MDC4, when the trusted block is in internal form, otherwise contains binary zero.

Note: See “Number representation in trusted blocks” on page 1590.

Trusted block section X'14' subsection X'0002': Subsection X'0002' of the trusted block information section (X'14') is the activation and expiration dates TLV object. This subsection is optional. It contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.

This subsection is defined in the following table:

Table 667. Activation/expiration dates subsection (X'0002') of trusted block information section (X'14')		
Offset (bytes)	**Length (bytes)**	**Description**
000	002	Subsection tag: **X'0002'** Activation and expiration dates TLV object
002	002	Subsection length in bytes (16).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	002	Flags: **X'0000'** The coprocessor does not check dates. **X'0001'** The coprocessor checks dates. Compare the activation date (offset 008) and the expiration date (offset 012) to the coprocessor's internal real-time clock. Return an error if the coprocessor date is before the activation date or after the expiration date.
008	004	Activation date. Contains the first date that the trusted block can be used for generating or exporting keys. Format of the date is YYMD, where: **YY** Big-endian year (return an error if greater than 9999) **M** Month (return an error if any value other than X'01' - X'0C') **D** Day of month (return an error if any value other than X'01' - X'1F'; Day must be valid for given month and year, including leap years.) Return an error if the activation date is after the expiration date or is not valid.
012	004	Expiration date. Contains the last date that the trusted block can be used. Same format as activation date (offset 008). Return an error if date is not valid.

Note: See “Number representation in trusted blocks” on page 1590.

Trusted block section X'15'

Trusted block section X'15' contains application-defined data. The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application; it is neither examined nor used by CCA in any way.

Section X'15' is optional. No multiple sections are allowed. It has no subsections defined. This section is defined in the following table:

Table 668. Trusted block application-defined data section X'15'		
Offset (bytes)	**Length (bytes)**	**Description**
000	001	Section identifier: **X'15'** Application-defined data
001	001	Section version number (X'00').
002	002	Section length (6+*xxx*)

Table 668. Trusted block application-defined data section X'15' (continued)		
Offset (bytes)	**Length (bytes)**	**Description**
004	002	Application data length (*xxx*) The value of *xxx* can be from 0 bytes to a length that does not cause the trusted block to exceed its maximum size of 3500 bytes.
006	*xxx*	Application-defined data May be used to hold a public-key certificate for the trusted public key.

Appendix C. Changing control vectors with the CVT callable service

This topic contains a control vector table which displays the default value of the control vector that is associated with each type of key. It also describes how to change control vectors with the control vector translate callable service.

DES control vector table

Note: The control vectors used in ICSF are the same as the ones documented in the CCA and the IBM 4765 PCIe and IBM 4767 PCIe Cryptographic Coprocessor documents.

The master key enciphers all DES keys operational on your system. A transport key enciphers keys that are distributed off your system. Before a master key or transport key enciphers a key, ICSF exclusive ORs both halves of the master key or transport key with a control vector. The same control vector is exclusive ORed to the left and right half of a master key or transport key.

Also, if you are entering a key part, ICSF exclusive ORs each half of the key part with a control vector before placing the key part into the CKDS.

Each type of key on ICSF (except the master key) has either one or two unique control vectors associated with it. The control vector that ICSF exclusive ORs the master key or transport key with depends on the type of key the master key or transport key is enciphering. For double-length keys, a unique control vector exists for each half of a specific key type. For example, there is a control vector for the left half of an input PIN-encrypting key, and a control vector for the right half of an input PIN-encrypting key.

If you are entering a key part into the CKDS, ICSF exclusive ORs the key part with the unique control vector or vectors associated with the key type. ICSF also enciphers the key part with two master key variants for a key part. One master key variant enciphers the left half of the key part, and another master key variant enciphers the right half of the key part. ICSF creates the master key variants for a key part by exclusive ORing the master key with the control vectors for key parts. These procedures protect key separation.

Table 669 on page 1603 displays the default value of the control vector that is associated with each type of key. Some key types do not have a default control vector. For keys that are double-length, ICSF enciphers a unique control vector on each half.

Table 669. Default control vector values

Key type	Control Vector Value (Hexadecimal value for left half of double-length key)	Control Vector Value (Hexadecimal value for right half of double-length key)
CIPHER (single length)	00 03 71 00 03 00 00 00	
CIPHER (double length)	00 03 71 00 03 41 00 00	00 03 71 00 03 21 00 00
CIPHER (triple length)	00 03 71 00 03 60 00 81	00 03 71 00 03 60 00 81
CIPHERXI	00 0C 50 00 03 C0 00 00	00 0C 50 00 03 A0 00 00
CIPHERXO	00 0C 60 00 03 C0 00 00	00 0C 60 00 03 A0 00 00
CIPHERXL	00 0C 71 00 03 C0 00 00	00 0C 71 00 03 A0 00 00
CVARDEC	00 3F 42 00 03 00 00 00	
CVARENC	00 3F 48 00 03 00 00 00	
CVARPINE	00 3F 41 00 03 00 00 00	

Table 669. Default control vector values (continued)

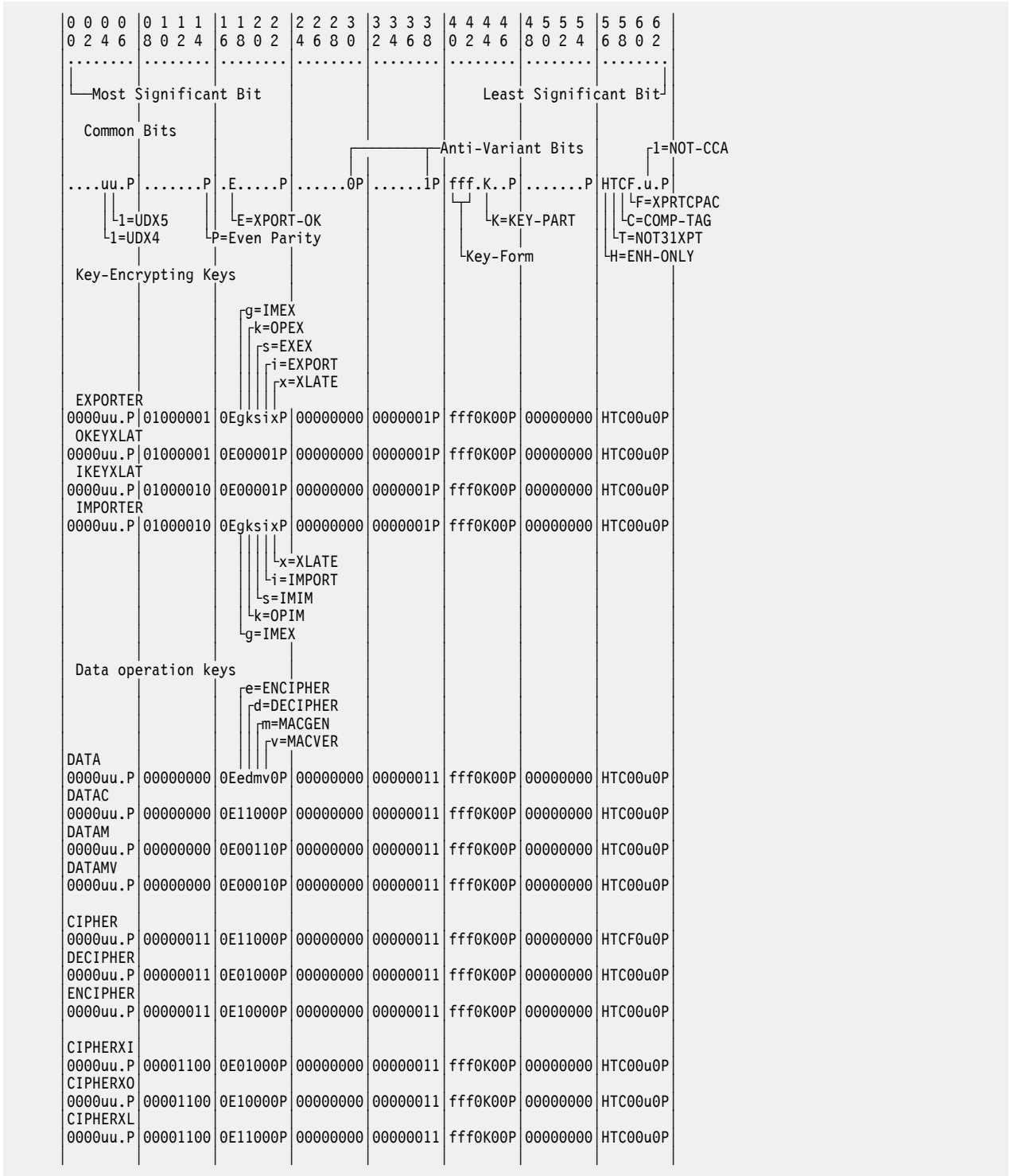
Key type	Control Vector Value (Hexadecimal value for left half of double-length key)	Control Vector Value (Hexadecimal value for right half of double-length key)
CVARXCVL	00 3F 44 00 03 00 00 00	
CVARXCVR	00 3F 47 00 03 00 00 00	
DATA	00 00 7D 00 03 00 00 00	
DATA zero CV	00 00 00 00 00 00 00 00	
DATA (double length)	00 00 7D 00 03 41 00 00	00 00 7D 00 03 21 00 00
DATA (double length) zero CV	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
DATA (triple-length)	00 00 7D 00 03 60 00 81	00 00 7D 00 03 60 00 81
DATA (triple-length) zero CV	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
DATAC	00 00 71 00 03 41 00 00	00 00 71 00 03 21 00 00
DATAM (external and internal)	00 00 4D 00 03 41 00 00	00 00 4D 00 03 21 00 00
DATAM (internal) DEPRECATED	00 00 4D 00 03 00 00 00	00 00 4D 00 03 00 00 00
DATAMV (external and internal)	00 00 44 00 03 41 00 00	00 00 44 00 03 21 00 00
DATAMV (internal) DEPRECATED	00 00 44 00 03 00 00 00	00 00 44 00 03 00 00 00
DECIPHER (single length)	00 03 50 00 03 00 00 00	
DECIPHER (double-length)	00 03 50 00 03 41 00 00	00 03 50 00 03 21 00 00
DECIPHER (triple-length)	00 03 50 00 03 60 00 81	00 03 50 00 03 60 00 81
DKYGENKY	00 71 44 00 03 41 00 00	00 71 44 00 03 21 00 00
ENCIPHER (single length)	00 03 60 00 03 00 00 00	
ENCIPHER (double-length)	00 03 60 00 03 41 00 00	00 03 60 00 03 21 00 00
ENCIPHER (triple-length)	00 03 60 00 03 60 00 81	00 03 60 00 03 60 00 81
EXPORTER (double-length)	00 41 7D 00 03 41 00 00	00 41 7D 00 03 21 00 00
EXPORTER (triple-length)	00 41 7D 00 03 60 00 81	00 41 7D 00 03 60 00 81
IKEYXLAT	00 42 42 00 03 41 00 00	00 42 42 00 03 21 00 00
IMP-PKA (double-length)	00 42 05 00 03 41 00 00	00 42 05 00 03 21 00 00
IMP-PKA (triple-length)	00 42 05 00 03 60 00 81	00 42 05 00 03 60 00 81
IMPORTER (double-length)	00 42 7D 00 03 41 00 00	00 42 7D 00 03 21 00 00
IMPORTER (triple-length)	00 42 7D 00 03 60 00 81	00 42 7D 00 03 60 00 81
IPINENC (double-length)	00 21 5F 00 03 41 00 00	00 21 5F 00 03 21 00 00
IPINENC (triple-length)	00 21 5F 00 03 60 00 00	00 21 5F 00 03 60 00 00
MAC (single length)	00 05 4D 00 03 00 00 00	
MAC (double-length)	00 05 4D 00 03 41 00 00	00 05 4D 00 03 21 00 00
MAC (triple-length)	00 05 4D 00 03 60 00 81	00 05 4D 00 03 60 00 81
MACVER (single length)	00 05 44 00 03 00 00 00	

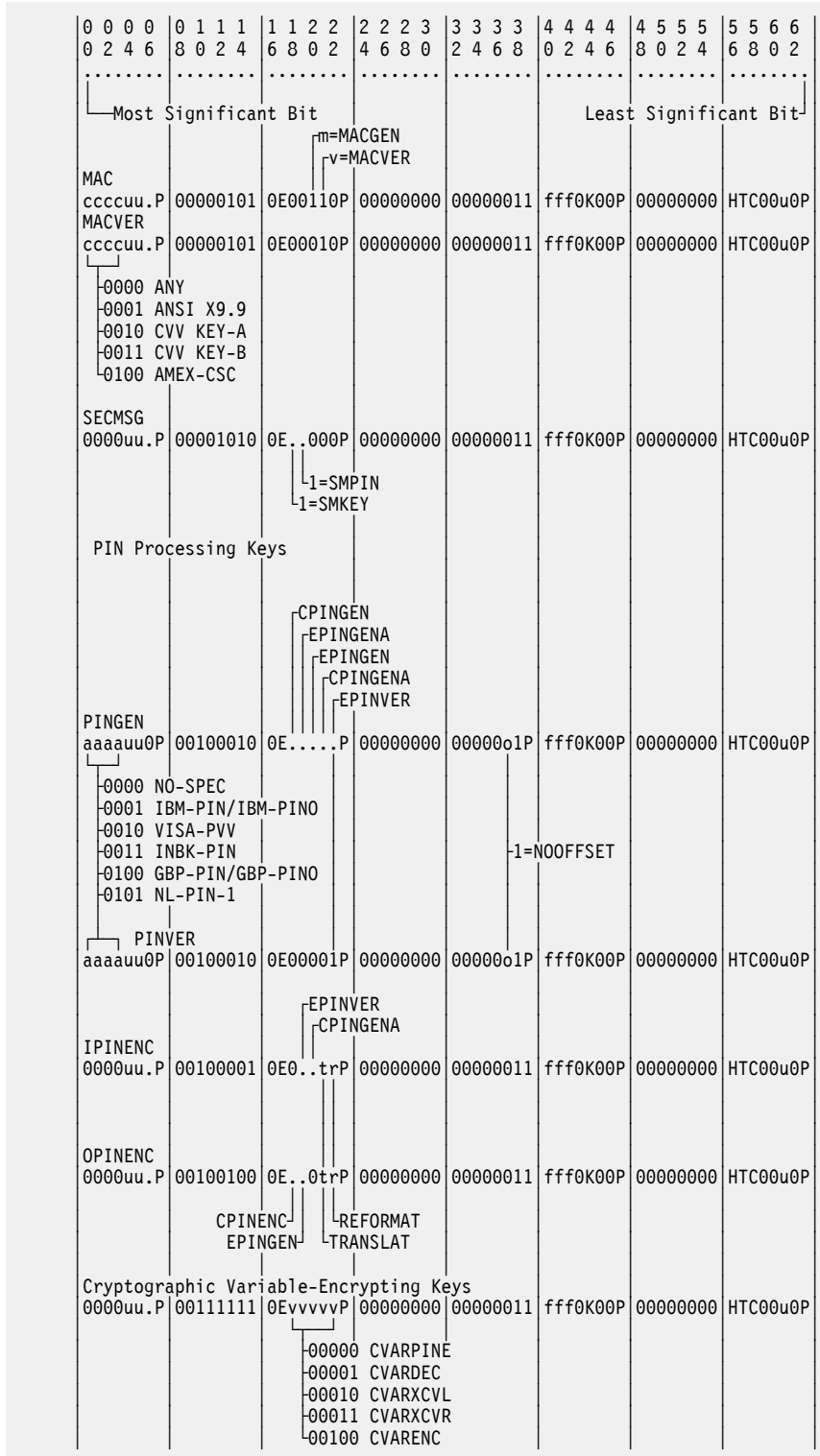
Table 669. Default control vector values (continued)		
Key type	**Control Vector Value (Hexadecimal value for left half of double-length key)**	**Control Vector Value (Hexadecimal value for right half of double-length key)**
MACVER (double-length)	00 05 44 00 03 41 00 00	00 05 44 00 03 21 00 00
MACVER (triple-length)	00 05 44 00 03 60 00 81	00 05 44 00 03 60 00 81
OKEYXLAT	00 41 42 00 03 41 00 00	00 41 42 00 03 21 00 00
OPINENC (double-length)	00 24 77 00 03 41 00 00	00 24 77 00 03 21 00 00
OPINENC (triple-length)	00 24 77 00 03 60 00 81	00 24 77 00 03 60 00 81
PINGEN (double-length)	00 22 7E 00 03 41 00 00	00 22 7E 00 03 21 00 00
PINGEN (triple-length)	00 22 7E 00 03 60 00 81	00 22 7E 00 03 60 00 81
PINVER (double-length)	00 22 42 00 03 41 00 00	00 22 42 00 03 21 00 00
PINVER (triple-length)	00 22 42 00 03 60 00 81	00 22 42 00 03 60 00 81

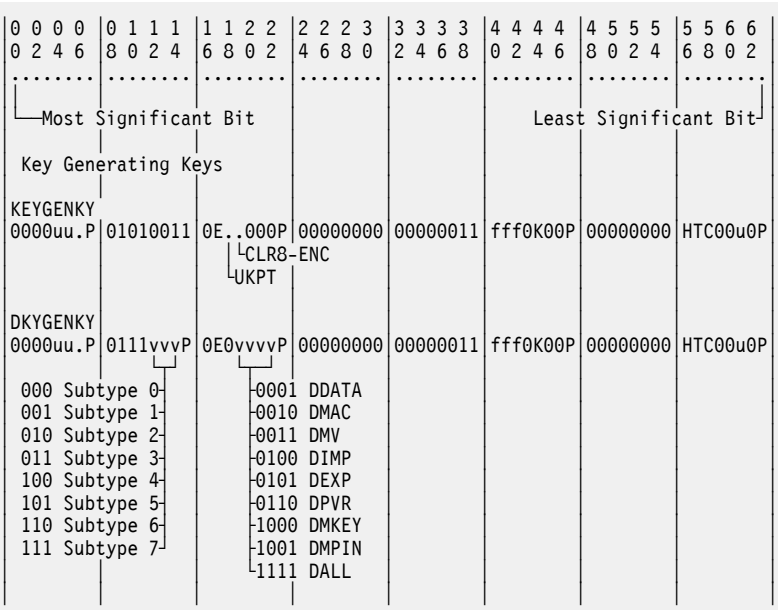
Note: The external control vectors for DATAC, DATAM, and DATAMV keys are also referred to as data compatibility control vectors.

Control-Vector Base Bits

Control Vector Base Bit Map (Common Bits and Key-Encrypting Keys):


0 0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 6 6
0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0 2
Most Significant Bit
Least Significant Bit
Common Bits
Anti-Variant Bits
1=NOT-CCA
....uu.P ....P .E.....P ......0P ......1P fff.K..P ....P HTCF.u.P
1=UDX5
1=UDX4
E=XPORT-OK
P=Even Parity
K=KEY-PART
Key-Form
F=XPRTCPAC
C=COMP-TAG
T=NOT31XPT
H=ENH-ONLY
Key-Encrypting Keys
g=IMEX
k=OPEX
s=EXEX
i=EXPORT
x=XLATE
EXPORTER
0000uu.P 01000001 0EgksixP 00000000 0000001P fff0K00P 00000000 HTC00u0P
OKEYXLAT
0000uu.P 01000001 0E00001P 00000000 0000001P fff0K00P 00000000 HTC00u0P
IKEYXLAT
0000uu.P 01000010 0E00001P 00000000 0000001P fff0K00P 00000000 HTC00u0P
IMPORTER
0000uu.P 01000010 0EgksixP 00000000 0000001P fff0K00P 00000000 HTC00u0P
x=XLATE
i=IMPORT
s=IMIM
k=OPIM
g=IMEX
Data operation keys
e=ENCIPHER
d=DECIPHER
m=MACGEN
v=MACVER
DATA
0000uu.P 00000000 0Eedmv0P 00000000 00000011 fff0K00P 00000000 HTC00u0P
DATAC
0000uu.P 00000000 0E11000P 00000000 00000011 fff0K00P 00000000 HTC00u0P
DATAM
0000uu.P 00000000 0E00110P 00000000 00000011 fff0K00P 00000000 HTC00u0P
DATAMV
0000uu.P 00000000 0E00010P 00000000 00000011 fff0K00P 00000000 HTC00u0P
CIPHER
0000uu.P 00000011 0E11000P 00000000 00000011 fff0K00P 00000000 HTCF0u0P
DECIPHER
0000uu.P 00000011 0E01000P 00000000 00000011 fff0K00P 00000000 HTC00u0P
ENCIPHER
0000uu.P 00000011 0E10000P 00000000 00000011 fff0K00P 00000000 HTC00u0P
CIPHERXI
0000uu.P 00001100 0E01000P 00000000 00000011 fff0K00P 00000000 HTC00u0P
CIPHERXO
0000uu.P 00001100 0E10000P 00000000 00000011 fff0K00P 00000000 HTC00u0P
CIPHERXL
0000uu.P 00001100 0E11000P 00000000 00000011 fff0K00P 00000000 HTC00u0P


0 0 0 0 0 2 4 6
0 1 1 1 8 0 2 4
1 1 2 2 6 8 0 2
2 2 2 3 4 6 8 0
3 3 3 3 2 4 6 8
4 4 4 4 0 2 4 6
4 5 5 5 8 0 2 4
5 5 6 6 6 8 0 2
Most Significant Bit
Least Significant Bit
m=MACGEN
v=MACVER
MAC
ccccuu.P 00000101 0E00110P 00000000 00000011 fff0K00P 00000000 HTC00u0P
MACVER
ccccuu.P 00000101 0E00010P 00000000 00000011 fff0K00P 00000000 HTC00u0P
0000 ANY
0001 ANSI X9.9
0010 CVV KEY-A
0011 CVV KEY-B
0100 AMEX-CSC
SECMSG
0000uu.P 00001010 0E..000P 00000000 00000011 fff0K00P 00000000 HTC00u0P
1=SMPIN
1=SMKEY
PIN Processing Keys
CPINGEN
EPINGENA
EPINGEN
CPINGENA
EPINVER
PINGEN
aaaauu0P 00100010 0E.....P 00000000 00000o1P fff0K00P 00000000 HTC00u0P
0000 NO-SPEC
0001 IBM-PIN/IBM-PINO
0010 VISA-PVV
0011 INBK-PIN
1=NOOFFSET
0100 GBP-PIN/GBP-PINO
0101 NL-PIN-1
PINVER
aaaauu0P 00100010 0E00001P 00000000 00000o1P fff0K00P 00000000 HTC00u0P
EPINVER
CPINGENA
IPINENC
0000uu.P 00100001 0E0..trP 00000000 00000011 fff0K00P 00000000 HTC00u0P
OPINENC
0000uu.P 00100100 0E..0trP 00000000 00000011 fff0K00P 00000000 HTC00u0P
CPINENC
EPINGEN
REFORMAT
TRANSLAT
Cryptographic Variable-Encrypting Keys
0000uu.P 00111111 0EvvvvvP 00000000 00000011 fff0K00P 00000000 HTC00u0P
00000 CVARPINE
00001 CVARDEC
00010 CVARXCVL
00011 CVARXCVR
00100 CVARENC



Key form bits, 'fff'

The key form bits, 40-42, and for a double-length key, bits 104-106, are designated 'fff' in the preceding illustration. These bits can have these values:

000
Single length key.

x10
Double length key, left half.

x01
Double length key, right half.

x11
Triple-length key.

1xx
Parts guaranteed unique ('110', '101', '111').

Specifying a control-vector-base value

You can determine the value of a control vector by working through the following series of questions:

1. Begin with a field of 64 bits (eight bytes) set to B'0'. The most significant bit is referred to as bit 0. Define the key type and subtype (bits 8 to 14), as follows:
   - The main key type bits (bits 8 to 11). Set bits 8 to 11 to one of the following values:

Table 670. Main key type for bits 8 to 11

Bits 8 to 11	Main Key Type
0000	Data operation keys
0010	PIN keys
0011	Cryptographic variable-encrypting keys
0100	Key-encrypting keys
0101	Key-generating keys
0111	Diversified key-generating keys

- The key subtype bits (bits 12 to 14). Set bits 12 to 14 to one of the following values:

  Note: For Diversified Key Generating Keys, the subtype field specifies the hierarchical level of the DKYGENKY. If the subtype is non-zero, then the DKYGENKY can only generate another DKYGENKY key with the hierarchy level decremented by one. If the subtype is zero, the DKYGENKY can only generate the final diversified key (a non-DKYGENKY key) with the key type specified by the usage bits.

Table 671. Key Subtype

Bits 12 to 14	Key Subtype
Data Operation Keys	
000	Compatibility key (DATA)
001	Confidentiality key (CIPHER, DECIPHER, or ENCIPHER)
010	MAC key (MAC or MACVER)
101	Secure messaging keys
110	Cipher text translate key (CIPHERXI, CIPHERXL, CIPHERXO)
Key-Encrypting Keys	
000	Transport-sending keys (EXPORTER and OKEYXLAT)
001	Transport-receiving keys (IMPORTER and IKEYXLAT)
PIN Keys	
001	PIN-generating key (PINGEN, PINVER)
000	Inbound PIN-block decrypting key (IPINENC)
010	Outbound PIN-block encrypting key (OPINENC)
Cryptographic Variable-Encrypting Keys	
111	Cryptographic variable-encrypting key (CVAR....)
Diversified Key Generating Keys	
000	DKY Subtype 0
001	DKY Subtype 1
010	DKY Subtype 2
011	DKY Subtype 3
100	DKY Subtype 4
101	DKY Subtype 5
110	DKY Subtype 6
111	DKY Subtype 7

2. For key-encrypting keys, set the following bits:

   - The key-generating usage bits (gks, bits 18 to 20). Set the gks bits to B'111' to indicate that the Key Generate callable service can use the associated key-encrypting key to encipher generated keys when the Key Generate callable service is generating various key-pair key-form combinations. See Key-Encrypting Keys in “Control-Vector Base Bits” on page 1605. Without any of the gks bits set to 1, the Key Generate callable service cannot use the associated key-encrypting key. The Key Token Build callable service can set the gks bits to 1 when you supply the OPIM, IMEX, IMIM, OPEX, and EXEX keywords.

- The IMPORT and EXPORT bit and the XLATE bit (ix, bits 21 and 22). If the 'i' bit is set to 1, the associated key-encrypting key can be used in the Data Key Import, Key Import, Data Key Export, and Key Export callable services. If the 'x' bit is set to 1, the associated key-encrypting key can be used in the Key Translate callable service.
- The key-form bits (fff, bits 40 to 42). The key-form bits indicate how the key was generated and how the control vector participates in multiple-enciphering. To indicate that the parts can be the same value, set these bits to B'010'. For information about the value of the key-form bits in the right half of a control vector, see Step 8.

3. For MAC and MACVER keys, set the following bits:
   - The MAC control bits (bits 20 and 21). For a MAC-generate key, set bits 20 and 21 to B'11'. For a MAC-verify key, set bits 20 and 21 to B'01'.
   - The key-form bits (fff, bits 40 to 42). For a single-length key, set the bits to B'000'. For a double-length key, set the bits to B'010'.
4. For PINGEN and PINVER keys, set the following bits:
   - The PIN calculation method bits (aaaa, bits 0 to 3). Set these bits to one of the following values:

Bits 0 to 3	Calculation Method Keyword	Description
0000	NO-SPEC	A key with this control vector can be used with any PIN calculation method.
0001	IBM-PIN or IBM-PINO	A key with this control vector can be used only with the IBM PIN or PIN Offset calculation method.
0010	VISA-PVV	A key with this control vector can be used only with the VISA-PVV calculation method.
0100	GBP-PIN or GBP-PINO	A key with this control vector can be used only with the German Banking Pool PIN or PIN Offset calculation method.
0011	INBK-PIN	A key with this control vector can be used only with the Interbank PIN calculation method.
0101	NL-PIN-1	A key with this control vector can be used only with the NL-PIN-1, Netherlands PIN calculation method.

   - The prohibit-offset bit (o, bit 37) to restrict operations to the PIN value. If set to 1, this bit prevents operation with the IBM 3624 PIN Offset calculation method and the IBM German Bank Pool PIN Offset calculation method.
5. For PINGEN, IPINENC, and OPINENC keys, set bits 18 to 22 to indicate whether the key can be used with the following callable services

Service Allowed	Bit Name	Bit
Clear PIN Generate	CPINGEN	18
Encrypted PIN Generate Alternate	EPINGENA	19
Encrypted PIN Generate	EPINGEN	20 for PINGEN 19 for OPINENC

Service Allowed	Bit Name	Bit
Clear PIN Generate Alternate	CPINGENA	21 for PINGEN 20 for IPINENC
Encrypted Pin Verify	EPINVER	19
Clear PIN Encrypt	CPINENC	18

6. For the IPINENC (inbound) and OPINENC (outbound) PIN-block ciphering keys, do the following:
   - Set the TRANSLAT bit (t, bit 21) to 1 to permit the key to be used in the PIN Translate callable service. The Control Vector Generate callable service can set the TRANSLAT bit to 1 when you supply the TRANSLAT keyword.
   - Set the REFORMAT bit (r, bit 22) to 1 to permit the key to be used in the PIN Translate callable service. The Control Vector Generate callable service can set the REFORMAT bit and the TRANSLAT bit to 1 when you supply the REFORMAT keyword.
7. For the cryptographic variable-encrypting keys (bits 18 to 22), set the variable-type bits (bits 18 to 22) to one of the following values:

Bits 18 to 22	Generic Key Type	Description
00000	CVARPINE	Used in the Encrypted PIN Generate Alternate service to encrypt a clear PIN.
00010	CVARXCVL	Used in the Control Vector Translate callable service to decrypt the left mask array.
00011	CVARXCVR	Used in the Control Vector Translate callable service to decrypt the right mask array.
00100	CVARENC	Used in the Cryptographic Variable Encipher callable service to encrypt an unformatted PIN.

8. For key-generating keys, set the following bits:
   - For KEYGENKY, set bit 18 for UKPT usage and bit 19 for CLR8-ENC usage.
   - For DKYGENKY, bits 12-14 will specify the hierarchical level of the DKYGENKY key. If the subtype CV bits are non-zero, then the DKYGENKY can only generate another DKYGENKY key with the hierarchical level decremented by one. If the subtype CV bits are zero, the DKYGENKY can only generate the final diversified key (a non-DKYGENKY key) with the key type specified by usage bits.

     To specify the subtype values of the DKYGENKY, keywords DKYL0, DKYL1, DKYL2, DKYL3, DKYL4, DKYL5, DKYL6 and DKYL7 will be used.
   - For DKYGENKY, bit 18 is reserved and must be zero.
   - Usage bits 19-22 for the DKYGENKY key type are defined as follows. They will be encoded as the final key type that the DKYGENKY key generates.

Bits 19 to 22	Keyword	Usage
0001	DDATA	DATA, DATAC, single or double length
0010	DMAC	MAC, DATAM
0011	DMV	MACVER, DATAMV
0100	DIMP	IMPORTER, IKEYXLAT

Bits 19 to 22	Keyword	Usage
0101	DEXP	EXPORTER, OKEYXLAT
0110	DPVR	PINVER
1000	DMKEY	Secure message key for encrypting keys
1001	DMPIN	Secure message key for encrypting PINs
1111	DALL	All key types may be generated except DKYGENKY and KEYGENKY keys. Usage of the DALL keyword is controlled by a separate access control point.

9. For secure messaging keys, set the following bits:
   - Set bit 18 to 1 if the key will be used in the secure messaging for PINs service. Set bit 19 to 1 if the key will be used in the secure messaging for keys service.
10. For CIPHER keys, set the CPACF exportable bit (XPRTCPAC, F - bit 59) to 1 to allow the key token to be exportable to the CPACF protected key format.
11. For all keys, set the following bits:
    - The export bit (E, bit 17). If set to 0, the export bit prevents a key from being exported. By setting this bit to 0, you can prevent the receiver of a key from exporting or translating the key for use in another cryptographic subsystem. Once this bit is set to 0, it cannot be set to 1 by any service other than Control Vector Translate. The Prohibit Export callable service can reset the export bit.
    - The key-part bit (K, bit 44). Set the key-part bit to 1 in a control vector associated with a key part. When the final key part is combined with previously accumulated key parts, the key-part bit in the control vector for the final key part is set to 0. The Control Vector Generate callable service can set the key-part bit to 1 when you supply the KEY-PART keyword.
    - The anti-variant bits (bit 30 and bit 38). Set bit 30 to 0 and bit 38 to 1. Many cryptographic systems have implemented a system of variants where a 7-bit value is exclusive-ORed with each 7-bit group of a key-encrypting key before enciphering the target key. By setting bits 30 and 38 to opposite values, control vectors do not produce patterns that can occur in variant-based systems.
    - Control vector bits 64 to 127. If bits 40 to 42 are B'000' (single-length key), set bits 64 to 127 to 0. If bits 41 and 42 are B'11', then copy bits 0 to 63 into bits 64 to 127. Otherwise, copy bits 0 to 63 into bits 64 to 127 and set bits 105 and 106 to B'01'.
    - Set the parity bits (low-order bit of each byte, bits 7, 15, ..., 127). These bits contain the parity bits (P) of the control vector. Set the parity bit of each byte so the number of zero-value bits in the byte is an even number.
    - For secure messaging keys, usage bit 18 on will enable the encryption of keys in a secure message and usage bit 19 on will enable the encryption of PINs in a secure message.
    - The ENH-ONLY bit (H, bit 56). Set the ENH-ONLY bit to 1 in a control vector to require the key value be encrypted with the enhanced wrapping. method. The Control Vector Generate callable service can set the ENH-ONLY bit to 1 when you supply the ENH-ONLY keyword.
    - The NOT31XPT bit (T, bit 57). Set T31XPOK bit to 1 to prevent the key from being exported by the TR-31 Translate service. Once this bit is set to 1, it cannot be set to 0 by any service. The Restrict Key Attribute service can set the bit to 1.
    - The compliance-tagged bit (COMP-TAG, C - bit 58). Set the COMP-TAG bit to 1 to prevent the token from being used in a non-compliant manner. Once this bit has been set to 1, it cannot be reset to 0. Key tokens may be created with the COMP-TAG bit set or the Key Translate2 (CSNBKTR2) service can be used to set the bit in an existing key token.

Changing control vectors with the Control Vector Translate callable service

Do the following when using the Control Vector Translate callable service:

- Provide the control information for testing the control vectors of the source, target, and key-encrypting keys to ensure that only sanctioned changes can be performed
- Select the key-half processing mode.

Providing the control information for testing the control vectors

To minimize your security exposure, the Control Vector Translate callable service requires control information (*mask array* information) to limit the range of allowable control vector changes. To ensure that this service is used only for authorized purposes, the source-key control vector, target-key control vector, and key-encrypting key (KEK) control vector must pass specific tests. The tests on the control vectors are performed within the secured cryptographic engine.

The tests consist of evaluating four logic expressions, the results of which must be a string of binary zeros. The expressions operate bitwise on information that is contained in the mask arrays and in the portions of the control vectors associated with the key or key-half that is being processed. If any of the expression evaluations do not result in all zero bits, the callable service is ended with a *control vector violation* return and reason code (8/39). See Figure 36 on page 1615. Only the 56 bit positions that are associated with a key value are evaluated. The low-order bit that is associated with key parity in each key byte is not evaluated.

Mask array preparation

A mask array consists of seven 8-byte elements: A_1, B_1, A_2, B_2, A_3, B_3, and B_4. You choose the values of the array elements such that each of the following four expressions evaluates to a string of binary zeros. (See Figure 36 on page 1615.) Set the **A** bits to the value that you require for the corresponding control vector bits. In expressions "1" on page 1613 through "3" on page 1613, set the **B** bits to select the control vector bits to be evaluated. In expression "4" on page 1613, set the **B** bits to select the source and target control vector bits to be evaluated. Also, use the following control vector information:

C_1 is the control vector associated with the left half of the KEK.
C_2 is the control vector associated with the source key, or selected source-key half/halves.
C_3 is the control vector associated with the target key or selected target-key half/halves.

1. (C_1 exclusive-OR A_1) logical-AND B_1

   This expression tests whether the KEK used to encipher the key meets your criteria for the desired translation.

2. (C_2 exclusive-OR A_2) logical-AND B_2

   This expression tests whether the control vector associated with the source key meets your criteria for the desired translation.

3. (C_3 exclusive-OR A_3) logical-AND B_3

   This expression tests whether the control vector associated with the target key meets your criteria for the desired translation.

4. (C_2 exclusive-OR C_3) logical-AND B_4

   This expression tests whether the control vectors associated with the source key and the target key meet your criteria for the desired translation.

Encipher two copies of the mask array, each under a different cryptographic-variable key (key type CVARENC). To encipher each copy of the mask array, use the Cryptographic Variable Encipher callable service. Use two different keys so that the enciphered-array copies are unique values. When using the Control Vector Translate callable service, the *mask_array_left* parameter and the *mask_array_right* parameter identify the enciphered mask arrays. The *array_key_left* parameter and the *array_key_right*

parameter identify the internal keys for deciphering the mask arrays. The *array_key_left* key must have a key type of CVARXCVL and the array_key_right key must have a key type of CVARXCVR. The cryptographic process deciphers the arrays and compares the results; for the service to continue, the deciphered arrays must be equal. If the results are not equal, the service returns the return and reason code for data that is not valid (8/385).

Use the Key Generate callable service to create the key pairs CVARENC-CVARXCVL and CVARENC-CVARXCVR. Each key in the key pair must be generated for a different node. The CVARENC keys are generated for, or imported into, the node where the mask array will be enciphered. After enciphering the mask array, you should destroy the enciphering key. The CVARXCVL and CVARXCVR keys are generated for, or imported into, the node where the Control Vector Translate callable service will be performed.

If using the B0TH keyword to process both halves of a double-length key, remember that bits 41, 42, 104, and 105 are different in the left and right halves of the CCA control vector and must be ignored in your mask-array tests (that is, make the corresponding $\mathbf{B_2}$ and/or $\mathbf{B_3}$ bits equal to zero).

When the control vectors pass the masking tests, the verb does the following:

- Deciphers the source key. In the decipher process, the service uses a key that is formed by the exclusive-OR of the KEK and the control vector in the key token variable the *source_key_token* parameter identifies.
- Enciphers the deciphered source key. In the encipher process, the service uses a key that is formed by the exclusive-OR of the KEK and the control vector in the key token variable the *target_key_token* parameter identifies.
- Places the enciphered key in the key field in the key token variable the *target_key_token* parameter identifies.

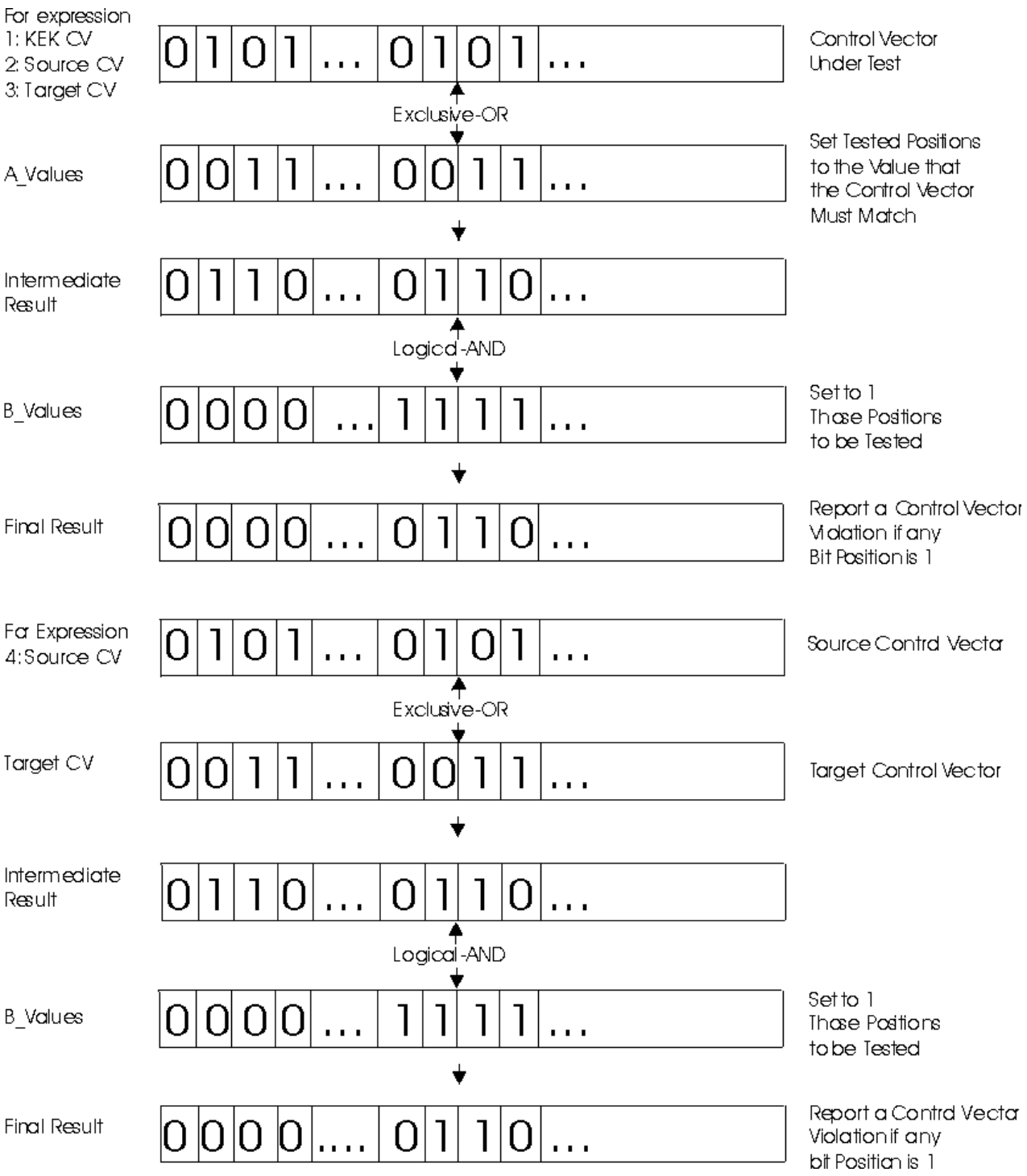


Figure 36. Control Vector Translate Callable Service Mask_Array Processing

Selecting the key-half processing mode

Use the Control Vector Translate callable service to change a control vector associated with a key. Rule-array keywords determine which key halves are processed in the call, as shown in Figure 37 on page 1616.

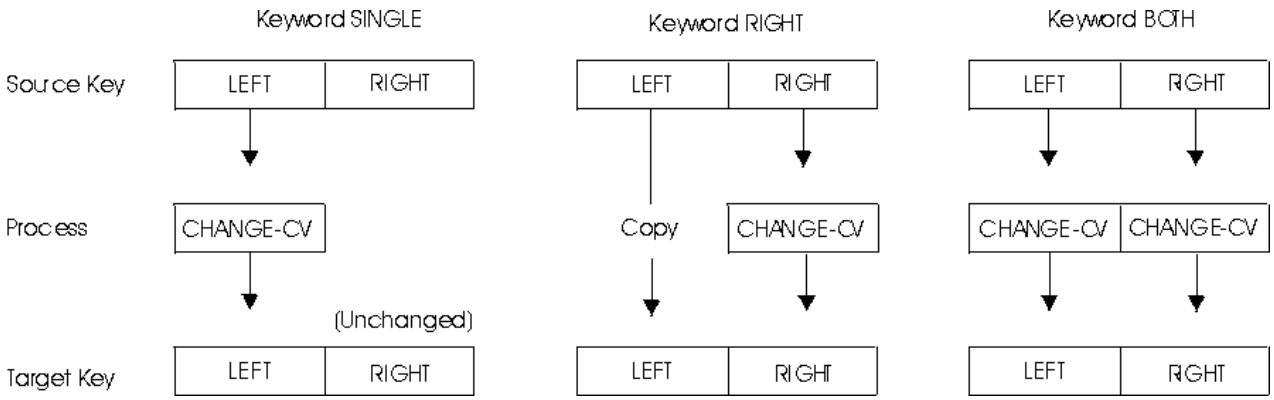


Figure 37. Control Vector Translate Callable Service

Keyword
Meaning

SINGLE
This keyword causes the control vector of the left half of the source key to be changed. The updated key half is placed into the left half of the target key in the target key token. The right half of the target key is unchanged.

The SINGLE keyword is useful when processing a single-length key, or when first processing the left half of a double-length key (to be followed by processing the right half).

RIGHT
This keyword causes the control vector of the right half of the source key to be changed. The updated key half is placed into the right half of the target key of the target key token. The left half of the source key is copied unchanged into the left half of the target key in the target key token.

BOTH
This keyword causes the control vector of both halves of the source key to be changed. The updated key is placed into the target key in the target key token.

A single set of control information must permit the control vector changes applied to each key half. Normally, control vector bit positions 41, 42, 105, and 106 are different for each key half. Therefore, set bits 41 and 42 to B'00' in mask array elements B_1, B_2, and B_3.

You can verify that the source and target key tokens have control vectors with matching bits in bit positions 40-42 and 104-106, the "form field" bits. Ensure that bits 40-42 of mask array B_4 are set to B'111'.

LEFT
This keyword enables you to supply a single-length key and obtain a double-length key. The source key token must contain:

- The KEK-enciphered single-length key
- The control vector for the single-length key (often this is a null value)
- A control vector, stored in the source token where the right-half control vector is normally stored, used in decrypting the single-length source key when the key is being processed for the target right half of the key.

The service first processes the source and target tokens as with the SINGLE keyword. Then the source token is processed using the single-length enciphered key and the source token right-half control vector to obtain the actual key value. The key value is then enciphered using the KEK and the control vector in the target token for the right-half of the key.

This approach is frequently of use when you must obtain a double-length CCA key from a system that only supports a single-length key, for example when processing PIN keys or key-encrypting keys received from non-CCA systems.

To prevent the service from ensuring that each key byte has odd parity, you can specify the NOADJUST keyword. If you do not specify the NOADJUST keyword, or if you specify the ADJUST keyword, the service ensures that each byte of the target key has odd parity.

When the target key token CV is null

When you use any of the LEFT, BOTH, or RIGHT keywords, and when the control vector in the target key token is null (all B'0'), then bit 3 in byte 59 will be set to B'1' to indicate that this is a double-length DATA key.

Control Vector Translate example

As an example, consider the case of receiving a single-length PIN-block encrypting key from a non-CCA system. Often such a key will be encrypted by an unmodified transport key (no control vector or variant is used). In a CCA system, an inbound PIN encrypting key is double-length.

First use the Key Token Build callable service to insert the single-length key value into the left-half key-space in a key token. Specify USE-CV as a key type and a control vector value set to 16 bytes of X'00'. Also specify EXTERNAL, KEY, and CV keywords in the rule array. This key token will be the source key key token.

Second, the target key token can also be created using the Key Token Build callable service. Specify a key type of IPINENC and the NO-EXPORT rule array keyword.

Then call the Control Vector Translate callable service and specify a rule-array keyword of LEFT. The mask arrays can be constructed as follows:

- A_1 is set to the value of the KEK's control vector, most likely the value of an IMPORTER key, perhaps with the NO-EXPORT bit set. B_1 is set to eight bytes of X'FF' so that all bits of the KEK's control vector will be tested.
- A_2 is set to eight bytes of X'00', the (null) value of the source key control vector. B_2 is set to eight bytes of X'FF' so that all bits of the source-key "control vector" will be tested.
- A_3 is set to the value of the target key's left-half control vector. B_3 is set to X'FFFF FFFF FF9F FFFF'. This will cause all bits of the control vector to be tested except for the two ("fff") bits used to distinguish between the left-half and right-half target-key control vector.
- B_4 is set to eight bytes of X'00' so that no comparison is made between the source and target control vectors.

Appendix D. Coding examples

This topic provides sample routines using the ICSF callable services for these languages:

- C
- COBOL
- High Level Assembler
- PL/I
- Rexx

The C, COBOL, and High Level Assembler examples that follow use the key generate, encipher, and decipher callable services to determine whether the deciphered text matches the starting text.

C

C programs must include the header file csfbext.h, which contains stubs for calling the ICSF services. This file is installed in the z/OS UNIX directory /usr/include and is copied to SYS1.SIEAHDR.H(CSFBEXT).

In addition, C applications that include csfbext.h must be link edited with the appropriate DLL sidedeck for the addressing model:

Standard 31-bit
Link with /usr/lib/CSFDLL31.x or SYS1.SIEASID(CSFDLL31)

31-bit with XPLINK
Link with /usr/lib/CSFDLL3X.x or SYS1.SIEASID(CSFDLL3X)

64-bit
Link with /usr/lib/CSFDLL64.x or SYS1.SIEASID(CSFDLL64)

Information on creating C applications that call ICSF PKCS #11 services is available in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

```
/*-------------------------------------------------------------------*
 * Example using C:                                                  *
 *   Invokes CSNBKGN (key generate), CSNBENC (DES encipher) and      *
 *   CSNBDEC (DES decipher)                                          *
 *-------------------------------------------------------------------*/
#include <stdio.h>
#include "csfbext.h"

/*-------------------------------------------------------------------*
 * Prototypes for functions in this example                          *
 *-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*
 * Utility for printing hex strings                                  *
 *-------------------------------------------------------------------*/
void printHex(unsigned char *, unsigned int);

/*********************************************************************/
/* Main Function                                                     */
/*********************************************************************/
int main(void) {

  /*-----------------------------------------------------------------*
   * Constant inputs to ICSF services                                *
   *-----------------------------------------------------------------*/
  static int textLen = 24;
  static unsigned char clearText[24]="ABCDEFGHIJKLMN0987654321";
  static unsigned char cipherProcessRule[8]="CUSP    ";
  static unsigned char keyForm[4]="OP  ";
  static unsigned char keyLength[8]="SINGLE  ";
  static unsigned char dataKeyType[8]="DATA    ";
  static unsigned char nullKeyType[8]="        ";
  static unsigned char ICV[8]={0};
  static int *pad=0;
```

```
static int exitDataLength = 0;
static unsigned char exitData[4]={0};
static int ruleArrayCount = 1;

/*-------------------------------------------------------------------*
 * Variable inputs/outputs for ICSF services          *
 *-------------------------------------------------------------------*/
unsigned char cipherText[24]={0};
unsigned char compareText[24]={0};
unsigned char dataKeyId[64]={0};
unsigned char nullKeyId[64]={0};
unsigned char dummyKEKKeyId1[64]={0};
unsigned char dummyKEKKeyId2[64]={0};
int returnCode = 0;
int reasonCode = 0;
unsigned char OCV[18]={0};

/*-------------------------------------------------------------------*
 * Begin executable code                                             *
 *-------------------------------------------------------------------*/
do {
  /*-----------------------------------------------------------------*
   * Call key generate                                               *
   *-----------------------------------------------------------------*/
  if ((returnCode = CSNBKGN(&returnCode,
                            &reasonCode,
                            &exitDataLength,
                            exitData,
                            keyForm,
                            keyLength,
                            dataKeyType,
                            nullKeyType,
                            dummyKEKKeyId1,
                            dummyKEKKeyId2,
                            dataKeyId,
                            nullKeyId)) != 0) {
    printf("\nKey Generate failed:\n");
    printf("   Return Code = %04d\n",returnCode);
    printf("   Reason Code = %04d\n",reasonCode);
    break;
    }
 /*-----------------------------------------------------------------*
  * Call encipher                                                   *
  *-----------------------------------------------------------------*/
 printf("\nClear Text\n");
 printHex(clearText,sizeof(clearText));

 if ((returnCode = CSNBENC(&returnCode,
                           &reasonCode,
                           &exitDataLength,
                           exitData,
                           dataKeyId,
                           &textLen,
                           clearText,
                           ICV,
                           &ruleArrayCount,
                           cipherProcessRule,
                           &pad,
                           OCV,
                           cipherText)) != 0) {
   printf("\nReturn from Encipher:\n");
   printf("   Return Code = %04d\n",returnCode);
   printf("   Reason Code = %04d\n",reasonCode);
   if (returnCode > 4)
     break;
   }

 /*-----------------------------------------------------------------*
  * Call decipher                                                   *
  *-----------------------------------------------------------------*/
 printf("\nCipher Text\n");
 printHex(cipherText,sizeof(cipherText));

 if ((returnCode = CSNBDEC(&returnCode,
                           &reasonCode,
                           &exitDataLength,
                           exitData,
                           dataKeyId,
                           &textLen,
                           cipherText,
                           ICV,
                           &ruleArrayCount,
```

```
                            cipherProcessRule,
                            OCV,
                            compareText)) != 0) {
     printf("\nReturn from Decipher:\n");
     printf("   Return Code = %04d\n",returnCode);
     printf("   Reason Code = %04d\n",reasonCode);
     if (returnCode > 4)
       break;
     }

   /*-----------------------------------------------------------------*
    * End                                                             *
    *-----------------------------------------------------------------*/
   printf("\nClear Text after decipher\n");
   printHex(compareText,sizeof(compareText));

   } while(0);

   return returnCode;

} /* end main */

void printHex (unsigned char * text, unsigned int len)
/*-----------------------------------------------------------------*
 * Prints a string as hex characters                               *
 *-----------------------------------------------------------------*/

{
  unsigned int i;

  for (i = 0; i < len; ++i)
    if ( ((i & 7) == 7) || (i == (len - 1)) )
      printf ("  %02x\n", text[i]);
    else
      printf ("  %02x", text[i]);
  printf ("\n");
} /* end printHex */
```

COBOL

```
      **************************
       IDENTIFICATION DIVISION.
      **************************
       PROGRAM-ID. COBOLXMP.
      *****************************
       ENVIRONMENT DIVISION.
      *********************************************************************
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-370.
       OBJECT-COMPUTER.  IBM-370.
      *****************************
       DATA DIVISION.
      *********************************************************************
       FILE SECTION.
       WORKING-STORAGE SECTION.
       77  INPUT-TEXT                    PIC       X(24)
           VALUE 'ABCDEFGHIJKLMN0987654321'.
       77  OUTPUT-TEXT                   PIC       X(24)
           VALUE LOW-VALUES.
       77  COMPARE-TEXT                  PIC       X(24)
           VALUE LOW-VALUES.
       77  CIPHER-PROCESSING-RULE        PIC       X(08)
           VALUE 'CUSP    '.
       77  KEY-FORM                      PIC       X(08)
           VALUE 'OP      '.
       77  KEY-LENGTH                    PIC       X(08)
           VALUE 'SINGLE  '.
       77  KEY-TYPE-1                    PIC       X(08)
           VALUE 'DATA    '.
       77  KEY-TYPE-2                    PIC       X(08)
           VALUE '        '.
       77  ICV                           PIC       X(08)
           VALUE LOW-VALUES.
       77  PAD                           PIC       X(01)
           VALUE LOW-VALUES.
      ************* DEFINE SAPI INPUT/OUTPUT PARAMETERS ************
       01  SAPI-REC.
           05  RETURN-CODE-S             PIC       9(08) COMP.
```

```
           05  REASON-CODE-S             PIC       9(08) COMP.
           05  EXIT-DATA-LENGTH-S        PIC       9(08) COMP.
           05  EXIT-DATA-S               PIC       X(04).
           05  KEK-KEY-ID-1-S            PIC       X(64)
               VALUE LOW-VALUES.
           05  KEK-KEY-ID-2-S            PIC       X(64)
               VALUE LOW-VALUES.
           05  DATA-KEY-ID-S             PIC       X(64)
               VALUE LOW-VALUES.
           05  NULL-KEY-ID-S             PIC       X(64)
               VALUE LOW-VALUES.
           05  KEY-FORM-S                PIC       X(08).
           05  KEY-LENGTH-S              PIC       X(08).
           05  DATA-KEY-TYPE-S           PIC       X(08).
           05  NULL-KEY-TYPE-S           PIC       X(08).
           05  TEXT-LENGTH-S             PIC       9(08) COMP.
           05  TEXT-S                    PIC       X(24).
           05  ICV-S                     PIC       X(08).
           05  PAD-S                     PIC       X(01).
           05  CPHR-TEXT-S               PIC       X(24).
           05  COMP-TEXT-S               PIC       X(24).
           05  RULE-ARRAY-COUNT-S        PIC       9(08) COMP.
           05  RULE-ARRAY-S.
               10  RULE-ARRAY            PIC       X(08).
           05  CHAINING-VECTOR-S         PIC       X(18).
      ****************************************************************
       PROCEDURE DIVISION.
      ****************************************************************
       MAIN-RTN.
      *************  CALL KEY GENERATE  ***************************
           MOVE 0               TO   EXIT-DATA-LENGTH-S.
           MOVE KEY-FORM        TO   KEY-FORM-S.
           MOVE KEY-LENGTH      TO   KEY-LENGTH-S.
           MOVE KEY-TYPE-1      TO   DATA-KEY-TYPE-S.
           MOVE KEY-TYPE-2      TO   NULL-KEY-TYPE-S.
           CALL 'CSNBKGN'   USING RETURN-CODE-S
                                  REASON-CODE-S
                                  EXIT-DATA-LENGTH-S
                                  EXIT-DATA-S
                                  KEY-FORM-S
                                  KEY-LENGTH-S
                                  DATA-KEY-TYPE-S
                                  NULL-KEY-TYPE-S
                                  KEK-KEY-ID-1-S
                                  KEK-KEY-ID-2-S
                                  DATA-KEY-ID-S
                                  NULL-KEY-ID-S.
           IF RETURN-CODE-S NOT = 0 OR
              REASON-CODE-S NOT = 0 THEN
              DISPLAY '*** KEY-GENERATE ***'
              DISPLAY '*** RETURN-CODE = ' RETURN-CODE-S
              DISPLAY '*** REASON-CODE = ' REASON-CODE-S
           ELSE
              MOVE 24                  TO  TEXT-LENGTH-S
              MOVE INPUT-TEXT          TO  TEXT-S
              MOVE 1                   TO  RULE-ARRAY-COUNT-S
              MOVE CIPHER-PROCESSING-RULE TO RULE-ARRAY-S
              MOVE LOW-VALUES          TO  CHAINING-VECTOR-S
              MOVE ICV             TO   ICV-S.
              MOVE PAD             TO   PAD-S.
      *************  CALL ENCIPHER ********************************
              CALL 'CSNBENC'  USING    RETURN-CODE-S
                                       REASON-CODE-S
                                       EXIT-DATA-LENGTH-S
                                       EXIT-DATA-S
                                       DATA-KEY-ID-S
                                       TEXT-LENGTH-S
                                       TEXT-S
                                       ICV-S
                                       RULE-ARRAY-COUNT-S
                                       RULE-ARRAY-S
                                       PAD-S
                                       CHAINING-VECTOR-S
                                       CPHR-TEXT-S
              IF RETURN-CODE-S NOT = 0 OR
                 REASON-CODE-S NOT = 0 THEN
                 DISPLAY '*** ENCIPHER ***'
                 DISPLAY '*** RETURN-CODE = ' RETURN-CODE-S
                 DISPLAY '*** REASON-CODE = ' REASON-CODE-S
              ELSE
      *************  CALL DECIPHER ********************************
                 CALL 'CSNBDEC' USING RETURN-CODE-S
```

```
                                    REASON-CODE-S
                                    EXIT-DATA-LENGTH-S
                                    EXIT-DATA-S
                                    DATA-KEY-ID-S
                                    TEXT-LENGTH-S
                                    CPHR-TEXT-S
                                    ICV-S
                                    RULE-ARRAY-COUNT-S
                                    RULE-ARRAY-S
                                    CHAINING-VECTOR-S
                                    COMP-TEXT-S
                IF RETURN-CODE-S  NOT = 0 OR
                   REASON-CODE-S  NOT = 0 THEN
                   DISPLAY '*** DECIPHER ***'
                   DISPLAY '*** RETURN-CODE = ' RETURN-CODE-S
                   DISPLAY '*** REASON-CODE = ' REASON-CODE-S
               ELSE
                   IF COMP-TEXT-S = TEXT-S THEN
                     DISPLAY '*** DECIPHERED TEXT = PLAIN TEXT ***'
                   ELSE
                     DISPLAY '*** DECIPHERED TEXT ê= PLAIN TEXT ***'.
           DISPLAY '*** TEST PROGRAM ENDED ***'
           STOP RUN.
```

High Level Assembler

```
         TITLE 'SAMPLE ENCIPHER/DECIPHER S/370 PROGRAM.'
*=====================================================================*
*       SYSTEM/370 High Level Assembler EXAMPLE                              *
*                                                                    *
*=====================================================================*
         SPACE
SAMPLE   START 0
         DS    0H
         STM   14,12,12(13)     SAVE REGISTERS
         BALR  12,0             USE R12 AS BASE REGISTER
         USING *,12             PROVIDE SAVE AREA FOR SUBROUTINE
         LA    14,SAVE          PERFORM SAVE AREA CHAINING
         ST    13,4(14)           "
         ST    14,8(13)           "
         LR    13,14              "
*
         CALL  CSFKGN,(RETCD,                                        *
               RESCD,                                                *
               EXDATAL,                                              *
               EXDATA,                                               *
               KEY_FORM,                                             *
               KEY_LEN,                                              *
               KEYTYP1,                                              *
               KEYTYP2,                                              *
               KEK_ID1,                                              *
               KEK_ID2,                                              *
               DATA_ID,                                              *
               NULL_ID)
         CLC   RETCD,=F'0'      CHECK RETURN CODE
         BNE   BACK             OUTPUT RETURN/REASON CODE AND STOP
         CLC   RESCD,=F'0'      CHECK REASON CODE
         BNE   BACK             OUTPUT RETURN/REASON CODE AND STOP
*
* CALL ENCIPHER WITH THE KEY JUST GENERATED
*  OPERATIONAL FORM
*
         MVC   RULEAC,=F'1'           SET RULE ARRAY COUNT
         MVC   RULEA,=CL8'CUSP    '   BUILD RULE ARRAY
         CALL CSFENC,(RETCD,                                         *
               RESCD,                                                *
               EXDATAL,                                              *
               EXDATA,                                               *
               DATA_ID,                                              *
               TEXTL,                                                *
               TEXT,                                                 *
               ICV,                                                  *
               RULEAC,                                               *
               RULEA,                                                *
               PAD_CHAR,                                             *
               OCV,                                                  *
               CIPHER_TEXT)
         CLC   RETCD,=F'0'      CHECK RETURN CODE
         BNE   BACK             OUTPUT RETURN/REASON CODE AND STOP
```

```
         CLC   RESCD,=F'0'      CHECK REASON CODE
         BNE   BACK             OUTPUT RETURN/REASON CODE AND STOP
         CALL CSFDEC,(RETCD,                                          *
               RESCD,                                                 *
               EXDATAL,                                               *
               EXDATA,                                                *
               DATA_ID,                                               *
               TEXTL,                                                 *
               CIPHER_TEXT,                                           *
               ICV,                                                   *
               RULEAC,                                                *
               RULEA,                                                 *
               OCV,                                                   *
               NEW_TEXT)
         CLC   RETCD,=F'0'      CHECK RETURN CODE
         BNE   BACK             OUTPUT RETURN/REASON CODE AND STOP
         CLC   RESCD,=F'0'      CHECK REASON CODE
         BNE   BACK             OUTPUT RETURN/REASON CODE AND STOP
*
COMPARE  EQU   *                        COMPARE START AND END TEXT
         CLC   TEXT,NEW_TEXT
         BE    GOODENC
         WTO   'DECIPHERED TEXT DOES NOT MATCH STARTING TEXT'
         B      BACK
GOODENC  WTO   'DECIPHERED TEXT MATCHES STARTING TEXT'
*
*
         WTO   'TEST PROGRAM TERMINATING'
         B     RETURN
*
*-------------------------------------------------
* CONVERT RETURN/REASON CODES FROM BINARY TO EBCDIC
*-------------------------------------------------
BACK     DS    0F               OUTPUT RETURN & REASON CODE
         L     5,RETCD          LOAD RETURN CODE
         L     6,RESCD          LOAD REASON CODE
         CVD   5,BCD1           CONVERT TO PACK-DECIMAL
         CVD   6,BCD2
         UNPK  ORETCD,BCD1      CONVERT TO EBCDIC
         UNPK  ORESCD,BCD2
         OI    ORETCD+7,X'F0'   CORRECT LAST DIGIT
         OI    ORESCD+7,X'F0'
*
         MVC   ERROUT+21(4),ORETCD+4
         MVC   ERROUT+41(4),ORESCD+4
ERROUT   WTO   'ERROR CODE =     , REASON CODE =     '
RETURN   EQU   *
         L     13,4(13)         SAVE AREA RESTORATION
         MVC   16(4,13),RETCD   SAVE RETURN CODE
         LM    14,12,12(13)
         BR    14               RETURN TO CALLER
*
BCD1     DS    D                CONVERT TO BCD TEMP AREA
BCD2     DS    D                CONVERT TO BCD TEMP AREA
ORETCD   DS    CL8'0'           OUTPUT RETURN CODE
ORESCD   DS    CL8'0'           OUTPUT REASON CODE
*
KEY_FORM DC    CL8'OP      '    KEY FORM
KEY_LEN  DC    CL8'SINGLE  '    KEY LENGTH
KEYTYP1  DC    CL8'DATA    '    KEY TYPE 1
KEYTYP2  DC    CL8'        '    KEY TYPE 2
TEXT     DC    C'ABCDEFGHIJKLMNOPQRSTUV0987654321'
TEXTL    DC    F'32'            TEXT LENGTH
CIPHER_TEXT DC CL32' '
NEW_TEXT DC    CL32' '
DATA_ID  DC    XL64'00'         DATA KEY TOKEN
NULL_ID  DC    XL64'00'         NULL KEY TOKEN - UNFILLED
KEK_ID1  DC    XL64'00'         KEK1 KEY TOKEN
KEK_ID2  DC    XL64'00'         KEK2 KEY TOKEN
RETCD    DS    F'0'             RETURN CODE
RESCD    DS    F'0'             REASON CODE
EXDATAL  DC    F'0'             EXIT DATA LENGTH
EXDATA   DS    0C               EXIT DATA
RULEA    DS    1CL8             RULE ARRAY
RULEAC   DS    F'0'             RULE ARRAY COUNT
ICV      DC    XL8'00'          INITIAL CHAINING VECTOR
OCV      DC    XL18'00'         OUTPUT CHAINING VECTOR
PAD_CHAR DC    F'0'             PAD CHARACTER
SAVE     DS    18F              SAVE REGISTER AREA
         END   SAMPLE
```

PL/I

```
/*********************************************************************/
/*                                                                   */
/* Sample program to call the one-way hash service to generate       */
/* the SHA-1 hash of the input text and call digital signature       */
/* generate with an RSA key using the ISO 9796 text formatting. The  */
/* RSA key token is built from supplied data and imported for the    */
/* signature generate service to use.                                */
/*                                                                   */
/*      INPUT:  TEXT          Message digest to be signed            */
/*                                                                   */
/*      OUTPUT: SIGNATURE_LENGTH  Length of the signature in bytes   */
/*                                Written to a dataset.              */
/*                                                                   */
/*              SIGNATURE     Signature for hash.  Written to a      */
/*                            dataset.                               */
/*                                                                   */
/*********************************************************************/
DSIGEXP:PROCEDURE( TEXT ) OPTIONS( MAIN );

/* Declarations - Parameters                                         */

DCL TEXT             CHAR( 64 ) VARYING;

/* Declarations - API parameters                                     */

DCL CHAINING_VECTOR_LENGTH  FIXED BINARY( 31, 0 ) INIT( 128 );
DCL CHAINING_VECTOR         CHAR( 128 );
DCL DUMMY_KEK               CHAR( 64 );
DCL EXIT_DATA               CHAR( 4 );
DCL EXIT_LEN                FIXED BINARY( 31, 0 ) INIT( 0 );

DCL HASH                    CHAR( 20 );
DCL HASH_LENGTH             FIXED BINARY( 31, 0 ) INIT( 20 );

DCL INTERNAL_PKA_TOKEN      CHAR( 1024 );
DCL INTERNAL_PKA_TOKEN_LENGTH FIXED BINARY( 31, 0 );

DCL KEY_VALUE_STRUCTURE     CHAR(139)
                         INIT(( '020000400003004080000000000000000'X ||
                                '01AE28DA4606D885EB7E0340D6BAAC51'X ||
                                '991C0CD0EAE835AFD9CFF3CD7E7EA741'X ||
                                '41DADD24A6331BEDF41A6626522CCF15'X ||
                                '767D167D01A16F9701000102 52BDAD42'X ||
                                '52BDAD425A8C6045D41AFAF746BEBD5F'X ||
                                '085D574FCD9C07F0B38C2C45017C2A1A'X ||
                                'B919ED2551350A76606BFA6AF2F1609A'X ||
                                '00A0A48DD719A55E9CA801'X ));
DCL KEY_VALUE_LENGTH        FIXED BINARY( 31, 0 ) INIT( 139 );

DCL OWH_TEXT                CHAR( 64 );

DCL PKA_KEY_TOKEN           CHAR( 1024 );
DCL PKA_TOKEN_LENGTH        FIXED BINARY( 31, 0 );

DCL PRIVATE_NAME            CHAR( 64 ) INIT( 'PL1.EXAMPLE.FOR.APG' );
DCL PRIVATE_NAME_LENGTH     FIXED BINARY( 31, 0 ) INIT( 0 );

DCL RETURN_CODE             FIXED BINARY( 31, 0 ) INIT( 0 );
DCL REASON_CODE             FIXED BINARY( 31, 0 ) INIT( 0 );

DCL RESERVED_FIELD_LENGTH   FIXED BINARY( 31, 0 ) INIT( 0 );
DCL RESERVED_FIELD          CHAR( 1 );

DCL RULE_ARY_CNT_DSG        FIXED BINARY( 31, 0 ) INIT( 1 );
DCL RULE_ARY_CNT_PKB        FIXED BINARY( 31, 0 ) INIT( 1 );
DCL RULE_ARY_CNT_PKI        FIXED BINARY( 31, 0 ) INIT( 0 );
DCL RULE_ARY_CNT_OWH        FIXED BINARY( 31, 0 ) INIT( 2 );
DCL RULE_ARY_DSG            CHAR( 8 ) INIT( 'ISO-9796' );
DCL RULE_ARY_PKB            CHAR( 8 ) INIT( 'RSA-PRIV' );
DCL RULE_ARY_PKI            CHAR( 8 );
DCL RULE_ARY_OWH            CHAR( 16 ) INIT( 'SHA-1   ONLY    ' );

DCL SIGNATURE_LENGTH        FIXED BINARY( 31, 0 );
DCL SIGNATURE               CHAR( 128 );
DCL SIG_BIT_LENGTH          FIXED BINARY( 31, 0 );

DCL TEXT_LENGTH             FIXED BINARY( 31, 0 );
```

```
/* Declarations - Files and entry points                            */

DCL SYSPRINT  FILE OUTPUT;
DCL SIGOUT    FILE RECORD OUTPUT;

DCL CSNDPKB   ENTRY EXTERNAL OPTIONS( ASM, INTER );
DCL CSNDPKI   ENTRY EXTERNAL OPTIONS( ASM, INTER );
DCL CSNBOWH   ENTRY EXTERNAL OPTIONS( ASM, INTER );
DCL CSNDDSG   ENTRY EXTERNAL OPTIONS( ASM, INTER );

/* Declarations - Internal variables                                */

DCL DSG_HEADER       CHAR( 32 )
                     INIT( '* DIGITAL SIGNATURE GENERATION *' );
DCL FILE_OUT_LINE    CHAR( 128 );
DCL OWH_HEADER       CHAR( 16 )
                     INIT( '* ONE WAY HASH *' );
DCL PKB_HEADER       CHAR( 16 )
                     INIT( '* PKA TOKEN BUILD *' );
DCL PKI_HEADER       CHAR( 16 )
                     INIT( '* PKA TOKEN IMPORT *' );
DCL RC_STRING        CHAR( 14 ) INIT( 'RETURN CODE = ' );
DCL RS_STRING        CHAR( 14 ) INIT( 'REASON CODE = ' );
DCL SIG_STRING       CHAR( 12 ) INIT( 'SIGNATURE = ' );
DCL SIG_LEN_STRING   CHAR( 26 ) INIT( 'SIGNATURE LENGTH(BYTES) = ' );

/* Declarations - Built-in functions                                */

DCL (SUBSTR, LENGTH) BUILTIN;

/*******************************************************************/
/* Call one-way hash to get the SHA-1 hash of the text.            */
/*******************************************************************/
TEXT_LENGTH = LENGTH( TEXT );
OWH_TEXT = SUBSTR( TEXT, 1, TEXT_LENGTH );

CALL CSNBOWH( RETURN_CODE,
              REASON_CODE,
              EXIT_LEN,
              EXIT_DATA,
              RULE_ARY_CNT_OWH,
              RULE_ARY_OWH,
              TEXT_LENGTH,
              OWH_TEXT,
              CHAINING_VECTOR_LENGTH,
              CHAINING_VECTOR,
              HASH_LENGTH,
              HASH );

PUT SKIP LIST( OWH_HEADER );
PUT SKIP LIST( RC_STRING || RETURN_CODE );
PUT SKIP LIST( RS_STRING || REASON_CODE );

/*******************************************************************/
/* Create the PKA RSA private external token.                      */
/*******************************************************************/
IF RETURN_CODE = 0 THEN
  DO;

  PKA_TOKEN_LENGTH = 1024;

  CALL CSNDPKB( RETURN_CODE,
                REASON_CODE,
                EXIT_LEN,
                EXIT_DATA,
                RULE_ARY_CNT_PKB,
                RULE_ARY_PKB,
                KEY_VALUE_LENGTH,
                KEY_VALUE_STRUCTURE,
                PRIVATE_NAME_LENGTH,
                PRIVATE_NAME,
                RESERVED_FIELD_LENGTH,
                RESERVED_FIELD,
                RESERVED_FIELD_LENGTH,
                RESERVED_FIELD,
                RESERVED_FIELD_LENGTH,
                RESERVED_FIELD,
                RESERVED_FIELD_LENGTH,
                RESERVED_FIELD,
                RESERVED_FIELD_LENGTH,
                RESERVED_FIELD,
                PKA_TOKEN_LENGTH,
```

```
                PKA_KEY_TOKEN );

  PUT SKIP LIST( PKB_HEADER );
  PUT SKIP LIST( RC_STRING || RETURN_CODE );
  PUT SKIP LIST( RS_STRING || REASON_CODE );

  END;

/*********************************************************************/
/* Import the clear RSA private external token.                      */
/*********************************************************************/
IF RETURN_CODE = 0 THEN
  DO;

  INTERNAL_PKA_TOKEN_LENGTH = 1024;

  CALL CSNDPKI( RETURN_CODE,
                REASON_CODE,
                EXIT_LEN,
                EXIT_DATA,
                RULE_ARY_CNT_PKI,
                RULE_ARY_PKI,
                PKA_TOKEN_LENGTH,
                PKA_KEY_TOKEN,
                DUMMY_KEK,
                INTERNAL_PKA_TOKEN_LENGTH,
                INTERNAL_PKA_TOKEN );

  PUT SKIP LIST( PKI_HEADER );
  PUT SKIP LIST( RC_STRING || RETURN_CODE );
  PUT SKIP LIST( RS_STRING || REASON_CODE );

  END;
/*********************************************************************/
/* Call digital signature generate.                                  */
/*********************************************************************/
IF RETURN_CODE = 0 THEN
  DO;

  SIGNATURE_LENGTH = 128;

  CALL CSNDDSG( RETURN_CODE,
                REASON_CODE,
                EXIT_LEN,
                EXIT_DATA,
                RULE_ARY_CNT_DSG,
                RULE_ARY_DSG,
                INTERNAL_PKA_TOKEN_LENGTH,
                INTERNAL_PKA_TOKEN,
                HASH_LENGTH,
                HASH,
                SIGNATURE_LENGTH,
                SIG_BIT_LENGTH,
                SIGNATURE );

  PUT SKIP LIST( DSG_HEADER );
  PUT SKIP LIST( RC_STRING || RETURN_CODE );
  PUT SKIP LIST( RS_STRING || REASON_CODE );

  IF RETURN_CODE = 0 THEN
    DO;

    /*****************************************************************/
    /* Write the signature and its length to the output file.        */
    /*****************************************************************/
    FILE_OUT_LINE = SIG_LEN_STRING || SIGNATURE_LENGTH;
    WRITE FILE(SIGOUT) FROM( FILE_OUT_LINE );
    FILE_OUT_LINE = SIG_STRING || SIGNATURE;
    WRITE FILE(SIGOUT) FROM( FILE_OUT_LINE );
    END;

  END;

END DSIGEXP;
```

Rexx

```
/* Rexx */
```

```
/*--------------------------------------------------------------------*/
/* This sample will convert an existing RSA private key so that it    */
/* can be used with the PKCS-PSS digital-signature hash formatting    */
/* method.                                                            */
/*                                                                    */
/* The RSA key to be converted must be an existing secure key         */
/* encrypted under the RSA master key.  The RSA key may be in         */
/* modulus-exponent (ME) format or Chinese Remainder Theorem (CRT)    */
/* format.  The RSA key token formats are described in the ICSF       */
/* Application Programmer's Guide (APG), Appendix "Key token formats".*/
/*                                                                    */
/* See the ICSF APG for more detailed information on the callable     */
/* services used in this sample.                                      */
/*                                                                    */
/* PKCS-PSS formatting method is supported on ICSF HCR77C0 and CEX5C  */
/* and above.  The coprocessor ECC master key must be active. If the  */
/* ECC master key is not active, see the ICSF Administrator's Guide   */
/* "Updating the key data sets with additional master keys".          */
/*--------------------------------------------------------------------*/

/* existing RSA private key label to convert */
existing_RSA_key_label = left('SAMPLE.RSA.CRT.MOD2048',64) ;

/* converted RSA private key label */
converted_RSA_private_key = left('SAMPLE.RSA.CRT.MOD2048.PSS',64) ;

/*-------------------*/
/* PKA Key Translate */
/*-------------------*/
PKT_rc            = 'FFFFFFFF'x ;
PKT_rs            = 'FFFFFFFF'x ;
exit_data_length  = '00000000'x ;
exit_data         = '' ;
rule_array_count  = d2c(2,4) ;
rule_array        = 'INTDWAKW'||,
                    'FR-PSS  ' ; /* format restriction keyword */
/* Once converted, this key may only be used with the PKCS-PSS
   digital-signature hash formatting method. For no restriction
   on usage, specify FR-NONE.  See the ICSF Application
   Programmer's Guide for more information.
*/
source_key_length = d2c(64,4) ;
source_key        = existing_RSA_key_label ;
source_xport_key_length = d2c(0,4) ;
source_xport_key  = '' ;
target_xport_key_length = d2c(0,4) ;
target_xport_key  = '' ;
target_key_length = d2c(3500,4) ;
target_key        = d2c(0,3500) ;

/* CALL CSNDPKT */
ADDRESS LINKPGM 'CSNDPKT' ,
                'PKT_rc' ,
                'PKT_rs' ,
                'exit_data_length' ,
                'exit_data' ,
                'rule_array_count' ,
                'rule_array' ,
                'source_key_length' ,
                'source_key' ,
                'source_xport_key_length' ,
                'source_xport_key' ,
                'target_xport_key_length' ,
                'target_xport_key' ,
                'target_key_length' ,
                'target_key' ;

IF PKT_rc /= '00000000'x THEN
  DO ;
   SAY 'PKT failed: rc =' c2x(PKT_rc) 'rs =' c2x(PKT_rs) ;
   EXIT ;
  END ;


/* Write converted RSA private key to PKDS */
key_label        = converted_RSA_private_key ;
key_token_length = target_key_length ;
key_token        = target_key ;
CALL PKRC ;

/*--------------------------------------------------------------------*/
/* Use the converted RSA private key to generate a signature using */
```

```
/* the PKCS-PSS digital signature formatting hash method.          */
/*-----------------------------------------------------------------*/
DSG_rc             = 'FFFFFFFF'x ;
DSG_rs             = 'FFFFFFFF'x ;
exit_data_length   = '00000000'x ;
exit_data          = '' ;
rule_array_count   = '00000004'x ;
rule_array         = 'RSA     '||,
                     'PKCS-PSS'||,
                     'HASH    '||,
                     'SHA-256 ' ;
private_key_length = d2c(64,4) ;
private_key        = converted_RSA_private_key ;
data_length        = '00000024'x ;
data               = '00000020'x||,       /* salt length */
  '9EFDE926830891B7F2889646D0105BD8'x||,  /* hash        */
  '09C64F6217EC046F5B384F625C9CCF66'x ;
sig_field_length   = '00000100'x ;        /* 256 decimal */
sig_bit_length     = '00000800'x ;        /* 2048 decimal */
sig_field          = copies('00'x,256) ;

/* CALL CSNDDSG */
ADDRESS LINKPGM 'CSNDDSG' ,
                'DSG_rc' ,
                'DSG_rs' ,
                'exit_data_length' ,
                'exit_data' ,
                'rule_array_count' ,
                'rule_array' ,
                'private_key_length' ,
                'private_key' ,
                'data_length' ,
                'data' ,
                'sig_field_length' ,
                'sig_bit_length' ,
                'sig_field' ;

IF DSG_rc /= '00000000'x THEN
  SAY 'DSG failed: rc =' c2x(DSG_rc) 'rs =' c2x(DSG_rs) ;
ELSE
 DO ;
  sig_field = substr(sig_field,1,c2d(sig_field_length)) ;
  SAY 'signature field length:' c2x(sig_field_length) ;
  SAY 'signature bit length:' c2x(sig_bit_length) ;
  SAY 'signature:' c2x(sig_field) ;
 END ;


EXIT ;

/*------------------------*/
/* PKDS Key Record Create */
/*------------------------*/
PKRC:

PKRC_rc = 'FFFFFFFF'x ;
PKRC_rs = 'FFFFFFFF'x ;
exit_data_length = '00000000' ;
exit_data        = '' ;
rule_array_count = '00000000'x ;
rule_array       = '' ;

/* CALL CSNDKRC */
ADDRESS LINKPGM 'CSNDKRC' ,
                'PKRC_rc' ,
                'PKRC_rs' ,
                'exit_data_length' ,
                'exit_data' ,
                'rule_array_count' ,
                'rule_array' ,
                'key_label' ,
                'key_token_length' ,
                'key_token' ;

IF PKRC_rc /= '00000000'x THEN
  DO ;
   SAY 'PKRC failed: rc =' c2x(PKRC_rc) 'rs =' c2x(PKRC_rs) ;
   EXIT ;
  END ;

RETURN ;
```

Appendix E. Cryptographic algorithms and processes

This topic describes the personal identification number (PIN) formats and algorithms.

PIN formats and algorithms

For PIN calculation procedures, see IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface reference.

PIN Notation

This section describes various PIN block formats. The following notations describe the contents of PIN blocks:

P =
A 4-bit decimal digit that is one digit of the PIN value.

C =
A 4-bit hexadecimal control value. The valid values are X'0', X'1', X'2', X'3', and X'4'.

L =
A 4-bit hexadecimal value that specifies the number of PIN digits. The value ranges from 4 to 12, inclusive.

F =
A 4-bit field delimiter of value X'F'.

f =
A 4-bit delimiter filler that is either P or F, depending on the length of the PIN.

D =
A 4-bit decimal padding value. All pad digits in the PIN block have the same value.

X =
A 4-bit hexadecimal padding value. All pad digits in the PIN block have the same value.

x =
A 4-bit hexadecimal filler that is either P or X, depending on the length of the PIN.

R =
A 4-bit hexadecimal random digit. The sequence of R digits can each take a different value.

r =
A 4-bit random filler that is either P or R, depending on the length of the PIN.

Z =
A 4-bit hexadecimal zero (X'0').

z =
A 4-bit zero filler that is either P or Z, depending on the length of the PIN.

S =
A 4-bit hexadecimal digit that constitutes one digit of a sequence number.

U =
A 4-bit decimal digit that constitutes one digit of a user-specified constant.

A =
A 4-bit fill digit of value X'A'.

a =
A 4-bit value that is either P or A, depending on the length of the PIN.

PIN block formats

This topic describes the PIN block formats and assigns a code to each format.

ANSI X9.8

This format is also named ISO format 0, VISA format 1, VISA format 4, and ECI format 1.

```
   P1 = CLPPPPffffffffFF

   P2 = ZZZZUUUUUUUUUUUU

   PIN Block = P1 XOR P2

where C = X’0’
      L = X’4’ to X’C’
```

Programming Note: The rightmost 12 digits (excluding the check digit) in P2 are the rightmost 12 digits of the account number for all formats except VISA format 4. For VISA format 4, the rightmost 12 digits (excluding the check digit) in P2 are the leftmost 12 digits of the account number.

ISO Format 1

This format is also named ECI format 4.

```
   PIN Block = CLPPPPrrrrrrrrRR

where C = X'1'
      L = X'4' to X'C'
```

ISO Format 2

```
   PIN Block = CLPPPPffffffffFF

where C = X'2'
      L = X'4' to X'C'
```

ISO Format 3

```
    P1 = CLPPPPrrrrrrrrRR

    P2 = ZZZZAAAAAAAAAAAA

    PIN Block = P1 XOR P2

where C = X'3'
      L = X'4' to X'C'
```

ISO Format 4

This block is 16 bytes long.

```
   PIN Block = CLPPPPaaaaaaaaAARRRRRRRRRRRRRRRR

where C = X’4’
      L = X’4’ to X’C’
```

VISA Format 2

```
   PIN Block = LPPPPzzDDDDDDDDD

where L = X'4' to X'6'
```

VISA Format 3

This format specifies that the PIN length can be 4-12 digits, inclusive. The PIN starts from the leftmost digit and ends by the delimiter (‘F’), and the remaining digits are padding digits.

An example of a 6-digit PIN:

```
   PIN Block = PPPPPPFXXXXXXXXX
```

IBM 4700 Encrypting PINPAD Format

This format uses the value X'F' as the delimiter for the PIN.

```
   PIN Block = LPPPPfffffffffFSS
where L = X'4' to X'C'
```

IBM 3624 Format

This format requires the program to specify the delimiter, X, for determining the PIN length.

```
   PIN Block = PPPPxxxxxxxxXXXX
```

IBM 3621 Format

This format requires the program to specify the delimiter, X, for determining the PIN length.

```
   PIN Block = SSSSPPPPxxxxxxxx
```

ECI Format 2

This format defines the PIN to be 4 digits.

```
   PIN Block = PPPPRRRRRRRRRRRR
```

ECI Format 3

```
   PIN Block = LPPPPzzRRRRRRRRR
where L = X'4' to X'6'
```

PIN extraction rules

This topic describes the PIN extraction rules for the Encrypted PIN verify and Encrypted PIN translate callable services.

Encrypted PIN Verify Callable Service

The service extracts the customer-entered PIN from the input PIN block according to the following rules:

- If the input PIN block format is ANSI X9.8, ISO format 0, VISA format 1, VISA format 4, ECI format 1, ISO format 1, ISO format 2, ISO format 4, VISA format 2, IBM Encrypting PINPAD format, or ECI format 3, the service extracts the PIN according to the length specified in the PIN block.
- If the input PIN block format is VISA format 3, the specified delimiter (padding) determines the PIN length. The search starts at the leftmost digit in the PIN block. If the input PIN block format is 3624, the specification of a PIN extraction method for the 3624 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.

- If the input PIN block format is 3621, the specification of a PIN extraction method for the 3621 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.
- If the input PIN block format is ECI format 2, the PIN is the leftmost 4 digits.

For the VISA algorithm, if the extracted PIN length is less than 4, the services sets a reason code that indicates that verification failed. If the length is greater than or equal to 4, the service uses the leftmost 4 digits as the referenced PIN.

For the IBM German Banking Pool algorithm, if the extracted PIN length is not 4, the service sets a reason code that indicates that verification failed.

For the IBM 3624 algorithm, if the extracted PIN length is less than the PIN check length, the service sets a reason code that indicates that verification failed.

Clear PIN Generate Alternate Callable Service

The service extracts the customer-entered PIN from the input PIN block according to the following rules:

- This service supports the specification of a PIN extraction method for the 3624 and 3621 PIN block formats through the use of the rule_array keyword. *Rule_array* points to an array of one or two 8-byte elements. The first element in the rule array specifies the PIN calculation method. The second element in the rule array (if specified) indicates the PIN extraction method. Refer to the "Clear PIN Generate Alternate (CSNBCPA and CSNECPA)" on page 634 for an explanation of PIN extraction method keywords.

Encrypted PIN Translate and Encrypted PIN Translate2 Callable Services

The service extracts the customer-entered PIN from the input PIN block according to the following rules:

- If the input PIN block format is ANSI X9.8, ISO format 0, VISA format 1, VISA format 4, ECI format 1, ISO format 1, ISO format 2, ISO format 4, VISA format 2, IBM Encrypting PINPAD format, or ECI format 3, and if the specified PIN length is less than 4, the service sets a reason code to reject the operation. If the specified PIN length is greater than 12, the operation proceeds to normal completion with unpredictable contents in the output PIN block. Otherwise, the service extracts the PIN according to the specified length.
- If the input PIN block format is VISA format 3, the specified delimiter (padding) determines the PIN length. The search starts at the leftmost digit in the PIN block. If the input PIN block format is 3624, the specification of a PIN extraction method for the 3624 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.
- If the input PIN block format is 3621, the specification of a PIN extraction method for the 3621 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.
- If the input block format is ECI format 2, the PIN is always the leftmost 4 digits.

If the maximum PIN length allowed by the output PIN block is shorter than the extracted PIN, only the leftmost digits of the extracted PIN that form the allowable maximum length are placed in the output PIN block. The PIN length field in the output PIN block, it if exists, specifies the allowable maximum length.

PIN Change/Unblock Callable Service

The PIN Block calculation PIN Change/Unblock:

1. Form three 8-byte, 16-digit blocks, -1, -2, and -3, and set all digits to X'0'
2. Replace the rightmost four bytes of block-1 with the authentication code described in the previous section.
3. Set the second digit of block-2 to the length of the new PIN (4 to 12), followed by the new PIN, and padded to the right with X'F'

4. Include any current PIN by placing it into the leftmost digits of block-3.
5. Exclusive-OR blocks -1, -2, and -3 to form the 8-byte PIN block.
6. Pad the PIN block with other portions of the message for the smart card:
   - Prepend X'08'
   - Append X'80'
   - Append an additional six bytes of X'00'

The resulting message is ECB-mode triple-encrypted with an appropriate session key.

IBM PIN algorithms

This topic describes the IBM PIN Generation algorithms, IBM PIN Offset Generation algorithm, and IBM PIN Verification algorithms.

3624 PIN Generation algorithm

This algorithm generates a n-digit PIN based on an account-related data or person-related data, namely the validation data. The assigned PIN length parameter specifies the length of the generated PIN.

The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 4-bit assigned PIN length
- A 128-bit PIN-generation key

The service uses the PIN generation key to encipher the validation data. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of the enciphered validation data. The result is an intermediate PIN. The leftmost n digits of the intermediate PIN are the generated PIN, where n is specified by the assigned PIN length.

Figure 38 on page 1636 illustrates the 3624 PIN generation algorithm.

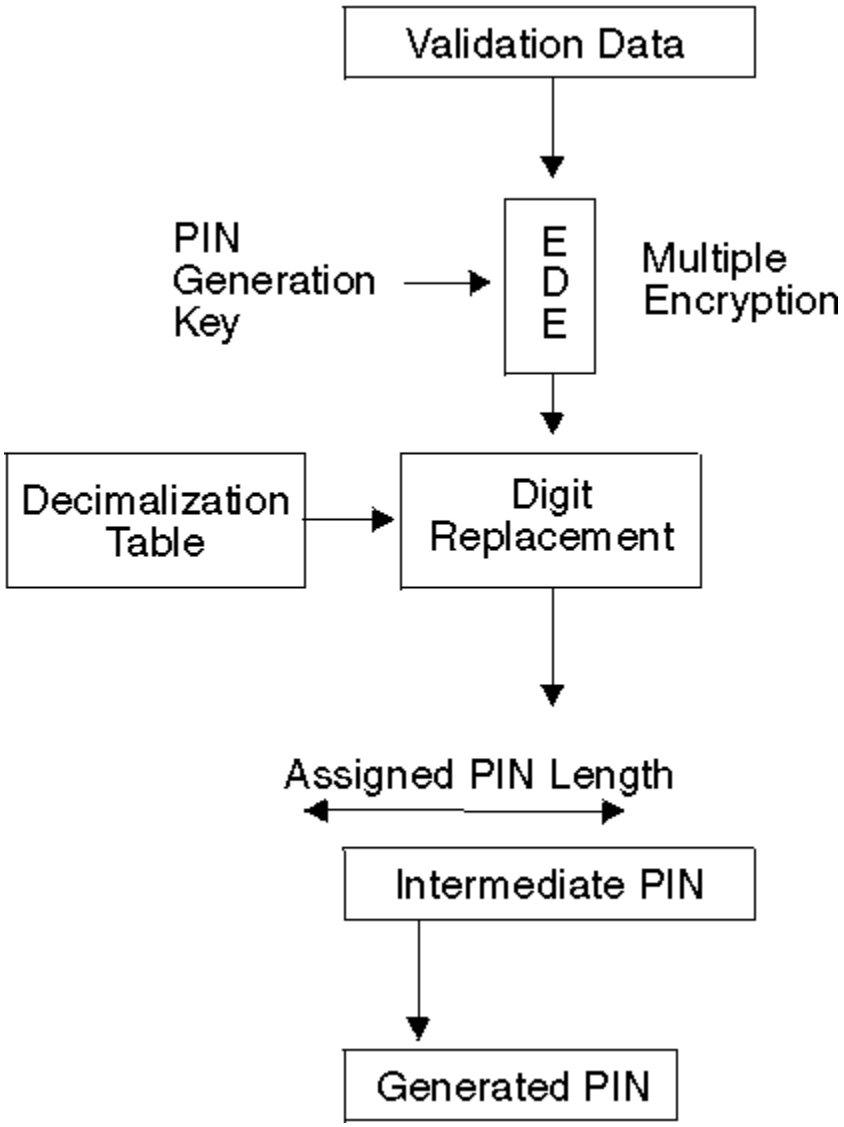


Figure 38. 3624 PIN Generation Algorithm

German Banking Pool PIN Generation algorithm

This algorithm generates a 4-digit PIN based on an account-related data or person-related data, namely the validation data.

The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 128-bit PIN-generation key

The validation data is enciphered using the PIN generation key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data. The result is an intermediate PIN. The rightmost 4 digits of the leftmost 6 digits of the intermediate PIN are extracted. The leftmost digit of the extracted 4 digits is checked for zero. If the digit is zero, the digit is changed to one; otherwise, the digit remains unchanged. The resulting four digits is the generated PIN.

Figure 39 on page 1637 illustrates the German Banking Pool (GBP) PIN generation algorithm.

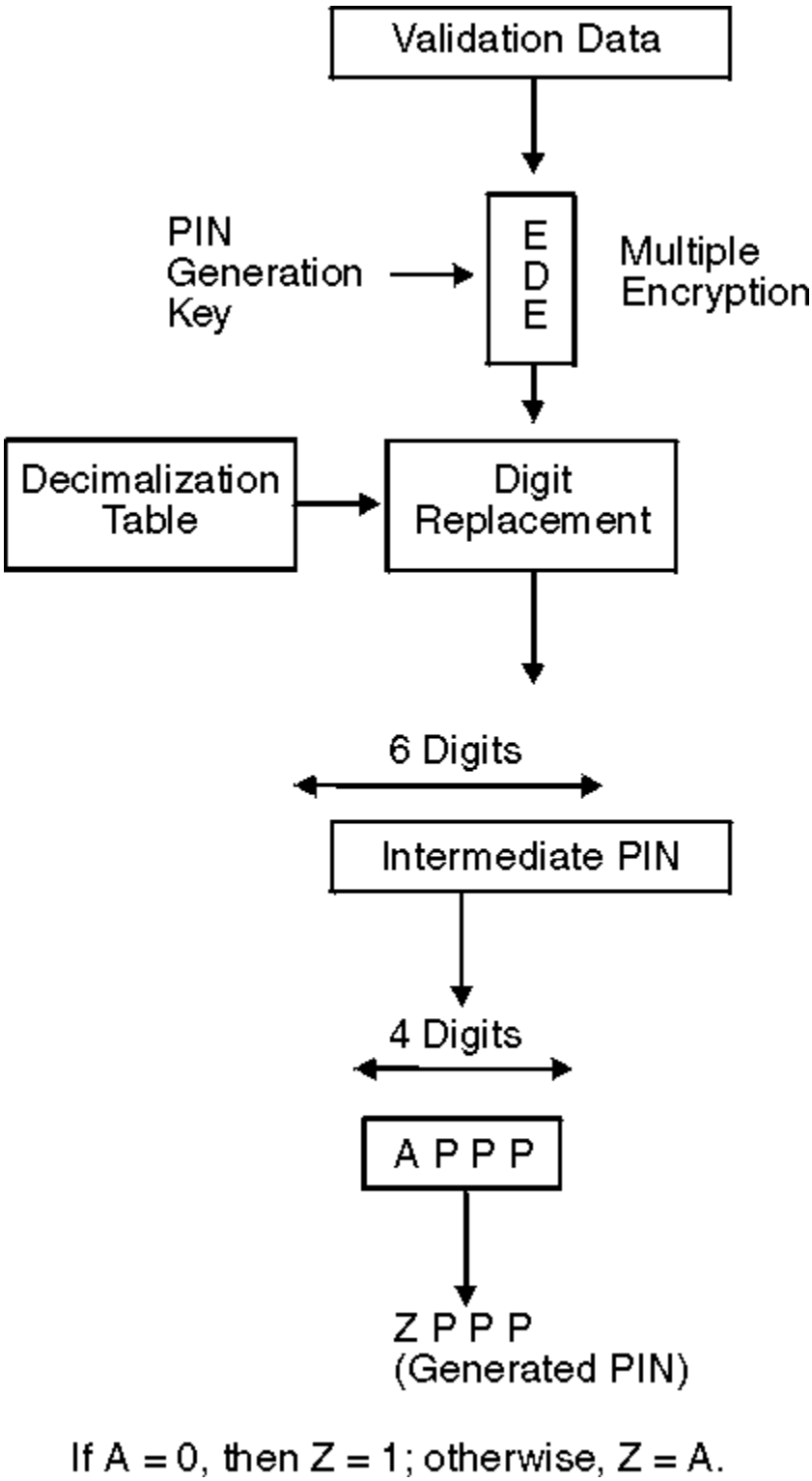


Figure 39. GBP PIN Generation Algorithm

PIN Offset Generation algorithm

To allow the customer to select his own PIN, a PIN offset is used by the IBM 3624 PIN generation algorithm to relate the customer-selected PIN to the generated PIN.

The PIN offset generation algorithm requires two parameters in addition to those used in the 3624 PIN generation algorithm. They are a customer-selected PIN and a 4-bit PIN check length. The length of the customer-selected PIN is equal to the assigned-PIN length, n.

The 3624 PIN generation algorithm described in the previous section is performed. The offset data value is the result of subtracting (modulo 10) the leftmost n digits of the intermediate PIN from the customer-selected PIN. The modulo 10 subtraction ignores borrows. The rightmost m digits of the offset data form the PIN offset, where m is specified by the PIN check length. Note that n cannot be less than m.

Figure 40 on page 1638 illustrates the PIN offset generation algorithm.

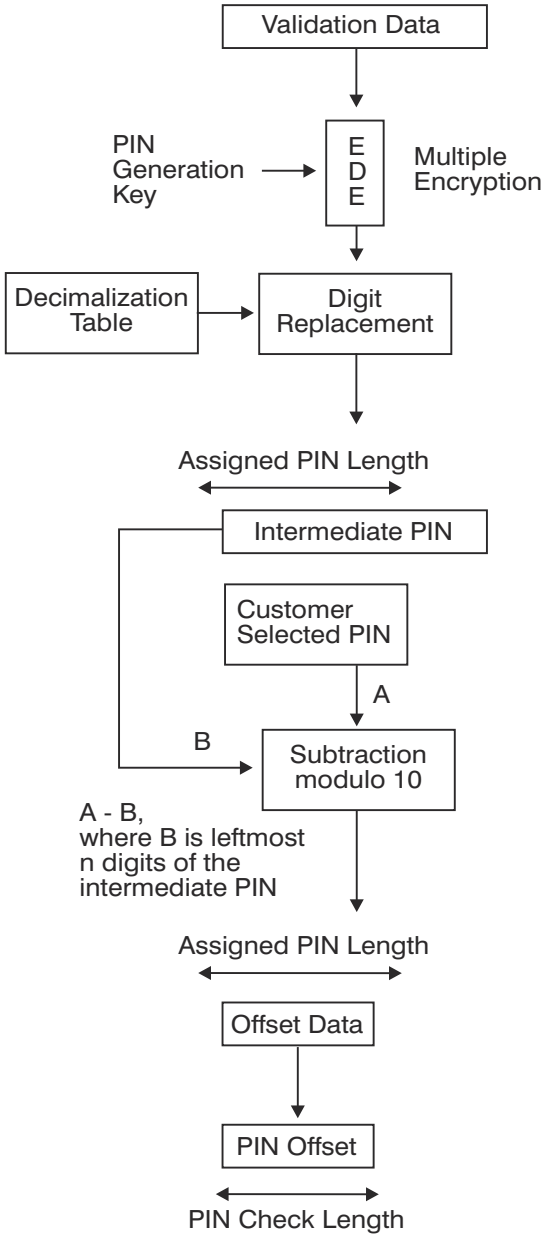


Figure 40. PIN-Offset Generation Algorithm

3624 PIN Verification algorithm

This algorithm generates an intermediate PIN based on the specified validation data. A part of the intermediate PIN is adjusted by adding an offset data. A part of the result is compared with the corresponding part of the customer-entered PIN.

The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 128-bit PIN-verification key

- A 4-bit PIN check length
- An offset data
- A customer-entered PIN

The rightmost m digits of the offset data form the PIN offset, where m is the PIN check length.

1. The validation data is enciphered using the PIN verification key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data.
2. The leftmost n digits of the result is added (modulo 10) to the offset data value, where n is the length of the customer-entered PIN. The modulo 10 addition ignores carries.
3. The rightmost m digits of the result of the addition operation form the PIN check number. The PIN check number is compared with the rightmost m digits of the customer-entered PIN. If they match, PIN verification is successful; otherwise, verification is unsuccessful.

When a nonzero PIN offset is used, the length of the customer-entered PIN is equal to the assigned PIN length.

Figure 41 on page 1640 illustrates the PIN verification algorithm.

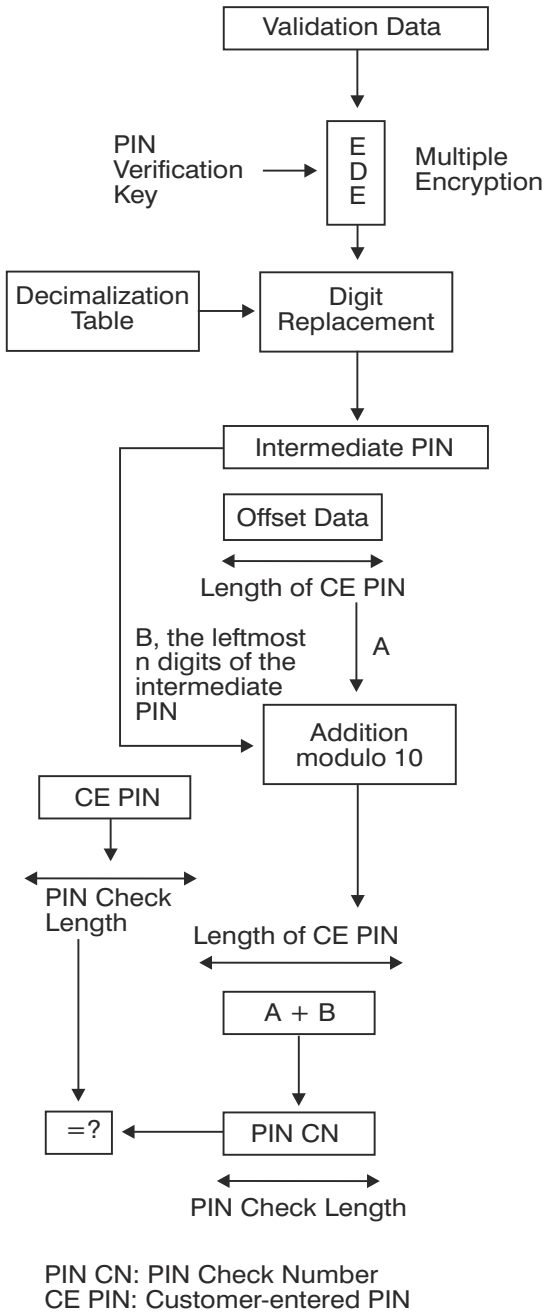


Figure 41. PIN Verification Algorithm

German Banking Pool PIN Verification algorithm

This algorithm generates an intermediate PIN based on the specified validation data. A part of the intermediate PIN is adjusted by adding an offset data. A part of the result is extracted. The extracted value may or may not be modified before it compares with the customer-entered PIN.

The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 128-bit PIN verification key
- An offset data
- A customer-entered PIN

The rightmost 4 digits of the offset data form the PIN offset.

1. The validation data is enciphered using the PIN verification key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data.
2. The leftmost 6 digits of the result is added (modulo 10) to the offset data. The modulo 10 addition ignores carries.
3. The rightmost 4 digits of the result of the addition (modulo 10) are extracted.
4. The leftmost digit of the extracted value is checked for zero. If the digit is zero, the digit is set to one; otherwise, the digit remains unchanged. The resulting four digits are compared with the customer-entered PIN. If they match, PIN verification is successful; otherwise, verification is unsuccessful.

Figure 42 on page 1641 illustrates the GBP PIN verification algorithm.

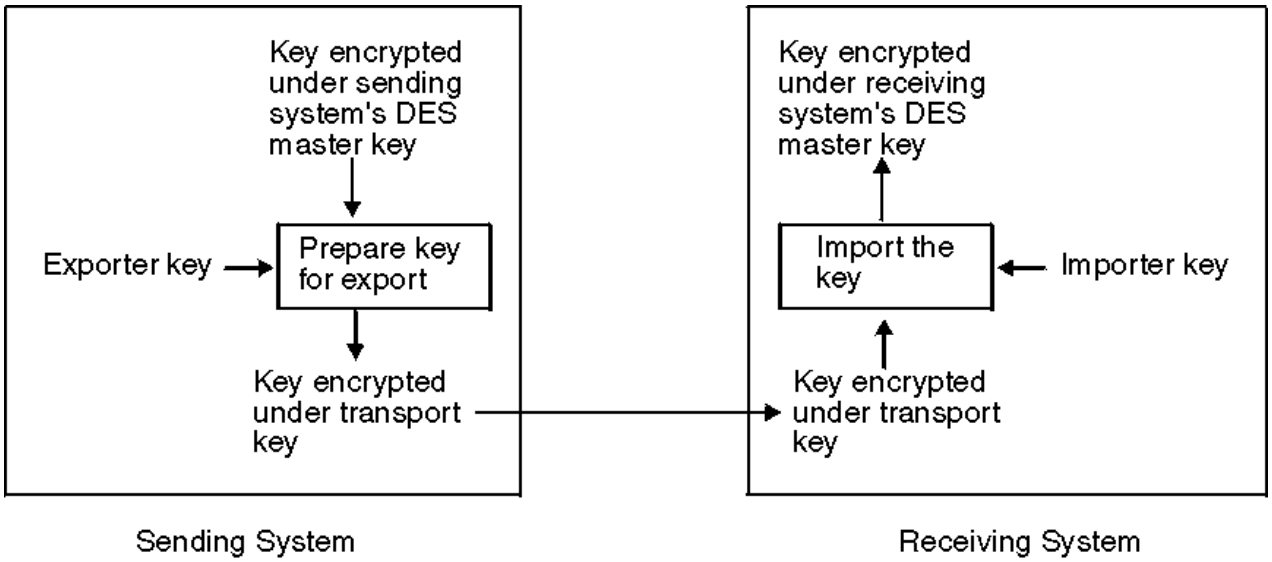


Figure 42. GBP PIN Verification Algorithm

VISA PIN algorithms

The VISA PIN verification algorithm performs a multiple encipherment of a value, called the transformed security parameter (TSP), and a extraction of a 4-digit PIN verification value (PVV) from the ciphertext. The calculated PVV is compared with the referenced PVV and stored on the plastic card or database. If they match, verification is successful.

PVV Generation algorithm

The algorithm generates a 4-digit PIN verification value (PVV) based on the transformed security parameter (TSP).

The algorithm requires the following input parameters:

- A 64-bit TSP
- A 128-bit PVV generation key

1. A multiple encipherment of the TSP using the double-length PVV generation key is performed.
2. The ciphertext is scanned from left to right. Decimal digits are selected during the scan until four decimal digits are found. Each selected digit is placed from left to right according to the order of selection. If four decimal digits are found, those digits are the PVV.
3. If, at the end of the first scan, less than four decimal digits have been selected, a second scan is performed from left to right. During the second scan, all decimal digits are skipped and only nondecimal digits can be processed. Nondecimal digits are converted to decimal digits by subtracting 10. The process proceeds until four digits of PVV are found.

Figure 43 on page 1642 illustrates the PVV generation algorithm.

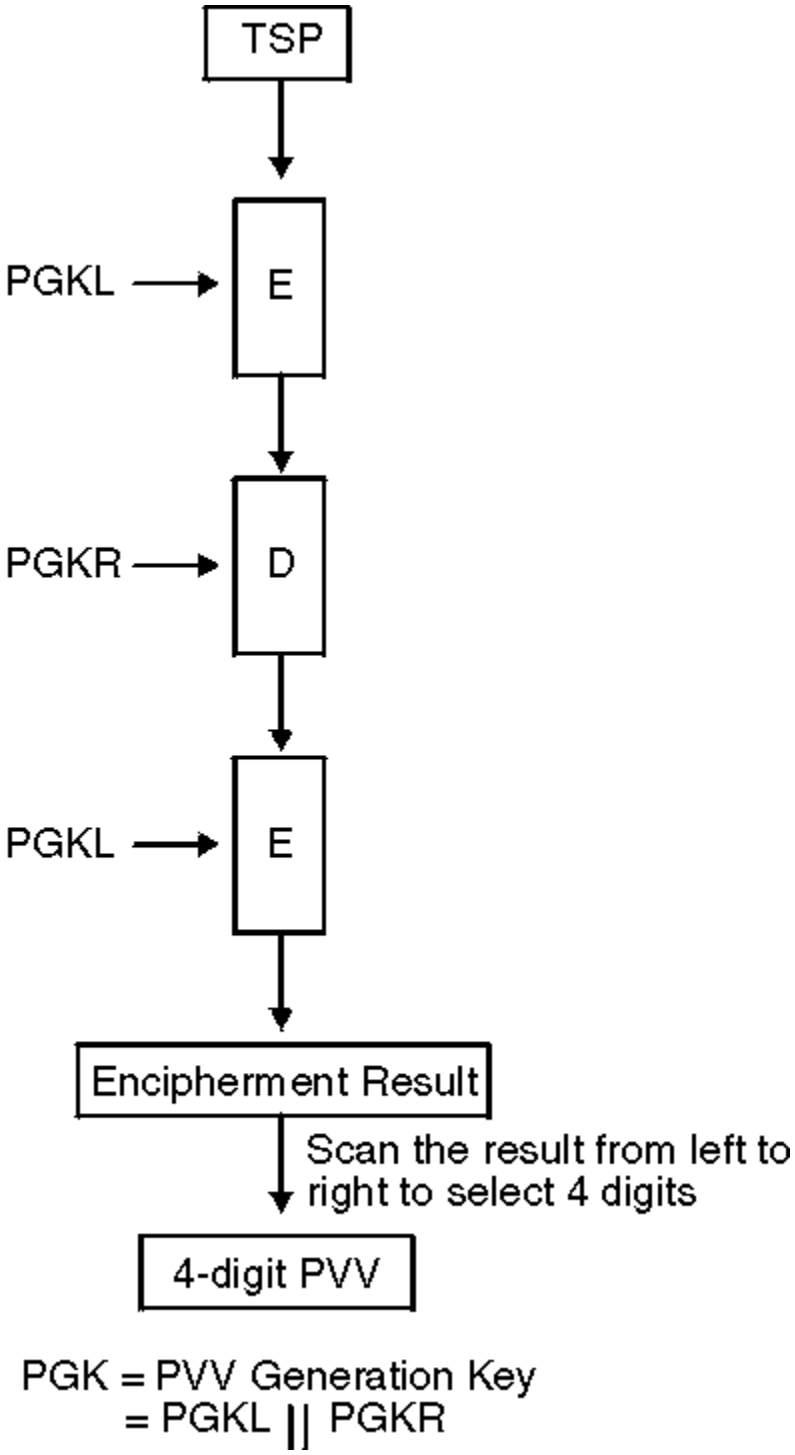


Figure 43. PVV Generation Algorithm

Programming Note: For VISA PVV algorithms, the leftmost 11 digits of the TSP are the personal account number (PAN), the leftmost 12th digit is a key table index to select the PVV generation key, and the rightmost 4 digits are the PIN. The key table index should have a value between 1 and 6, inclusive.

PVV Verification algorithm

The algorithm requires the following input parameters:

- A 64-bit TSP

- A 16-bit referenced PVV
- A 128-bit PVV verification key

A PVV is generated using the PVV generation algorithm, except a PVV verification key rather than a PVV generation key is used. The generated PVV is compared with the referenced PVV. If they match, verification is successful.

Interbank PIN Generation algorithm

The Interbank PIN calculation method consists of the following steps:

1. Let X denote the transaction_security parameter element converted to an array of 16 4-bit numeric values. This parameter consists of (in the following sequence) the 11 rightmost digits of the customer PAN (excluding the check digit), a constant of 6, a 1-digit key indicator, and a 3-digit validation field.
2. Encrypt X with the double-length PINGEN (or PINVER) key to get 16 hexadecimal digits (64 bits).
3. Perform decimalization on the result of the previous step by scanning the 16 hexadecimal digits from left to right, skipping any digit greater than X'9' until 4 decimal digits (for example, digits that have values from X'0' to X'9') are found.

   If all digits are scanned but 4 decimal digits are not found, repeat the scanning process, skipping all digits that are X'9' or less and selecting the digits that are greater than X'9'. Subtract 10 (X'A') from each digit selected in this scan.

   If the 4 digits that were found are all zeros, replace the 4 digits with 0100.
4. Concatenate and use the resulting digits for the Interbank PIN. The 4-digit PIN consists of the decimal digits in the sequence in which they are found.

Cipher processing rules

DES defines operations on 8-byte data strings. Although the fundamental concepts of ciphering (enciphering and deciphering) and data verification are simple, there are different approaches to processing data strings that are not a multiple of 8 bytes in length. These approaches are defined in various standards and IBM products.

CBC and ANSI INCITS 106

ANSI INCITS 106 defines four methods of operation for ciphering. One of these modes, cipher block chaining (CBC), defines the basic method for performing ciphering on multiple blocks. A plaintext data string, which must be a multiple of the block size, is processed as a series of blocks. The ciphered result from processing a block is exclusive ORed with the next block. The last block of the ciphered result is defined as an output chaining vector (OCV). ICSF stores the output chaining vector value in the *chaining_vector* parameter.

An initial chaining vector is exclusive ORed with the first group of 8 input bytes.

In summary:

- An input chaining vector (ICV) is required.
- If the *text_length* is not an exact multiple of 8 bytes, the request fails.
- The plaintext is not padded, for example, the output text length is not increased.

ICSF provides an enhancement to CBC mode called ciphertext-stealing. This allows for a text length that is not a multiple of the block size. This is accomplished by manipulating the last two blocks in a certain way. The second to last block is encrypted in the normal manner, but then some of the bits are "stolen" and added to the last (partial) block. These bits can be recovered by decrypting the last block. This enhancement is currently proposed to NIST as *Proposal To Extend CBC Mode By "Ciphertext Stealing"*, dated May 6, 2007.

ANSI X9.23 and IBM 4700

An enhancement to the basic cipher block chaining mode of ANSI INCITS 106 is defined so the data lengths that are not an exact multiple of 8 bytes can be processed. The ANSI X9.23 method *always* adds from 1 byte to 8 bytes to the plaintext before encipherment. The last added byte is the count of the added bytes and is in the range of X'01' to X'08'. The standard defines that the other added bytes, the pad characters, are random.

When ICSF enciphers the plaintext, the resulting ciphertext is always 1 to 8 bytes longer than the plaintext.

When ICSF deciphers the ciphertext, ICSF uses the last byte of the deciphered data as the number of bytes to be removed (the pad bytes and the count byte). The resulting plaintext is the same as the original plaintext.

The output chaining vector can be used as feedback with this method in the same way as with the INCITS 106 method.

In summary, for the ANSI X9.23 method:

- X9.23 processing requires the caller to supply an ICV.
- X9.23 encipher does not allow specification of a pad character.

The 4700 padding rule is similar to the X9.23 rule. The only difference is that in the X9.23 method, the padding character is not user-selected, but the padding string is selected by the encipher process.

Segmenting

The callable services can operate on large data objects. *Segmenting* is the process of dividing the function into more than one processing step. Your application can divide the process into multiple steps without changing the final outcome.

To provide segmenting capability, the MAC generation, MAC verification, and MDC generation callable services require an 18-byte system work area in the application address space that is provided as the chaining vector parameter to the callable service. The application program must not change the system work area.

Cipher last-block rules

The DES defines cipher-block chaining as operating on multiples of 8 bytes, and AES uses multiples of 16 bytes. Various algorithms are used to process strings that are multiples of the block size. The algorithms are generically named "last-block rules". You select the supported last-block rules by using these keywords:

- X9.23
- IPS
- CUSP (also used with PCF)
- 4700-PAD
- CBC-CS

You specify which cipher last-block rule you want to use in the *rule_array* parameter of the callable service.

CUSP

If the length of the data to be enciphered is an exact multiple of 8 bytes, the ICV is exclusive ORed with the first 8-byte block of plaintext, and the resulting 8 bytes are passed to the DES with the specified key. The resulting 8-byte block of ciphertext is then exclusive ORed with the second 8-byte block of plaintext, and the value is enciphered. This process continues until the last 8-byte block of plaintext is to

be enciphered. Because the length of this last block is exactly 8 bytes, the last block is processed in an identical manner to all the preceding blocks.

To produce the OCV, the last block of *ciphertext* is enciphered again (thus producing a double-enciphered block). The user can pass this value of the OCV as the ICV in his next encipher call to produce chaining between successive calls. The caller can alternatively pass the same ICV on every call to the callable service.

If the length of data to be enciphered is greater than 7 bytes, and is *not* an exact multiple of 8 bytes, the process is the same until the last partial block of 1 to 7 bytes is reached. To encipher the last short block, the previous 8-byte block of ciphertext is passed to the DES with the specified key. The first 1 to 7 bytes of this double-enciphered block has two uses. The first use is to exclusive OR this block with the last short block of plaintext to form the last short block of the ciphertext. The second use is to pass it back as the OCV. Thus, the OCV is the last complete 8-byte block of plaintext, doubly enciphered.

If the length of the data to be enciphered is less than 8 bytes, the ICV is enciphered under the specified key. The first 1 to 7 bytes of the enciphered ICV is exclusive ORed with the plaintext to form the ciphertext. The OCV is the enciphered ICV.

The Information Protection System (IPS)

The Information Protection System (IPS) offers two forms of chaining: block and record. Under record chaining, the OCV for each enciphered data string becomes the ICV for the next. Under block chaining, the same ICV is used for each encipherment.

Files that are enciphered directly with the ICSF encipher callable service cannot be properly deciphered using the IPS/CMS CIPHER command or the IPS/CMS subroutines. Both IPS/CMS CIPHER and AMS REPRO ENCIPHER write headers to their files that contain information (principally the ICV and chaining method) needed for decipherment. The encipher callable service does not generate these headers. Specialized techniques are described in IPS/CMS documentation to overcome some, if not all, of these limitations, depending on the chaining mode. As a rough test, you can attempt a decipherment with the CIPHER command HDWARN option, which causes CIPHER to continue processing even though the header is absent.

The encipher callable service returns an OCV used by IPS for record chaining. This allows cryptographic applications using ICSF to be compatible with IPS record chaining.

Record chaining provides a superior method of handling successive short blocks, and has better error recovery features when the caller passes successive short blocks.

The principle used by record chaining is that *the OCV is the last 8 bytes of ciphertext*. This is handled as follows:

- If the length of the data to be enciphered is an exact multiple of 8 bytes, the ICV is exclusive ORed with the first 8 byte block of plaintext, and the resulting 8 bytes are passed to the DES with the specified key. The resulting 8-byte block of ciphertext is then exclusive ORed with the second 8-byte block of plaintext, and the resulting value is enciphered. This process continues until the last 8-byte block of plaintext is to be enciphered. Because the length of this last block is exactly 8 bytes, the last block is processed in an identical manner to all the preceding blocks.

  The OCV is the last 8 bytes of ciphertext.

  The user can pass this value as the ICV in the next encipher call to produce chaining between successive calls.
- If the length of data to be enciphered is greater than 7 bytes, and is *not* an exact multiple of 8 bytes, the process is the same until the last partial block of 1 to 7 bytes is reached. To encipher the last short block, the previous 8-byte block of ciphertext is passed to the DES with the specified key. The first 1 to 7 bytes of this doubly enciphered block is then exclusive ORed with the last short block of plaintext to form the last short block of the ciphertext. The OCV is the last 8 bytes of ciphertext.
- If the length of the data to be enciphered is less than 8 bytes, then the ICV is enciphered under the specified key. The first 1 to 7 bytes of the enciphered ICV is exclusive ORed with the plaintext to form

the ciphertext. The OCV is the rightmost 8 bytes of the plaintext ICV concatenated with the short block of ciphertext. For example:

```
ICV         = ABCDEFGH
ciphertext  = XYZ
OCV         = DEFGHXYZ
```

PKCS padding method

This topic describes the algorithm used to pad clear text when the PKCS-PAD method is specified. Padding is applied before encryption when this keyword is specified with the Symmetric Algorithm Encipher callable service, and it is removed from decrypted data when the keyword is specified with the Symmetric Algorithm Decipher callable service.

The rules for PKCS padding are very simple:

- Padding bytes are always added to the clear text before it is encrypted.
- Each padding byte has a value equal to the total number of padding bytes that are added. For example, if 6 padding bytes must be added, each of those bytes will have the value 0x06.
- The total number of padding bytes is at least one, and is the number that is required in order to bring the data length up to a multiple of the cipher algorithm block size.

The callable services described in this document use AES, which has a cipher block size of 16 bytes. The total number of padding bytes added to the clear text will always be between 1 and 16. Table 672 on page 1646 indicates exactly how many padding bytes are added according to the data length and also shows the value of the padding bytes that are applied.

Table 672. Padding bytes added according to the data length

Value of clear text length (mod 16)	Number of padding bytes added	Value of each padding byte
0	16	0x10
1	15	0x0F
2	14	0x0E
3	13	0x0D
4	12	0x0C
5	11	0x0B
6	10	0x0A
7	9	0x09
8	8	0x08
9	7	0x07
10	6	0x06
11	5	0x05
12	4	0x04
13	3	0x03
14	2	0x02
15	1	0x01

Note that the PKCS standards that define this padding method describe it in a way that limits the maximum padding length to 8 bytes. This is a consequence of the fact that the algorithms at that time used 8-byte blocks. We extend the definition to apply to 16-byte AES cipher blocks.

PKCS padding method (Example 1)

Clear text consists of the following18 bytes:

```
F14ADBDA019D6DB7 EFD91546E3FF8444 9BCB
```

In order to make this a multiple of 16 bytes (the AES block size), we must add 14 bytes. Each byte will contain the value 0x0E, which is 14, the total number of padding bytes added. The result is that the padded clear text is as follows:

```
F14ADBDA019D6DB7 EFD91546E3FF8444 9BCB0E0E0E0E0E0E
0E0E0E0E0E0E0E0E
```

The padded value is 32 bytes in length, which is two AES blocks. This padded string is encrypted in CBC mode, and the resulting ciphertext will also be 32 bytes in length.

PKCS padding method (Example 2)

Clear text consists of the following16 bytes:

```
971ACD01C9C7ADEA CC83257926F490FF
```

This is already a multiple of the AES block size, but PKCS padding rules say that padding is always applied. Thus, we add 16 bytes of padding to bring the total length to 32, the next multiple of the AES block size. Each pad byte has the value 0x10, which is 16, the total number of padding bytes added. The result is that the padded clear text is as follows:

```
971ACD01C9C7ADEA CC83257926F490FF 1010101010101010
1010101010101010
```

The padded value is 32 bytes in length, which is two AES blocks. This padded string is encrypted in CBC mode, and the resulting cipher text will also be 32 bytes in length.

Wrapping methods for symmetric key tokens

This topic explains how symmetric keys are wrapped with master and key-encrypting keys. The fixed-length, 64-byte key tokens use one of several methods. The variable-length key tokens with associated data uses the payload wrapping method.

ECB wrapping of DES keys in a fixed-length token (WRAP-ECB)

The wrapping of a double-length key (*K) using a double-length *KEK is defined as follows:

```
e*KEK(KL) || e*KEK(KR) = eKEKL(dKEKR(eKEKL(KL))) || eKEKL(dKEKR(eKEKL(KR)))
```

Where:

- KL is the left 64 bits of *K.
- KR is the right 64 bits of *K.
- KEKL is the left 64 bits of *KEK.
- KEKR is the right 64 bits of *KEK.
- || means concatenation

CBC wrapping of AES keys in fixed-length tokens

The key value in AES tokens are wrapped using the AES algorithm and cipher block chaining (CBC) mode of encryption. The key value is left justified in a 32-byte block, padded on the right with zero and encrypted.

The enhanced wrapping of an AES key (*K) using an AES *MK is defined as follows: e*MK(*K) = ecbcMK(*K)

Enhanced CBC wrapping of DES keys in fixed-length tokens (WRAP-ENH, WRAPENH2, WRAPENH3)

The enhanced CBC wrapping method uses triple DES encryption, an internal chaining of the key value, and CBC mode.

WRAP-ENH processing

1. Wrapping Key Derivation: Derive the key that will do the encryption step.
2. Wrapping Key Variant Creation: The CV of the key being wrapped is XORed to the wrapping key.
3. Chaining of Key Data: Hashes of key sections are applied to bind the sections together.
4. CBC Encryption of the Key: The wrapping key variant is used to CBC encrypt the chained key.

WRAPENH2 processing

1. Wrapping Key Derivation: Same as WRAP-ENH.
2. Wrapping Key Variant Creation: Same as WRAP-ENH.
3. Chaining of Key Data: Use SHA-256 as the hash function.
4. CBC Encryption of the Key: Same as WRAP-ENH.

WRAPENH3 processing

1. Wrapping Key Derivation: Same as WRAP-ENH with different derivation label.
2. Chaining of Key Data: Use SHA-256 as the hash function.
3. TDES-CMAC Key Derivation: Derive the key that will do the TDES-CMAC calculation.
4. TDES-CMAC Calculation: Calculate TDES-CMAC over full key block.

Wrapping key derivation follows NIST standard SP 800-108, 'Recommendation for Dey Derivation Using Pseudorandom Functions' (October 2009). Derivation will use the method 'KDF in Counter Mode”' using pseudorandom function (PRF) HMAC-SHA256.

For the WRAPENH3 method, an authentication code is generated for the key token. The authentication code is generated using the TDES-CMAC algorithm and stored in the key token where the right control vector was stored (offset 40). The MAC key is derived from the master key in the same manner as the wrapping key.

Variable length token (AESKW method)

The wrapping method for the variable-length key tokens will be AESKW as defined in ANSI X9.102.

The wrapping of the payload of a variable length key (*K) using an AES *MK is defined as follows:

```
e*MK(*K) = eAESKW*MK(P)

P = ICV || Pad Length || Hash Length || Hash options || Data Hash || *K || Padding
```

Where:

- ICV is the 6 byte constant 0xA6A6A6A6A6A6.
- Pad length is the length of the Padding in bits.
- Hash length is the length of the Data Hash in bytes.

- Hash options is a 4-byte field.
- Data Hash is the hash of the associated data block.
- Padding is the number of bytes, 0x00, to make the overall length of P a multiple of 16.
- eAESKW means encryption using the AESKW method.

PKA92 key format and encryption process

The PKA Symmetric Key Generate and the PKA Symmetric Key Import callable services optionally support a PKA92 method of encrypting a DES key with an RSA public key. This format is adapted from the IBM Transaction Security System (TSS) 4753 and 4755 product's implementation of "PKA92". The callable services do not create or accept the complete PKA92 AS key token as defined for the TSS products. Rather, the callable services only support the actual RSA-encrypted portion of a TSS PKA92 key token, the *AS External Key Block*.

Forming an AS External Key Block: The PKA96 implementation forms an AS External Key Block by RSA-encrypting a key block using a public key. The key block is formed by padding the key record detailed in Table 673 on page 1649 with zero bits on the left, high-order end of the key record. The process completes the key block with three sub-processes: masking, overwriting, and RSA encrypting.

Table 673. PKA96 Clear DES Key Record

Offset (Bytes)	Length (Bytes)	Description
Zero-bit padding to form a structure as long as the length of the public key modulus. The implementation constrains the public key modulus to a multiple of 64 bits in the range of 512 to 1024 bits. Note that government export or import regulations can impose limits on the modulus length. The maximum length is validated by a check against a value in the Function Control Vector.		
000	005	Header and flags: X'01 0000 0000'
005	016	Environment Identifier (EID), encoded in ASCII
021	008	Control vector base for the DES key
029	008	Repeat of the CV data at offset 021
037	008	The single-length DES key or the left half of a double-length DES key
045	008	The right half of a double-length DES key or a random number. This value is locally designated "K."
053	008	Random number, "IV"
061	001	Ending byte, X'00'

Masking Sub-process:

1. Form the initial key block by padding the PKR with zero bits on the left, high-order end to the length of the modulus.
2. Create a mask by CBC encrypting a multiple of 8 bytes of binary zeros using K as the key and the length of the modulus, and IV as the initialization vector as defined in the key record at offsets 45 and 53. Exclusive-OR the mask with the key record and call the result PKR.
3. Exclusive-OR the mask with the key block.

Overwriting Sub-process:

1. Set the high-order bits of PKR to B'01', and set the low-order bits to B'0110'.
2. Exclusive-OR K and IV and write the result at offset 45 in PKR.
3. Write IV at offset 53 in PKR. This causes the masked and overwritten PKR to have IV at its original position.

Encrypting Sub-process: RSA encrypt the overwritten PKR masked key record using the public key of the receiving node. This is the last step in creating an AS external key block

Recovering a Key from an AS External Key Block: Recover the encrypted DES key from an AS External Key Block by performing decrypting, validating, unmasking, and extraction sub-processes.

Decrypting Sub-process: RSA decrypt the AS External Key Block using an RSA private key and call the result of the decryption PKR. The private key must be usable for key management purposes.

Validating Sub-process: Verify that the high-order two bits of the decrypted key block are valued to B'01' and that the low-order four bits of the PKR record are valued to B'0110'.

Unmasking Sub-process: Set IV to the value of the 8 bytes at offset 53 of the PKR record. Note that there is a variable quantity of padding prior to offset 0. See Table 673 on page 1649.

Set K to the exclusive-OR of IV and the value of the 8 bytes at offset 45 of the PKR record.

Create a mask that is equal in length to the key block by CBC encrypting a multiple of 8 bytes of binary zeros using K as the key and IV as the initialization vector. Exclusive-OR the mask with PKR and call the result the key record.

Copy K to offset 45 in the PKR record.

Extraction Sub-process: Confirm that:

- The four bytes at offset 1 in the PKR are valued to X'0000 0000'
- The two control vector fields at offsets 21 and 29 are identical
- If the control vector is an IMPORTER or EXPORTER key class, that the EID in the key record is not the same as the EID stored in the cryptographic engine.

The control vector base of the recovered key is the value at offset 21. If the control vector base bits 40 to 42 are valued to B'010' or B'110', the key is double length. Set the right half of the received key's control vector equal to the left half and reverse bits 41 and 42 in the right half.

The recovered key is at offset 37 and is either 8 or 16 bytes long based on the control vector base bits 40 to 42. If these bits are valued to B'000', the key is single length. If these bits are valued to B'010' or B'110', the key is double length.

Formatting hashes and keys in public-key cryptography

The digital signature generate and digital signature verify callable services support several methods for formatting a hash, and in some cases a descriptor for the hashing method, into a bit-string to be processed by the cryptographic algorithm. This topic discusses the ANSI X9.31 and PKCS #1 methods. The ISO 9796-1 method can be found in the ISO standard.

This topic also describes the PKCS #1, version 1, 1.5, and 2.0, methods for placing a key in a bit string for RSA ciphering in a key exchange.

ANSI X9.31 hash format

With ANSI X9.31, the string that is processed by the RSA algorithm is formatted by the concatenation of a header, padding, the hash and a trailer, from the most significant bit to the least significant bit, such that the resulting string is the same length as the modulus of the key. For the ICSF implementation, the modulus length must be a multiple of 8 bits.

- The header consists of X'6B'
- The padding consists of X'BB', repeated as many times as required, and terminated by X'BA'
- The hash value follows the padding
- The trailer consists of a hashing mechanism specifier and final byte. These specifiers are defined:
  - X'31': RIPEMD-160
  - X'33': SHA-1

  - X'34': SHA-256
  - X'35': SHA-512
  - X'36': SHA-384
- A final byte of X'CC'.

PKCS #1 formats

The PKCS #1 standard [4] defines methods for formatting keys and hashes prior to RSA encryption of the resulting data structures. PKCS #1 Version 1.5 defined block types 0, 1, and 2, but in the current standard that terminology is dropped.

ICSF implemented these processes using the terminology of the Version 2.0 (RFC 2437), Version 2.1 (RFC 3447), and Version 2.2 (RFC 8017) PKCS #1 standard:

- For formatting keys for secured transport (CSNDSYX, CSNDSYG, CSNDSYI):
  - RSAES-OAEP, the preferred method for key-encipherment[5] when exchanging DATA keys between systems. Keywords PKCSOAEP (Version 2.0) and PKOAEP2 (Version 2.1) are used to invoke this formatting technique. The P parameter described in the standard is not used and its length is set to zero.
  - RSAES-PKCS1-v1_5, is an older method for formatting keys. Keyword PKCS-1.2 is used to invoke this formatting technique.
- For formatting hashes for digital signatures (CSNDDSG and CSNDDSV):
  - RSASSA-PKCS1-v1_5, the newer name for the block-type 1 format. Keyword PKCS-1.1 is used to invoke this formatting technique.
  - The PKCS #1 specification no longer discusses use of block-type 0. Keyword PKCS-1.0 is used to invoke this formatting technique. Use of block-type 0 is discouraged.
- Version 2.2 of the standard defined a parameter field for RSASSA-PSS (RSA Signature Scheme with Appendix – Probabilistic Signature Scheme) that has the following four parameters:
  - hashAlgorithm - This parameter identifies the hash function. ICSF requires this to be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
  - maskGenAlgorithm – This parameter identifies the mask generation function. MGF1 is a mask generation function based on a hash function and is the only function currently defined by the standard. The hash function of which MGF1 is based is always the same as hashAlgorithm.
  - saltLength – This is the length of the salt, which is a randomly generated value. In no case can the derived salt length be less than the salt length specified by the input data.
  - trailerField – This is the trailer field number. ICSF always sets this to the value X'BC'. Other trailer field numbers are not supported by the standard.

Keyword PKCS-PSS is used to invoke the RSASSA-PSS formatting technique.

Using the terminology from older versions of the PKCS #1 standard, block types 0 and 1 are used to format a hash and block type 2 is used to format a DES key. The blocks consist of (|| means concatenation): X'00' || BT || PS || X'00' D where:

- BT is the block type, X'00', X'01', X'02'.
- PS is the padding of as many bytes as required to make the block the same length as the modulus of the RSA key, and is bytes of X'00' for block type 0, X'01' for block type 1, and random and non-X'00' for block type 2. The length of PS must be at least 8 bytes.
- D is the key, or the concatenation of the BER-encoded hash identifier and the hash.

---

[4] PKCS standards can be retrieved from OASIS Open (www.oasis-open.org).

[5] The PKA 92 method and the method incorporated into the SET standard are other examples of the Optimal Asymmetric Encryption Padding (OAEP) technique. The OAEP technique is attributed to Bellare and Rogaway.

You can create the ASN.1 BER encoding of an MD5, SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512 value by prepending a string to the hash value, as shown:

```
MD5  X’3020300C 06082A86 4886F70D 02050500 0410’ || 16-byte hash value
SHA-1  X'30213009 06052B0E 03021A05 000414’ || 20-byte hash value
SHA-224  X’302D300D 06096086 48016503 04020405 00041C’ || 28-byte hash value
SHA-256  X’3031300D 06096086 48016503 04020105 000420’ || 32-byte hash value
SHA-384  X’3041300D 06096086 48016503 04020205 000430’ || 48-byte hash value
SHA-512  X’3051300D 06096086 48016503 04020305 000440’ || 64-byte hash value
```

Visa, MasterCard, and EMV-related smart card formats and processes

The VISA, MasterCard, and EMV specifications for performing secure messaging with an EMV compliant smart card are covered in these documents:

- *EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2*
- *Design Visa Integrated Circuit Card Specification Manual*
- *Integrated Circuit Card Specification (VIS) 1.4.0 Corrections*
- *MasterCard International: M/Chip 4 Security & Key Management Version 1.0*

M/Chip 4 Security & Key Management describes how a smart-card, card-specific session key is derived from a card-issuer-supplied master key.

Book 2, Annex A1.3, describes how a smart-card, card-specific authentication code is derived from a card-issuer-supplied encryption key (ENC-MDK). The *Integrated Circuit Card Specification (VIS) 1.4.0 Corrections* indicates that the key used should be an authentication key (MAC-MDK).

Book 2, Annex A1.3 describes how a smart-card, card-specific session key is derived from a card-issuer-supplied PIN-block-encryption key (ENC-MDK). The encryption key is derived using a "tree-based-derivation" technique. IBM CCA offers two variations of the tree-based technique (TDESEMV2 and TDESEMV4), and a third technique CCA designates TDES-XOR.

In addition, Book 2 describes construction of the PIN block sent to an EMV card to initialize or update the user's PIN.

Design Visa Integrated Circuit Card Specification Manual, Annex B.4, contains a description of the session-key derivation technique CCA designates TDES-XOR.

Augmented by the previously-mentioned documentation, the relevant processes are described in these sections:

- “Deriving the smart-card-specific authentication code ” on page 1652
- “Constructing the PIN-block for transporting an EMV smart-card PIN ” on page 1652
- “Deriving the CCA TDES-XOR session key ” on page 1653
- “Deriving the EMV TDESEMVn tree-based session key” on page 1653
- “PIN-block self-encryption ” on page 1654

Deriving the smart-card-specific authentication code

To ensure that an original or replacement PIN is received from an authorized source, the EMV PIN-transport PIN-block incorporates an authentication code. The authentication code is the rightmost four bytes resulting from the ECB-mode triple-DES encryption of (the first) eight bytes of card-specific data (that is, the rightmost four bytes of the Unique DEA Key A).

Constructing the PIN-block for transporting an EMV smart-card PIN

The PIN block is used to transport a new PIN value. The PIN block also contains an authentication code, and optionally the "current" PIN value, enabling the smart card to further ensure receipt of a valid PIN

value. To enable incorporation of the PIN block into the a message for an EMV smart-card, the PIN block is padded to 16 bytes prior to encryption.

PINs of length 4 - 12 digits are supported.

PIN-block construction:

1. Form three 8-byte, 16-digit blocks, block-1, block-2, and block-3, and set all digits to X'0'.
2. Replace the rightmost four bytes of block-1 with the authentication code described in the previous section.
3. Set the second digit of block-2 to the length of the new PIN (4 to 12), followed by the new PIN, and padded to the right with X'F'.
4. Include any current PIN by placing it into the leftmost digits of block-3.
5. Exclusive-OR block-1, block-2, and block-3 to form the 8-byte PIN block.
6. Pad the PIN block with other portions of the message for the smart card:
   - Prepend X'08' (the length of the PIN block)
   - Append X'80', followed by 6 bytes of X'00'

The resulting message is ECB-mode triple-encrypted with an appropriate session key.

Deriving the CCA TDES-XOR session key

In the diversified key generate and PIN change/unblock services, the TDES-XOR process first derives a smart-card-specific intermediate key from the issuer-supplied ENC-MDK key and card-specific data. (This intermediate key is also used in the TDESEMV2 and TDESEMV4 processes. See the next section.) The intermediate key is then modified using the application transaction counter (ATC) value supplied by the smart card.

The double-length session-key creation steps:

1. Obtain the left-half of an intermediate key by ECB-mode triple-DES encrypting the (first) eight bytes of card specific data using the issuer-supplied ENC-MDK key.
2. Again using the ENC-MDK key, obtain the right-half of the intermediate key by one of the following ways:
   - When 16 bytes have been supplied, ECB-mode triple-DES encrypting the second 8 bytes of card-specific derivation data.
   - When 16 bytes have not been supplied, ECB-mode triple-DES encrypting the exclusive-OR of the supplied 8 bytes of derivation data with X'FFFFFFFFFFFFFFFF'.
3. Pad the ATC value to the left with six bytes of X'00' and exclusive-OR the result with the left-half of the intermediate key to obtain the left-half of the session key.
4. Obtain the one's complement of the ATC by exclusive-ORing the ATC with X'FFFF'. Pad the result on the left with six bytes of X'00'. Exclusive-OR the 8-byte result with the right-half of the intermediate key to obtain the right-half of the session key.

Deriving the EMV TDESEMVn tree-based session key

In the diversified key generate and PIN change/unblock services, the TDESEMV2 and TDESEMV4 keywords call for the creation of the session key with this process:

1. The intermediate key is obtained as explained previously for the TDES-XOR process.
2. Combine the intermediate key with the two-byte Application Transaction Counter (ATC) and an optional Initial Value. The process is defined in the EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2, Annex A1.3.
   - TDESEMV2 causes processing with a branch factor of 2 and a height of 16.
   - TDESEMV4 causes processing with a branch factor of 4 and a height of 8.

PIN-block self-encryption

In the Secure Messaging for PINs (CSNBSPN and CSNESPN) service, you can use the SELFENC rule-array keyword to specify that the 8-byte PIN block shall be used as a DES key to encrypt the PIN block. The service copies the self-encrypted PIN block to the clear PIN block in the output message.

Key test verification pattern algorithms

The key test verification pattern algorithms are:

- The DES algorithm is used by the Key Test callable service to generate and verify the verification pattern.
- The SHAVP1 algorithm is used by the Key Test2 callable service to generate and verify the verification pattern.
- The SHA-256 algorithm is used by the Key Test and Key Test2 callable services to generate and verify the verification pattern.

DES algorithm (single-length and double-length keys)

For DES keys, the Key Test callable service uses this algorithm to generate and verify the verification pattern.

KK = eC(KL) XOR KL
VP = eKK(KR XOR RN) XOR KR XOR RN

where:

- eK(x) - x is encrypted by key K using the DES algorithm
- KL is the left 64-bit clear key value of the key
- KR is the right 64-bit clear key value of the key (will be hex zero for a single length key)
- C is X'4545454545454545'
- KK is a 64-bit intermediate value
- RN is a 64-bit pseudo-random number
- VP is the 64-bit verification pattern

SHAVP1 algorithm

This algorithm is used by the Key Test2 callable service to generate and verify the verification pattern.

VP = Trunc64(SHA256(KA || KT || KL || K))

Where:

- VP is the 64-bit verification pattern
- TruncN(x) is truncation of the string x to the left most N bits
- SHA256(x) is the SHA-256 hash of the string x
- KA is the one-byte CCA variable-length key token constant for the algorithm of key (HMAC X'03')
- KT is the two-byte CCA variable-length key token constant for the type of key (MAC X'0002')
- KL is the two-byte bit length of the clear key value
- K is the clear key value left justified and padded on the right with binary zeros to byte boundary
- || is string concatenation

SHA-256 algorithm

This algorithm is used by the Key Test and Key Test2 callable services to generate and verify the verification pattern.

VP = Trunc64(SHA256(KA || K))

Where:

- VP is the 64-bit verification pattern.
- TruncN(*x*) is truncation of the string *x* to the left most N bits.
- SHA256(*x*) is the SHA-256 hash of the string *x*.
- KA is the one-byte CCA variable-length key token constant for the algorithm of key (AES X'01').
- K is the clear key value left-justified and padded on the right with binary zeros to byte boundary.
- || is string concatenation.

Appendix F. EBCDIC and ASCII default conversion tables

This topic contains EBCDIC to ASCII and ASCII to EBCDIC conversion tables. In the table headers, EBC refers to EBCDIC and ASC refers to ASCII.

Table 674 on page 1657 shows the EBCDIC to ASCII default conversion table.

Table 674. EBCDIC to ASCII default conversion table

EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC
00	00	20	81	40	20	60	2D	80	F8	A0	C8	C0	7B	E0	5C
01	01	21	82	41	A6	61	2F	81	61	A1	7E	C1	41	E1	E7
02	02	22	1C	42	E1	62	DF	82	62	A2	73	C2	42	E2	53
03	03	23	84	43	80	63	DC	83	63	A3	74	C3	43	E3	54
04	CF	24	86	44	EB	64	9A	84	64	A4	75	C4	44	E4	55
05	09	25	0A	45	90	65	DD	85	65	A5	76	C5	45	E5	56
06	D3	26	17	46	9F	66	DE	86	66	A6	77	C6	46	E6	57
07	7F	27	1B	47	E2	67	98	87	67	A7	78	C7	47	E7	58
08	D4	28	89	48	AB	68	9D	88	68	A8	79	C8	48	E8	59
09	D5	29	91	49	8B	69	AC	89	69	A9	7A	C9	49	E9	5A
0A	C3	2A	92	4A	9B	6A	BA	8A	96	AA	EF	CA	CB	EA	A0
0B	0B	2B	95	4B	2E	6B	2C	8B	A4	AB	C0	CB	CA	EB	85
0C	0C	2C	A2	4C	3C	6C	25	8C	F3	AC	DA	CC	BE	EC	8E
0D	0D	2D	05	4D	28	6D	5F	8D	AF	AD	5B	CD	E8	ED	E9
0E	0E	2E	06	4E	2B	6E	3E	8E	AE	AE	F2	CE	EC	EE	E4
0F	0F	2F	07	4F	7C	6F	3F	8F	C5	AF	F9	CF	ED	EF	D1
10	10	30	E0	50	26	70	D7	90	8C	B0	B5	D0	7D	F0	30
11	11	31	EE	51	A9	71	88	91	6A	B1	B6	D1	4A	F1	31
12	12	32	16	52	AA	72	94	92	6B	B2	FD	D2	4B	F2	32
13	13	33	E5	53	9C	73	B0	93	6C	B3	B7	D3	4C	F3	33
14	C7	34	D0	54	DB	74	B1	94	6D	B4	B8	D4	4D	F4	34
15	B4	35	1E	55	A5	75	B2	95	6E	B5	B9	D5	4E	F5	35
16	08	36	EA	56	99	76	FC	96	6F	B6	E6	D6	4F	F6	36
17	C9	37	04	57	E3	77	D6	97	70	B7	BB	D7	50	F7	37
18	18	38	8A	58	A8	78	FB	98	71	B8	BC	D8	51	F8	38
19	19	39	F6	59	9E	79	60	99	72	B9	BD	D9	52	F9	39
1A	CC	3A	C6	5A	21	7A	3A	9A	97	BA	8D	DA	A1	FA	B3

Table 674. EBCDIC to ASCII default conversion table (continued)															
EBC	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**
1B	CD	3B	C2	5B	24	7B	23	9B	87	BB	D9	DB	AD	FB	F7
1C	83	3C	14	5C	2A	7C	40	9C	CE	BC	BF	DC	F5	FC	F0
1D	1D	3D	15	5D	29	7D	27	9D	93	BD	5D	DD	F4	FD	FA
1E	D2	3E	C1	5E	3B	7E	3D	9E	F1	BE	D8	DE	A3	FE	A7
1F	1F	3F	1A	5F	5E	7F	22	9F	FE	BF	C4	DF	8F	FF	FF

Table 675 on page 1658 shows the ASCII to EBCDIC default conversion table.

Table 675. ASCII to EBCDIC default conversion table															
ASC	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**	**ASC**	**EBC**
00	00	20	40	40	7C	60	79	80	43	A0	EA	C0	AB	E0	30
01	01	21	5A	41	C1	61	81	81	20	A1	DA	C1	3E	E1	42
02	02	22	7F	42	C2	62	82	82	21	A2	2C	C2	3B	E2	47
03	03	23	7B	43	C3	63	83	83	1C	A3	DE	C3	0A	E3	57
04	37	24	5B	44	C4	64	84	84	23	A4	8B	C4	BF	E4	EE
05	2D	25	6C	45	C5	65	85	85	EB	A5	55	C5	8F	E5	33
06	2E	26	50	46	C6	66	86	86	24	A6	41	C6	3A	E6	B6
07	2F	27	7D	47	C7	67	87	87	9B	A7	FE	C7	14	E7	E1
08	16	28	4D	48	C8	68	88	88	71	A8	58	C8	A0	E8	CD
09	05	29	5D	49	C9	69	89	89	28	A9	51	C9	17	E9	ED
0A	25	2A	5C	4A	D1	6A	91	8A	38	AA	52	CA	CB	EA	36
0B	0B	2B	4E	4B	D2	6B	92	8B	49	AB	48	CB	CA	EB	44
0C	0C	2C	6B	4C	D3	6C	93	8C	90	AC	69	CC	1A	EC	CE
0D	0D	2D	60	4D	D4	6D	94	8D	BA	AD	DB	CD	1B	ED	CF
0E	0E	2E	4B	4E	D5	6E	95	8E	EC	AE	8E	CE	9C	EE	31
0F	0F	2F	61	4F	D6	6F	96	8F	DF	AF	8D	CF	04	EF	AA
10	10	30	F0	50	D7	70	97	90	45	B0	73	D0	34	F0	FC
11	11	31	F1	51	D8	71	98	91	29	B1	74	D1	EF	F1	9E
12	12	32	F2	52	D9	72	99	92	2A	B2	75	D2	1E	F2	AE
13	13	33	F3	53	E2	73	A2	93	9D	B3	FA	D3	06	F3	8C
14	3C	34	F4	54	E3	74	A3	94	72	B4	15	D4	08	F4	DD
15	3D	35	F5	55	E4	75	A4	95	2B	B5	B0	D5	09	F5	DC
16	32	36	F6	56	E5	76	A5	96	8A	B6	B1	D6	77	F6	39
17	26	37	F7	57	E6	77	A6	97	9A	B7	B3	D7	70	F7	FB
18	18	38	F8	58	E7	78	A7	98	67	B8	B4	D8	BE	F8	80
19	19	39	F9	59	E8	79	A8	99	56	B9	B5	D9	BB	F9	AF

Table 675. ASCII to EBCDIC default conversion table (continued)

ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC
1A	3F	3A	7A	5A	E9	7A	A9	9A	64	BA	6A	DA	AC	FA	FD
1B	27	3B	5E	5B	AD	7B	C0	9B	4A	BB	B7	DB	54	FB	78
1C	22	3C	4C	5C	E0	7C	4F	9C	53	BC	B8	DC	63	FC	76
1D	1D	3D	7E	5D	BD	7D	D0	9D	68	BD	B9	DD	65	FD	B2
1E	35	3E	6E	5E	5F	7E	A1	9E	59	BE	CC	DE	66	FE	9F
1F	1F	3F	6F	5F	6D	7F	07	9F	46	BF	BC	DF	62	FF	FF

Appendix G. Access control points and callable services

For information about PKCS #11 access control points, see 'PKCS #11 Coprocessor Access Control Points' in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

Access to callable services that are executed on a coprocessor is through access control points in the domain role. To execute services on the coprocessor, access control points must be enabled for each service in the domain role. The access control points available depend on the coprocessor you are using.

A new or a zeroized coprocessor (or domain) comes with an initial set of access control points (ACPs) that are enabled by default. The table of access control points lists the default setting of each access control point.

When a firmware upgrade is applied to an existing cryptographic coprocessor, the upgrade may introduce new ACPs.

- If a TKE workstation has been used to manage a cryptographic coprocessor, the firmware upgrade does not retroactively enable the new ACPs. These ACPs must be enabled via the TKE (or subsequent zeroize) in order to utilize the new support they govern.
- If a TKE workstation has not been used to manage a cryptographic coprocessor, the firmware upgrade retroactively updates the new ACPs that would be enabled by default.

Note: Access control points for ICSF utilities are listed in *z/OS Cryptographic Services ICSF Administrator's Guide*.

If an access control point is disabled, the corresponding ICSF callable service will fail during execution with an access denied error.

The following tables list usage information using the following abbreviations:

AE
Always enabled, cannot be disabled.

ED
Enabled by default.

DD
Disabled by default.

SC
Usage of this access control point requires special consideration.

This table lists access control points that affect multiple services or require special consideration when enabling the access control point. The Offset is the hexadecimal offset, or access-control-point code, for the control in the domain role in the coprocessor.

Table 676. Access control points affecting multiple services or requiring special consideration

Name	Callable services	Notes	Value (hex)	Usage
Allow RSA2048 to wrap stronger keys for CKM-RAKW	CSNDSYX / CSNFSYX, CSNDPKT / CSNFPKT	When enabled, this control allows the services to wrap a stronger key with a weaker RSA key.	033E	DD, SC

Table 676. Access control points affecting multiple services or requiring special consideration (continued)				
Name	**Callable services**	**Notes**	**Value (hex)**	**Usage**
Allow weak DES wrap of RSA	CSNDPKG / CSNFPKG, CSNDPKI / CSNFPKI, CSNDPKT / CSNFPKT	A weaker DES key-encrypting key is allowed to wrap an RSA private key token. The **Prohibit weak wrap – Transport keys** access control must be enabled and this access control will override the restriction.	0331	DD, SC
Allow weak wrapping of compliance-tagged keys by DES MK	All callable services that use compliant-tagged DES key tokens.		02EB	DD, SC
ANSI X9.8 PIN - Allow modification of PAN	CSNBPTR / CSNEPTR, CSNBPTR2 / CSNEPTR2, CSNBPTRE / CSNEPTRE, CSNBSPN / CSNESPN	See “ANSI X9.8 PIN restrictions” on page 606 for a description of this control.	0351	DD, SC
ANSI X9.8 PIN - Allow only ANSI PIN blocks	CSNBPTR / CSNEPTR, CSNBPTR2 / CSNEPTR2, CSNBPTRE / CSNEPTRE, CSNBPVR2 / CSNEPVR2, CSNBSPN / CSNESPN	See “ANSI X9.8 PIN restrictions” on page 606 for a description of this control.	0352	DD, SC
ANSI X9.8 PIN - Enforce PIN block restrictions	CSNBCPA / CSNECPA, CSNBPTR / CSNEPTR, CSNBPTR2 / CSNEPTR2, CSNBPTRE / CSNEPTRE, CSNBPFO / CSNEPFO, CSNBSPN / CSNESPN	See “ANSI X9.8 PIN restrictions” on page 606 for a description of this control.	0350	DD, SC

Table 676. Access control points affecting multiple services or requiring special consideration (continued)				
Name	**Callable services**	**Notes**	**Value (hex)**	**Usage**
ANSI X9.8 PIN - Use stored decimalization tables only	CSNBCPA / CSNECPA, CSNBEPG / CSNEEPG, CSNBPFO / CSNEPFO, CSNBPGN / CSNEPGN, CSNBPVR / CSNEPVR	See "ANSI X9.8 PIN restrictions" on page 606 for a description of this control.	0356	DD, SC
Authenticated Key Export - DRVTXKEY	CSNBSYD / CSFNESYD, CSNBSYD1 / CSNESYD1, CSNBSYE / CSNESYE, CSNBSYE1 / CSNESYE1, CSNBFLD / CSNEFLD, CSNBFLE / CSNEFLE, CSNBKRR2 / CSNEKRR2	Required in order to establish a secure communication channel between the coprocessor and CPACF.	02F6	AE
Authenticated Key Export - EXPTSK	CSNBSYD / CSFNESYD, CSNBSYD1 / CSNESYD1, CSNBSYE / CSNESYE, CSNBSYE1 / CSNESYE1, CSNBFLD / CSNEFLD, CSNBFLE / CSNEFLE, CSNBKRR2 / CSNEKRR2	Required in order to export secure key tokens to CPACF protected key format.	02F7	AE

Table 676. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable services	Notes	Value (hex)	Usage
Authenticated Key Export - SETSNKEY	CSNBSYD / CSFNESYD, CSNBSYD1 / CSNESYD1, CSNBSYE / CSNESYE, CSNBSYE1 / CSNESYE1, CSNBFLD / CSNEFLD, CSNBFLE / CSNEFLE, CSNBKRR2 / CSNEKRR2	Required in order to establish a secure communication channel between the coprocessor and CPACF.	02F5	AE
DATAM Key Management Control	CSNBKGN / CSNEKGN, CSNBKIM / CSNEKIM, CSNBKEX / CSNEKEX, CSNBDKG / CSNEDKG	When enabled, the DATAM and DATAMV key types can be used. When disabled, the key types are not allowed.	0275	ED
Disable 56-bit effective length DES keys	All CCA callable services that accept or generate 56-bit effective length DES keys including loading master keys.	When enabled, all requests to CCA callable services with 56-bit effective length DES keys (112-bit or 168-bit keys with repeated 56-bit sections) will fail. This will also disallow loading a master key that has a 56-bit effective length.	0027	DD, SC
Disable 56-bit length DES keys	All CCA callable services that accept or generate 56-bit length DES keys.	When enabled, all requests to CCA callable services with 56-bit length DES keys will fail.	0026	DD, SC
Disable ECC keys weaker than 224-bit (P192, BP160, BP192)	All CCA callable services that accept or generate ECC keys weaker than 224-bit.	When enabled, all requests to CCA callable services with ECC keys weaker than 224-bit (P192, BP160, BP192) will fail.	004D	DD, SC

Table 676. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable services	Notes	Value (hex)	Usage
Disable RSA keys with less than 1024-bit modulus length	All CCA callable services that accept or generate RSA keys with less than 1024-bit modulus length.	When enabled, all requests to CCA callable services with RSA keys with a modulus length less than 1024-bit will fail.	002B	DD, SC
Disable RSA keys with less than 2048-bit modulus length	All CCA callable services that accept or generate RSA keys with less than 2048-bit modulus length.	When enabled, all requests to CCA callable services with RSA keys with a modulus length less than 2048-bit will fail.	002C	DD, SC
Disallow 24-byte DATA wrapped with 16-byte Key	All callable services that wrap key under an exporter or importer KEK or a 16-byte DES master key	When enabled, a triple-length 0 CV DATA keys cannot be wrapped by a 16-byte DES Key, either the master key or a key-encrypting key. See "Key strength and wrapping of key" on page 33 for more information.	032D	DD, SC

Table 676. Access control points affecting multiple services or requiring special consideration (continued)				
Name	**Callable services**	**Notes**	**Value (hex)**	**Usage**
Disallow PIN block format ISO-1	CSNBCPE / CSNECPE, CSNBCPA / CSNECPA, CSNBEPG / CSNEEPG, CSNBPTR / CSNEPTR, CSNBPTR2 / CSNEPTR2, CSNBPTRE / CSNEPTRE, CSNBPVR / CSNEPVR, CSNBPCU / CSNEPCU, CSNBPFO / CSNEPFO, CSNBSPN / CSNESPN, CSNBDMP / CSNEDMP, CSNBDPMT / CSNEDPMT, CSNBDPC / CSNEDPC, CSNBDPV / CSNEDPV	When the format in the input or output PIN block profile is ISO-1, the request will fail.	032F	DD, SC
Disallow translation from AES wrapping to DES wrapping	CSNBKTR2 / CSNEKTR2, CSNBPTR2 / CSNEPTR2, CSNDPKT / CSNFPKT	Disallows a key, PIN block, or PIN from being unwrapped or generated by an AES key and the wrapped by a DES key.	01C5	DD
Disallow translation from AES wrapping to weaker AES wrapping	CSNBKTR2 / CSNEKTR2, CSNBPTR2 / CSNEPTR2, CSNDPKT / CSNFPKT	Disallows a key, PIN block, or PIN from being unwrapped or generated by an AES key and the wrapped by a weaker AES key.	01C6	DD

Table 676. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable services	Notes	Value (hex)	Usage
Disallow translation from DES wrapping to weaker DES wrapping	CSNBAPG / CSNEAPG, CSNBEPG / CSNEEPG, CSNBKTR / CSNEKTR, CSNBKTR2 / CSNEKTR2, CSNBPFO / CSNEPFO, CSNBPTR / CSNEPTR, CSNBPTRE / CSNEPTRE, CSNBPTR2 / CSNEPTR2, CSNBSKY / CSNESKY, CSNDPKT / CSNFPKT	Disallows a key, PIN block, or PIN from being unwrapped or generated by a DES key and the wrapped by a weaker DES key.	01C7	DD
DUKPT - PIN Verify, PIN Translate	CSNBFPED / CSNEFPED, CSNBFPEE / CSNEFPEE, CSNBFPET / CSNEFPET, CSNBPVR / CSNEPVR, CSNBPTR / CSNEPTR, CSNBPTR2 / CSNEPTR2, CSNBPTRE / CSNEPTRE	When enabled, the listed services can use DUKPT key derivation.	00E1	ED
Encrypted PIN Translate - Translate PIN Check Mode	CSNBPTR / CSNEPTR, CSNBPTR2 / CSNEPTR2	See "Enhanced PIN checking for CSNBPTR and CSNBPTR2" on page 609 for a description of this control.	03A0	DD, SC

Table 676. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable services	Notes	Value (hex)	Usage
Enhanced PIN Security	CSNBCPE / CSNECPE, CSNBCPA / CSNECPA, CSNBEPG / CSNEEPG, CSNBPTR / CSNEPTR, CSNBPTR2 / CSNEPTR2, CSNBPTRE / CSNEPTRE, CSNBPVR / CSNEPVR, CSNBPVR2 / CSNEPVR2, CSNBPCU / CSNEPCU, CSNBPFO / CSNEPFO	See "Enhanced PIN security mode" on page 608 for a description of this control.	0313	DD, SC
General ISO PIN Error Security	CSNBPTR / CSNEPTR, CSNBPTR2 / CSNEPTR2, CSNBDPC / CSNEDPC, CSNBDPV / CSNEDPV	See "PIN block error processing mode" on page 609 for a description of this control.	039F	DD, SC
High-performance secure AES keys	CSNBSYD / CSFNESYD, CSNBSYD1 / CSNESYD1, CSNBSYE / CSNESYE, CSNBSYE1 / CSNESYE1, CSNBFLD / CSNEFLD, CSNBFLE / CSNEFLE, CSNBKRR2 / CSNEKRR2, CSFWRP / CSFWRP6	When enabled, encrypted AES DATA key tokens in the CKDS can be used for the CPACF instructions. Required for CSFWRP/CSFWRP6.	0296	ED

Table 676. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable services	Notes	Value (hex)	Usage
High-performance secure DES keys	CSNBSYD / CSFNESYD, CSNBSYD1 / CSNESYD1, CSNBSYE / CSNESYE, CSNBSYE1 / CSNESYE1, CSNBFLD / CSNEFLD, CSNBFLE / CSNEFLE, CSNBKRR2 / CSNEKRR2	When enabled, encrypted DES DATA key tokens in the CKDS can be used for the CPACF instructions.	0295	ED
ISO PIN blocks do not check PIN digits	CSNBCPA / CSNECPA, CSNBPTR / CSNEPTR, CSNBPTR2 / CSNEPTR2, CSNBPTRE / CSNEPTRE, CSNBPVR / CSNEPVR, CSNBPVR2 / CSNEPVR2, CSNBPCU / CSNEPCU, CSNBSPN / CSNESPN, CSNBDMP / CSNEDMP, CSNBDPMT / CSNEDPMT, CSNBDPT / CSNEDPT, CSNBDPC / CSNEDPC, CSNBDPV / CSNEDPV, CSNBDPNU / CSNEDPNU, CSNBDCU2 / CSNEDCU2, CSNBDRP / CSNEDRP	When enabled, the affected services do not verify that the PIN digits conform to permissible values	0055	ED

Table 676. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable services	Notes	Value (hex)	Usage
NOCV KEK usage for export-related functions	CSNBGIM / CSNEGIM, CSNBKEX / CSNEKEX, CSNBSKM / CSNESKM, CSNBKGN / CSNEKGN	When enabled, NOCV key-encrypting keys can be used by the listed services.	0300	ED, SC
NOCV KEK usage for import-related functions	CSNBKIM / CSNEKIM, CSNBSKI / CSNESKI, CSNBSKM / CSNESKM, CSNBKGN / CSNEKGN	When enabled, NOCV key-encrypting keys can be used by the listed services.	030A	ED, SC
Prohibit weak wrapping - Master keys	All Services that wrap or import keys. Both symmetric and asymmetric keys are affected	When enabled, an error return code will be returned when attempting to wrap a stronger key with a weaker master key. Also, an error return code will be returned when the last part is loaded into the DES or RSA new master key register, if the complete master key is weak. See "Key strength and wrapping of key" on page 33 and "Key strength and wrapping of key" on page 94 for more information.	0333	DD, SC
Prohibit weak wrapping - Transport keys	All Services that wrap or import keys. Both symmetric and asymmetric keys are affected	When enabled, an error return code will be returned when attempting to wrap a stronger key with a weaker key-encrypting key. See "Key strength and wrapping of key" on page 33 for more information.	0328	DD, SC
Symmetric Key Token Change2 - RTCMK	Services that use the variable-length symmetric key tokens	When enabled, this control allows symmetric key tokens under the old master key to be reenciphered under the current master key. These reenciphered tokens are returned from all callable service that use symmetric tokens.	00F1	AE
Symmetric Key Token Change - RTCMK	Services that use symmetric key tokens	When enabled, this control allows symmetric key tokens under the old master key to be reenciphered under the current master key. These reenciphered tokens are returned from all callable service that use symmetric tokens.	0090	AE

Table 676. Access control points affecting multiple services or requiring special consideration (continued)				
Name	**Callable services**	**Notes**	**Value (hex)**	**Usage**
Symmetric token wrapping - External enhanced method	Services that wrap external symmetric key tokens	When enabled, this control allows ICSF to change the default wrapping setting for all generated or exported keys to be the enhanced method. The default wrapping can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 33 for more information.	013B	AE
Symmetric token wrapping - External enhanced method version 3	Services that wrap external symmetric key tokens	When enabled, this control allows ICSF to change the default wrapping setting for all generated or exported keys to be the enhanced method version 3. The default wrapping can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 33 for more information.	0145	AE
Symmetric token wrapping - External original method	Services that wrap external symmetric key tokens	When enabled, this control allows ICSF to change the default wrapping setting for all generated or exported keys to be the original method. The default wrapping can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 33 for more information.	013C	AE
Symmetric token wrapping - Internal enhanced method	Services that wrap internal symmetric key tokens	When enabled, this control allows ICSF to change the default wrapping setting for all generated or imported keys to be the enhanced method. The default wrapping can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 33 for more information.	0139	AE
Symmetric token wrapping - Internal enhanced method version 3	Services that wrap internal symmetric key tokens	When enabled, this control allows ICSF to change the default wrapping setting for all generated or imported keys to be the enhanced method version 3. The default wrapping can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 33 for more information.	0143	AE
Symmetric token wrapping - Internal original method	Services that wrap internal symmetric key tokens	When enabled, this control allows ICSF to change the default wrapping setting for all generated or imported keys to be the original method. The default wrapping can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 33 for more information.	013A	AE
TR-34 - Allow expired CRL	CSNDT34B/ CSNFT34B, CSNDT34C/ CSNFT34C, CSNDT34D/ CSNFT34D, CSNDT34R/ CSNFT34R	When enabled, the CRLEXPAL rule is permitted. This rule allows the certificate revocation list (CRL) to be used if it is expired. Instead of a failure, an informational reason code is returned.	003D	ED

Table 676. Access control points affecting multiple services or requiring special consideration (continued)				
Name	**Callable services**	**Notes**	**Value (hex)**	**Usage**
TR-34 - Allow expired KRD Certificate	CSNDT34B/ CSNFT34B, CSNDT34C/ CSNFT34C, CSNDT34D/ CSNFT34D	When enabled, the RCTEXPAL rule is permitted. This rule allows the KRD credential X.509 certificate to be used if it is expired. Instead of a failure, an informational reason code is returned.	003E	ED
Warn when weak wrap - Master keys	All Services that wrap or import keys. Both symmetric and asymmetric keys are affected	When enabled, an informational return code will be returned when attempting to wrap a stronger key with a master key that is weaker. Also, a warning return code will be returned when the last part is loaded into the DES or RSA new master key register, if the master key is weak. See “Key strength and wrapping of key” on page 33 and “Key strength and wrapping of key” on page 94 for more information.	0332	DD. SC
Warn when weak wrap - Transport keys	All Services that wrap or import keys. Both symmetric and asymmetric keys are affected	When enabled, an informational return code will be returned when attempting to wrap a stronger key with a weaker key or when attempting to import a key token that has previously been wrapped with a weaker key, as indicated by its security history field. See “Key strength and wrapping of key” on page 33 and “Key strength and wrapping of key” on page 94 for more information.	032C	DD. SC

There are relationships between certain access control points. A controlling access control point is required to be enabled before subordinate access control points can enabled. The TKE workstation will enable the controlling access control point when a subordinate access control point is enabled.

- The Allow weak DES wrap of RSA access control point is only checked if the Prohibit weak wrapping – Transport keys access control point is enabled.
- The ANSI X9.8 PIN - Allow modification of PAN and ANSI X9.8 PIN - Allow only ANSI PIN blocks access control points can only be enable when the ANSI X9.8 PIN - Enforce PIN block restrictions access control point is enabled.

This following table lists access control points that affect specific services indicated in the access control point name. There is a description of the usage of the access control point in the Usage Notes section of the callable service description.

Note: If the domain role has been changed via the TKE workstation, all new access control points are disabled by default.

Table 677. Access control points - Callable Services			
Name	**Callable service**	**Value (Hex)**	**Usage**
Authentication Parameter Generate	CSNBAPG / CSNEAPG	02B1	ED
Authentication Parameter Generate - Clear	CSNBAPG / CSNEAPG	02B2	ED
Cipher Text Translate2	CSNBCTT2 / CSNECTT2, CSNBCTT3 / CSNECTT3	01C0	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
Cipher Text Translate2 - Allow only cipher text translate types	CSNBCTT2 / CSNECTT2, CSNBCTT3 / CSNECTT3	01C4	DD
Cipher Text Translate2 - Allow translate from AES to TDES	CSNBCTT2 / CSNECTT2, CSNBCTT3 / CSNECTT3	01C1	ED
Cipher Text Translate2 - Allow translate to weaker AES	CSNBCTT2 / CSNECTT2, CSNBCTT3 / CSNECTT3	01C2	ED
Cipher Text Translate2 - Allow translate to weaker DES	CSNBCTT2 / CSNECTT2, CSNBCTT3 / CSNECTT3	01C3	ED
Clear Key Import / Multiple Clear Key Import - DES	CSNBCKI / CSNECKI, CSNBCKM / CSNECKM	00C3	ED
Clear PIN Encrypt	CSNBCPE / CSNECPE	00AF	ED
Clear PIN Generate - 3624	CSNBPGN / CSNEPGN	00A0	ED
Clear PIN Generate Alternate - 3624 Offset	CSNBCPA / CSNECPA	00A4	ED
Clear PIN Generate Alternate - VISA PVV	CSNBCPA / CSNECPA	00BB	ED
Clear PIN Generate - GBP	CSNBPGN / CSNEPGN	00A1	ED
Clear PIN Generate - Interbank	CSNBPGN / CSNEPGN	00A3	ED
Clear PIN Generate - VISA PVV	CSNBPGN / CSNEPGN	00A2	ED
Control Vector Translate	CSNBCVT / CSNECVT	00D6	ED
Cryptographic Variable Encipher	CSNBCVE / CSNECVE	00DA	ED
CVV Key Combine	CSNBCKC / CSNECKC	0155	ED
CVV Key Combine - Allow wrapping override keywords	CSNBCKC / CSNECKC	0156	ED
CVV Key Combine - Permit mixed key types	CSNBCKC / CSNECKC	0157	ED
Data Key Export	CSNBDKX / CSNEDKX	010A	ED
Data Key Export - Unrestricted	CSNBDKX / CSNEDKX	0277	ED
Data Key Import	CSNBDKM / CSNEDKM	0109	ED
Data Key Import - Unrestricted	CSNBDKM / CSNEDKM	027C	ED
Decipher - DES	CSNBDEC / CSNEDEC, CSNBEVF / CSNEEVF	000F	ED
Digital Signature Generate	CSNDDSG / CSNFDSG	0100	ED
Digital Signature Generate - PKCS-PSS allow small salt	CSNDDSG / CSNFDSG	033C	DD
Digital Signature Verify	CSNDDSV / CSNFDSV	0101	ED
Digital Signature Verify - PKCS-PSS allow not exact salt length	CSNDDSV / CSNFDSV	033B	DD
Diversified Key Generate2 - Allow length option for KDFFM-DK	CSNBDKG2 / CSNEDKG2	02D4	DD
Diversified Key Generate2 - DALL	CSNBDKG2 / CSNEDKG2	02CD	DD, SC

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
Diversified Key Generate2 - KDFFM-DK	CSNBDKG2 / CSNEDKG2	02D3	ED
Diversified Key Generate2 - MK-OPTC	CSNBDKG2 / CSNEDKG2	02D2	ED
Diversified Key Generate2 - SESS-ENC	CSNBDKG2 / CSNEDKG2	02CC	ED
Diversified Key Generate - A28OWFCL	CSNBDKG / CSNEDKG	03BA	ED
Diversified Key Generate - A28OWFEC	CSNBDKG / CSNEDKG	03B9	ED
Diversified Key Generate - A28XOREC	CSNBDKG / CSNEDKG	03BB	ED
Diversified Key Generate - Allow wrapping override keywords	CSNBDKG / CSNEDKG	013D	ED
Diversified Key Generate - CLR8-ENC	CSNBDKG / CSNEDKG	0040	ED
Diversified Key Generate - DKYGENKY - DALL	CSNBDKG / CSNEDKG, CSNBPCU / CSNEPCU	0290	DD, SC
Diversified Key Generate - SESS-XOR	CSNBDKG / CSNEDKG, CSNBESC / CSNEESC, CSNBEVF / CSNEEVF	0043	ED
Diversified Key Generate - Single length or same halves	CSNBDKG / CSNEDKG	0044	ED
Diversified Key Generate - TDES-CBC	CSNBDKG / CSNEDKG	02B8	ED
Diversified Key Generate - TDES-DEC	CSNBDKG / CSNEDKG	0042	ED
Diversified Key Generate - TDESEMV2/TDESEMV4	CSNBDCM / CSNEDCM, CSNBDKG / CSNEDKG, CSNBDSK / CSNEDSK, CSNBEAC / CSNEEAC, CSNBESC / CSNEESC, CSNBEVF / CSNEEVF	0046	ED
Diversified Key Generate - TDES-ENC	CSNBDCM / CSNEDCM, CSNBDKG / CSNEDKG, CSNBDSK / CSNEDSK, CSNBEAC / CSNEEAC, CSNBESC / CSNEESC, CSNBEVF / CSNEEVF	0041	ED
Diversified Key Generate - TDES-XOR	CSNBDCM / CSNEDCM, CSNBDKG / CSNEDKG, CSNBDSK / CSNEDSK, CSNBEAC / CSNEEAC, CSNBESC / CSNEESC	0045	ED
Diversify Directed Key	CSNBDDK / CSNEDDK	0080	DD
Diversify Directed Key - Allow KDFFM DERIVE	CSNBDDK / CSNEDDK	0081	DD
Diversify Directed Key - Allow KDFFM GENERATE	CSNBDDK / CSNEDDK	0082	DD
DK Deterministic PIN Generate	CSNBDDPG / CSNEDDPG	02C6	DD
DK Migrate PIN	CSNBDMP / CSNEDMP	02CE	DD
DK PAN Modify in Transaction	CSNBDPMT / CSNEDPMT	02C5	DD

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
DK PAN Translate	CSNBDPT / CSNEDPT	02C7	DD
DK PIN Change	CSNBDPC / CSNEDPC	02C2	DD
DK PIN Verify	CSNBDPV / CSNEDPV	02C1	DD
DK PRW Card Number Update	CSNBDPNU / CSNEDPNU	02C3	DD
DK PRW Card Number Update2	CSNBDCU2 / CSNEDCU2	0025	DD
DK PRW CMAC Generate	CSNBDPCG / CSNBPCG	02C4	DD
DK Random PIN Generate	CSNBDRPG / CSNEDRPG	02C0	DD
DK Random PIN Generate2	CSNBDRG2 / CSNEDRG2	0024	DD
DK Regenerate PRW	CSNBDRP / CSNEDRP	02C8	DD
DSG - ZERO-PAD unrestricted hash length	CSNDDSG / CSNFDSG	030C	DD
ECC Diffie-Hellman	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0360	ED
ECC Diffie-Hellman - Allow BP Curve 160	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0368	ED
ECC Diffie-Hellman - Allow BP Curve 192	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0369	ED
ECC Diffie-Hellman - Allow BP Curve 224	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	036A	ED
ECC Diffie-Hellman - Allow BP Curve 256	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	036B	ED
ECC Diffie-Hellman - Allow BP Curve 320	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	036C	ED
ECC Diffie-Hellman - Allow BP Curve 384	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	036D	ED
ECC Diffie-Hellman - Allow BP Curve 512	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	036E	ED
ECC Diffie-Hellman - Allow DERIV02	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	035F	ED
ECC Diffie-Hellman - Allow Hybrid QSA Scheme	CSNDEDH / CSNFEDH	035D	ED
ECC Diffie-Hellman - Allow key wrap override	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0362	ED
ECC Diffie-Hellman - Allow Koblitz Curve 256	CSNDEDH / CSNFEDH	035E	ED
ECC Diffie-Hellman - Allow PASSTHRU	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0361	ED
ECC Diffie-Hellman - Allow Prime Curve 192	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0363	ED
ECC Diffie-Hellman - Allow Prime Curve 224	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0364	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
ECC Diffie-Hellman - Allow Prime Curve 256	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0365	ED
ECC Diffie-Hellman - Allow Prime Curve 384	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0366	ED
ECC Diffie-Hellman - Allow Prime Curve 521	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	0367	ED
ECC Diffie-Hellman - Prohibit weak key generate	CSFPDVK / CSFPDVK6, CSNDEDH / CSNFEDH	036F	DD, SC
Encipher - DES	CSNBENC / CSNEENC, CSNBESC / CSNEESC, CSNBEVF / CSNEEVF	000E	ED
Encrypted PIN Generate - 3624	CSNBEPG / CSNEEPG	00B0	ED
Encrypted PIN Generate - GBP	CSNBEPG / CSNEEPG	00B1	ED
Encrypted PIN Generate - Interbank	CSNBEPG / CSNEEPG	00B2	ED
Encrypted PIN Translate2 - Permit ISO-0 to ISO-4 Reformat	CSNBPTR2 / CSNEPTR2	038E	ED
Encrypted PIN Translate2 - Permit ISO-1 to ISO-4 Reformat	CSNBPTR2 / CSNEPTR2	038C	ED
Encrypted PIN Translate2 - Permit ISO-1 to ISO-4 RFMT1TO4	CSNBPTR2 / CSNEPTR2	0393	DD
Encrypted PIN Translate2 - Permit ISO-4 Reformat w/ PAN Chg	CSNBPTR2 / CSNEPTR2	038B	DD
Encrypted PIN Translate2 - Permit ISO-4 to ISO-0 Reformat	CSNBPTR2 / CSNEPTR2	038F	ED
Encrypted PIN Translate2 - Permit ISO-4 to ISO-1 Reformat	CSNBPTR2 / CSNEPTR2	038D	ED
Encrypted PIN Translate2 - Permit ISO-4 to ISO-1 RFMT4TO1	CSNBPTR2 / CSNEPTR2	0394	DD
Encrypted PIN Translate2 - Permit ISO-4 to ISO-4 PTR2AUTH	CSNBPTR2 / CSNEPTR2	0395	DD
Encrypted PIN Translate2 - Permit ISO-4 to ISO-4 Translate	CSNBPTR2 / CSNEPTR2	038A	ED
Encrypted PIN Translate2 - REFORMAT	CSNBPTR2 / CSNEPTR2	0391	ED
Encrypted PIN Translate2 - TRANSLAT	CSNBPTR2 / CSNEPTR2	0392	ED
Encrypted PIN Translate Enhanced	CSNBPTRE / CSNEPTRE	02D5	ED
Encrypted PIN Translate - Reformat	CSNBPTR / CSNEPTR, CSNBPTRE / CSNEPTRE, CSNBPTR2 / CSNEPTR2	00B7	ED
Encrypted PIN Translate - Translate	CSNBPTR / CSNEPTR	00B3	ED
Encrypted PIN Verify2 – REFPIN	CSNBPVR2 / CSNEPVR2	03B0	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
Encrypted PIN Verify2 – TRUNCPIN	CSNBPVR2 / CSNEPVR2	03B1	ED
Encrypted PIN Verify - 3624	CSNBPVR / CSNEPVR	00AB	ED
Encrypted PIN Verify - GBP	CSNBPVR / CSNEPVR	00AC	ED
Encrypted PIN Verify - Interbank	CSNBPVR / CSNEPVR	00AE	ED
Encrypted PIN Verify - VISA PVV	CSNBPVR / CSNEPVR	00AD	ED
Format Preserving Algorithms Decipher	CSNBFFXD	039A	ED
Format Preserving Algorithms Encipher	CSNBFFXE	0399	ED
Format Preserving Algorithms Encipher/Decipher - Allow FF1	CSNBFFXD / CSNBFFXE / CSNBFFXT	0396	ED
Format Preserving Algorithms Encipher/Decipher - Allow FF2	CSNBFFXD / CSNBFFXE / CSNBFFXT	0397	ED
Format Preserving Algorithms Encipher/Decipher - Allow FF2.1	CSNBFFXD / CSNBFFXE / CSNBFFXT	0398	ED
Format Preserving Algorithms Translate	CSNBFFXT	039B	ED
Format Preserving Algorithms Translate - Allow weaker output key	CSNBFFXT	039C	ED
FPE Decrypt	CSNBFPED / CSNEFPED	02D0	ED
FPE Encrypt	CSNBFPEE / CSNEFPEE	02CF	ED
FPE Translate	CSNBFPET / CSNEFPET	02D1	ED
HMAC Generate - SHA-1	CSNBHMG / CSNEHMG, CSNBHMG1 / CSNEHMG1, CSNBMGN2 / CSNEMGN2, CSNBMGN3 / CSNEMGN3	00E4	ED
HMAC Generate - SHA-224	CSNBHMG / CSNEHMG, CSNBHMG1 / CSNEHMG1, CSNBMGN2 / CSNEMGN2, CSNBMGN3 / CSNEMGN3	00E5	ED
HMAC Generate - SHA-256	CSNBHMG / CSNEHMG, CSNBHMG1 / CSNEHMG1, CSNBMGN2 / CSNEMGN2, CSNBMGN3 / CSNEMGN3	00E6	ED
HMAC Generate - SHA-384	CSNBHMG / CSNEHMG, CSNBHMG1 / CSNEHMG1, CSNBMGN2 / CSNEMGN2, CSNBMGN3 / CSNEMGN3	00E7	ED
HMAC Generate - SHA-512	CSNBHMG / CSNEHMG, CSNBHMG1 / CSNEHMG1, CSNBMGN2 / CSNEMGN2, CSNBMGN3 / CSNEMGN3	00E8	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
HMAC Verify - SHA-1	CSNBHMV / CSNEHMV, CSNBHMV1 / CSNEHMV1, CSNBMVR2 / CSNEMVR2, CSNBMVR3 / CSNEMVR3	00F7	ED
HMAC Verify - SHA-224	CSNBHMV / CSNEHMV, CSNBHMV1 / CSNEHMV1, CSNBMVR2 / CSNEMVR2, CSNBMVR3 / CSNEMVR3	00F8	ED
HMAC Verify - SHA-256	CSNBHMV / CSNEHMV, CSNBHMV1 / CSNEHMV1, CSNBMVR2 / CSNEMVR2, CSNBMVR3 / CSNEMVR3	00F9	ED
HMAC Verify - SHA-384	CSNBHMV / CSNEHMV, CSNBHMV1 / CSNEHMV1, CSNBMVR2 / CSNEMVR2, CSNBMVR3 / CSNEMVR3	00FA	ED
HMAC Verify - SHA-512	CSNBHMV / CSNEHMV, CSNBHMV1 / CSNEHMV1, CSNBMVR2 / CSNEMVR2, CSNBMVR3 / CSNEMVR3	00FB	ED
Key Encryption Translate - CBC to ECB	CSNBKET / CSNEKET	030D	DD, ED - March 2016 or later licensed internal code (LIC) on IBM z13 and above processors.
Key Encryption Translate - ECB to CBC	CSNBKET / CSNEKET	030E	DD, ED - March 2016 or later licensed internal code (LIC) on IBM z13 and above processors.

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
Key Export	CSNBKEX / CSNEKEX	0013	ED
Key Export - Unrestricted	CSNBKEX / CSNEKEX	0276	ED
Key Generate2 - Allow GEN of OPOP EPVR/OPIN Key Pair	CSNBKGN2 / CSNEKGN2	039D	DD
Key Generate2 - DK PIN admin1 key set MAC	CSNBKGN2 / CSNEKGN2	02BE	DD
Key Generate2 - DK PIN admin1 key set PINPROT	CSNBKGN2 / CSNEKGN2	02BD	DD
Key Generate2 - DK PIN admin2 key set MAC	CSNBKGN2 / CSNEKGN2	02BF	DD
Key Generate2 - DK PIN key set	CSNBKGN2 / CSNEKGN2	02BB	DD
Key Generate2 - DK PIN print key	CSNBKGN2 / CSNEKGN2	02BC	DD
Key Generate2 - Key set	CSNBKGN2 / CSNEKGN2	00EB	ED
Key Generate2 - Key set extended	CSNBKGN2 / CSNEKGN2	00EC	ED
Key Generate2 - OP	CSNBKGN2 / CSNEKGN2	00EA	ED
Key Generate - Key set	CSNBKGN / CSNEKGN	008C	ED
Key Generate - Key set extended	CSNBKGN / CSNEKGN	00D7	ED
Key Generate - OP	CSNBGIM / CSNEGIM, CSNBKGN / CSNEKGN, CSNBRNG / CSNERNG	008E	ED
Key Generate - SINGLE-R	CSNBKGN / CSNEKGN, CSNDRKX / CSNFRKX	00DB	ED
Key Import	CSNBKIM / CSNEKIM	0012	ED
Key Import - Unrestricted	CSNBKIM / CSNEKIM	027B	ED
Key Part Import2 - Add last required key part	CSNBKPI2 / CSNEKPI2	029B	ED
Key Part Import2 - Add optional key part	CSNBKPI2 / CSNEKPI2	029C	ED
Key Part Import2 - Add second of 3 or more key parts	CSNBKPI2 / CSNEKPI2	029A	ED
Key Part Import2 - Complete key	CSNBKPI2 / CSNEKPI2	029D	ED
Key Part Import2 - Load first key part, require 1 key parts	CSNBKPI2 / CSNEKPI2	0299	ED
Key Part Import2 - Load first key part, require 2 key parts	CSNBKPI2 / CSNEKPI2	0298	ED
Key Part Import2 - Load first key part, require 3 key parts	CSNBKPI2 / CSNEKPI2	0297	ED
Key Part Import - ADD-PART	CSNBKPI / CSNEKPI	0278	ED
Key Part Import - Allow wrapping override keywords	CSNBKPI / CSNEKPI	0140	ED
Key Part Import - COMPLETE	CSNBKPI / CSNEKPI	0279	ED
Key Part Import - first key part	CSNBKPI / CSNEKPI	001B	ED
Key Part Import - middle and last	CSNBKPI / CSNEKPI	001C	ED
Key Part Import - Unrestricted	CSNBKPI / CSNEKPI	027A	ED
Key Test2 - AES, CMACZERO	CSNBKYT2 / CSNEKYT2	0022	ED
Key Test2 - AES, ENC-ZERO	CSNBKYT2 / CSNEKYT2	0021	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
Key Test2 - AES, KEY-LEN	CSNBKYT2 / CSNEKYT2	003B	ED
Key Test2 - DES, CMACZERO	CSNBKYT2 / CSNEKYT2	0023	ED
Key Test2 - DES, KEY-LEN	CSNBKYT2 / CSNEKYT2	003C	ED
Key Test and Key Test2	CSNBKYT / CSNEKYT, CSNBKYT2 / CSNEKYT2, CSNBKYTX / CSNEKYTX	001D	AE
Key Test - Warn when keyword inconsistent with key length	CSNBKYT / CSNEKYT, CSNBKYTX / CSNEKYTX	01CB	DD
Key Translate	CSNBKTR / CSNEKTR	001F	ED
Key Translate2	CSNBKTR2 / CSNEKTR2	0149	ED
Key Translate2 - Allow use of REFORMAT	CSNBKTR2 / CSNEKTR2	014B	ED
Key Translate2 - Allow wrapping override keywords	CSNBKTR2 / CSNEKTR2	014A	ED
Key Translate2 - COMP-CHK	CSNBKTR2/CSNEKTR2	02F9	ED
Key Translate2 - COMP-TAG	CSNBKTR2/CSNEKTR2	02F8	ED
Key Translate2 - Disallow AES ver 5 to ver 4 conversion	CSNBKTR2 / CSNEKTR2	032A	DD
Key Translate2 - Translate fixed to variable payload	CSNBKTR2 / CSNEKTR2	0334	DD, SC
KPI2 - Allow TR-31 clear key import	CSNBKPI2 / CSNEKPI2	03BC	ED
MAC Generate	CSNBEAC / CSNEEAC, CSNBESC / CSNEESC, CSNBMGN / CSNEMGN	0010	ED
MAC Generate2 - AES CMAC	CSNBMGN2 / CSNEMGN2 / CSNBMGN3 / CSNEMGN3	0336	ED
MAC Verify	CSNBEAC / CSNEEAC, CSNBMVR / CSNEMVR	0011	ED
MAC Verify2 - AES CMAC	CSNBMVR2 / CSNEMVR2 / CSNBMVR3 / CSNEMVR3	0337	ED
Multiple Clear Key Import / Multiple Secure Key Import - AES	CSNBCKM / CSNECKM, CSNBSKM / CSNESKM	0129	ED
Multiple Clear Key Import - Allow wrapping override keywords	CSNBCKM / CSNECKM	0141	ED
Multiple Secure Key Import - Allow wrapping override keywords	CSNBSKM / CSNESKM	0142	ED
Operational Key Load	CSNBOKL / CSNEOKL	0309	ED
Operational Key Load - Variable-Length Tokens	CSNBOKL / CSNEOKL	029E	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
Permit X.509 without PKI root validation	CSNDDSV / CSNFDSV, CSNDPKE / CSNFPKE, CSNDSYX / CSNFSYX, CSNDSYG / CSNFSYG, CSNDT34B / CSNFT34B, CSNDT34C / CSNFT34C, CSNDT34D / CSNFT34D, CSNDT34R / CSNFT34R	01FF	ED
PIN Change/Unblock - Change EMV PIN with IPINENC	CSNBESC / CSNEESC, CSNBPCU / CSNEPCU	00BD	ED
PIN Change/Unblock - Change EMV PIN with OPINENC	CSNBESC / CSNEESC, CSNBPCU / CSNEPCU	00BC	ED
PKA Decrypt	CSNDPKD / CSNFPKD	011F	ED
PKA Decrypt - Allow CRYSTALS-Kyber keys	CSNDPKD / CSNFPKD	0084	ED
PKA Decrypt - Disallow PKCS-1.2	CSNDPKD / CSNFPKD	020A	DD
PKA Decrypt - Disallow PKCSOAEP	CSNDPKD / CSNFPKD	020C	DD
PKA Decrypt - Disallow PKOAEP2	CSNDPKD / CSNFPKD	03F2	DD
PKA Decrypt - Disallow ZEROPAD	CSNDPKD / CSNFPKD	020B	DD
PKA Encrypt	CSNDPKE / CSNFPKE	011E	ED
PKA Encrypt - Allow CRYSTALS-Kyber keys	CSNDPKE / CSNFPKE	0083	ED
PKA Encrypt - Disallow MRP	CSNDPKE / CSNFPKE	0208	DD
PKA Encrypt - Disallow PKCS-1.2	CSNDPKE / CSNFPKE	0206	DD
PKA Encrypt - Disallow PKCSOAEP	CSNDPKE / CSNFPKE	0209	DD
PKA Encrypt - Disallow PKOAEP2	CSNDPKE / CSNFPKE	03F1	DD
PKA Encrypt - Disallow ZEROPAD	CSNDPKE / CSNFPKE	0207	DD
PKA Key Generate	CSNDPKG / CSNFPKG	0103	ED
PKA Key Generate - Clear CRYSTALS-Dilithium keys	CSNDPKG / CSNFPKG	027F	ED
PKA Key Generate - Clear CRYSTALS-Kyber keys	CSNDPKG / CSNFPKG	020E	ED
PKA Key Generate - Clear ECC keys	CSNDPKG / CSNFPKG	0326	ED
PKA Key Generate - Clear RSA keys	CSNDPKG / CSNFPKG	0205	ED
PKA Key Generate - Clone	CSNDPKG / CSNFPKG	0204	ED
PKA Key Generate - Permit Regeneration Data	CSNDPKG / CSNFPKG	027D	ED
PKA Key Generate - Permit Regeneration Data Retain	CSNDPKG / CSNFPKG	027E	ED
PKA Key Import	CSNDPKI / CSNFPKI	0104	ED
PKA Key Import - Disallow clear key import	CSNDPKI / CSNFPKI	003A	DD, SC
PKA Key Import - Import an external trusted block	CSNDPKI / CSNFPKI	0311	ED
PKA Key Token Change RTCMK	CSNDKTC / CSNFKTC	0102	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
PKA Key Translate - Allow COMP-CHK	CSNDPKT / CSNFPKT	01EF	ED
PKA Key Translate - Allow COMP-TAG	CSNDPKT / CSNFPKT	01EE	ED
PKA Key Translate - Allow ECC private key export	CSNDPKT / CSNFPKT	00EF	DD
PKA Key Translate - Allow INTUSCHG	CSNDPKT / CSNFPKT	02EE	ED
PKA Key Translate - Allow QSA private key export	CSNDPKT / CSNFPKT	020F	DD
PKA Key Translate – From CCA ECC to CKM-RAKW format	CSNDPKT / CSNFPKT	03B7	DD
PKA Key Translate - From CCA RSA CRT to EMV CRT format	CSNDPKT / CSNFPKT	033A	ED
PKA Key Translate - From CCA RSA CRT to EMV DDAE format	CSNDPKT / CSNFPKT	0339	ED
PKA Key Translate - From CCA RSA CRT to EMV DDA format	CSNDPKT / CSNFPKT	0338	ED
PKA Key Translate – From CCA RSA to CKM-RAKW format	CSNDPKT / CSNFPKT	03B6	DD
PKA Key Translate - From CCA RSA to SC CRT format	CSNDPKT / CSNFPKT	031A	ED
PKA Key Translate - From CCA RSA to SC ME format	CSNDPKT / CSNFPKT	0319	ED
PKA Key Translate - From CCA RSA to SC Visa format	CSNDPKT / CSNFPKT	0318	ED
PKA Key Translate - From source EXP KEK to target EXP KEK	CSNDPKT / CSNFPKT	031B	ED
PKA Key Translate - From source IMP KEK to target EXP KEK	CSNDPKT / CSNFPKT	031C	ED
PKA Key Translate - From source IMP KEK to target IMP KEK	CSNDPKT / CSNFPKT	031D	ED
PKA Key Translate - Translate external key token	CSNDPKT / CSNFPKT	00FF	ED
PKA Key Translate - Translate internal key token	CSNDPKT / CSNFPKT	00FE	ED
Prohibit Export	CSNBPEX / CSNEPEX	00CD	ED
Prohibit Export Extended	CSNBPEXX / CSNEPEXX	0301	ED
Public Infrastructure Certificate	CSNDPIC / CSNFPIC	0070	ED
Public Infrastructure Certificate - PK10SNRQ	CSNDPIC / CSNFPIC	007C	ED
Random Number Generate Long – TDES-CBC	CSNBRNGL / CSNERNGL	03B5	ED
Recover PIN From Offset	CSNBPFO / CSNEPFO	02B0	ED
Remote Key Export - Allow wrapping override keywords	CSNDRKX / CSNFRKX	02BA	DD
Remote Key Export - Gen or export a non-CCA node key	CSNDRKX / CSNFRKX	0312	ED
Remote Key Export - Include RKX in default wrap config	CSNDRKX / CSNFRKX	013F	DD
Restrict Key Attribute - Export Control	CSNBRKA / CSNERKA	00E9	ED
Restrict Key Attribute - Permit setting the TR-31 Translate bit	CSNBRKA / CSNERKA	0154	ED
Retained Key Delete	CSNDRKD / CSNFRKD	0203	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
Retained Key List	CSNDRKL / CSNFRKL	0230	ED
Secure Key Import2 - IM	CSNBSKI2 / CSNESKI2	00F3	ED
Secure Key Import2 - OP	CSNBSKI2 / CSNESKI2	00F2	ED
Secure Key Import - DES, IM	CSNBSKI / CSNESKI, CSNBSKM / CSNESKM	00DC	ED
Secure Key Import - DES, OP	CSNBSKI / CSNESKI, CSNBSKM / CSNESKM	00C4	ED
Secure Messaging for Keys	CSNBSKY / CSNESKY	0273	ED
Secure Messaging for PINs	CSNBESC / CSNEESC, CSNBSPN / CSNESPN	0274	ED
SET Block Compose	CSNDSBC / CSNFSBC	010B	ED
SET Block Decompose	CSNDSBD / CSNFSBD	010C	ED
SET Block Decompose - PIN Extension IPINENC	CSNDSBD / CSNFSBD	0121	ED
SET Block Decompose - PIN Extension OPINENC	CSNDSBD / CSNFSBD	0122	ED
SKY - Allow K0 for secmsg key identifier	CSNBSKY / CSNESKY	03F3	ED
SPN - Allow K0 for secmsg key identifier	CSNBSPN / CSNESPN	03F4	ED
Symmetric Algorithm Decipher - Galois/Counter mode AES	CSNBSAD / CSNESAD, CSNBSAD1 / CSNESAD1	01CE	ED
Symmetric Algorithm Decipher - Secure AES keys	CSNBSAD / CSNESAD, CSNBSAD1 / CSNESAD1	012B	ED
Symmetric Algorithm Encipher – Allow A28MACGN and A28MACVR	CSNBSAE / CSNESAE	03B2	ED
Symmetric Algorithm Encipher - Allow A28OWFCL	CSNBSAE / CSNESAE	03B4	ED
Symmetric Algorithm Encipher – Allow A28OWFEC	CSNBSAE / CSNESAE	03B3	ED
Symmetric Algorithm Encipher - Galois/Counter mode AES	CSNBSAE / CSNESAE, CSNBSAE1 / CSNESAE1	01CD	ED
Symmetric Algorithm Encipher - Secure AES keys	CSNBSAE / CSNESAE, CSNBSAE1 / CSNESAE1	012A	ED
Symmetric Key Export – AES, CKM-RAKW	CSNDSYX / CSNFSYX	03B8	DD
Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2	CSNDSYX / CSNFSYX, CSNDSXD / CSNFSXD	0130	ED
Symmetric Key Export - AES, PKOAEP2	CSNDSYX / CSNFSYX	00FC	ED
Symmetric Key Export - AES, ZERO-PAD	CSNDSYX / CSNFSYX	0131	ED
Symmetric Key Export - AESKW	CSNDSYX / CSNFSYX	0327	ED
Symmetric Key Export - AESKWCV	CSNDSYX / CSNFSYX	02B3	ED
Symmetric Key Export - DES, PKCS-1.2	CSNDSYX / CSNFSYX, CSNDSXD / CSNFSXD	0105	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
Symmetric Key Export - DES, ZERO-PAD	CSNDSYX / CSNFSYX	023E	ED
Symmetric Key Export - HMAC, PKOAEP2	CSNDSYX / CSNFSYX	00F5	ED
Symmetric Key Export with Data	CSNDSXD / CSNFSXD	02B5	DD
Symmetric Key Export with Data - Special	CSNDSXD / CSNFSXD	02B6	DD
Symmetric Key Generate - AES, PKCSOAEP, PKCS-1.2	CSNDSYG / CSNFSYG	012C	ED
Symmetric Key Generate - AES, ZERO-PAD	CSNDSYG / CSNFSYG	012D	ED
Symmetric Key Generate - Allow wrapping override keywords	CSNDSYG / CSNFSYG	013E	ED
Symmetric Key Generate - DES, PKA92	CSNDSYG / CSNFSYG	010D	ED
Symmetric Key Generate - DES, PKCS-1.2	CSNDSYG / CSNFSYG	023F	ED
Symmetric Key Generate - DES, ZERO-PAD	CSNDSYG / CSNFSYG	023C	ED
Symmetric Key Import2 - AES,PKOAEP2	CSNDSYI2 / CSNFSYI2	00FD	ED
Symmetric Key Import2 - AESKW	CSNDSYI2 / CSNFSYI2	0329	ED
Symmetric Key Import2 - AESKWCV	CSNDSYI2 / CSNFSYI2	02B4	ED
Symmetric Key Import2 - Allow wrapping override keywords	CSNDSYI2 / CSNFSYI2	02B9	ED
Symmetric Key Import2 - Disallow weak import	CSNDSYI / CSNFSYI, CSNDSYI2 / CSNFSYI2, CSNBUKD / CSNEUKD	032B	DD, SC
Symmetric Key Import2 - HMAC,PKOAEP2	CSNDSYI2 / CSNFSYI2	00F4	ED
Symmetric Key Import - AES, PKCSOAEP, PKCS-1.2	CSNDSYI / CSNFSYI	012E	ED
Symmetric Key Import - AES, ZERO-PAD	CSNDSYI / CSNFSYI	012F	ED
Symmetric Key Import - Allow wrapping override keywords	CSNDSYI / CSNFSYI	0144	ED
Symmetric Key Import - DES, PKA92 KEK	CSNDSYI / CSNFSYI	0235	ED
Symmetric Key Import - DES, PKCS-1.2	CSNDSYI / CSNFSYI	0106	ED
Symmetric Key Import - DES, ZERO-PAD	CSNDSYI / CSNFSYI	023D	ED
T31C - Permit TR-31 AES creation	CSNBT31C / CSNET31C	03C1	ED
T31C - Permit TR-31 DES creation	CSNBT31C / CSNET31C	03C2	ED
T31C - Permit TR-31 external key creation	CSNBT31C / CSNET31C	03C5	ED
T31C - Permit TR-31 HMAC creation	CSNBT31C / CSNET31C	03C3	ED
T31C - Permit TR-31 internal/external key pair creation	CSNBT31C / CSNET31C	03C6	ED
T31C - Permit TR-31 internal key creation	CSNBT31C / CSNET31C	03C4	ED
T31C - Permit TR-31 KB Version A creation	CSNBT31C / CSNET31C	03C7	ED
T31C - Permit TR-31 KB Version B creation	CSNBT31C / CSNET31C	03C8	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
T31C - Permit TR-31 KB Version C creation	CSNBT31C / CSNET31C	03C9	ED
T31C - Permit TR-31 KB Version D creation	CSNBT31C / CSNET31C	03CA	ED
T31I - Disallow Partial DES Key Import with CV in IBMC01 OB	CSNBT31I / CSNET31I	006F	DD
T31I - Permit AES 10 to KDKGENKY:KDKTYPEA	CSNBT31I / CSNET31I	0387	DD
T31I - Permit AES 11 to KDKGENKY:KDKTYPEB	CSNBT31I / CSNET31I	0388	DD
T31I - Permit AES K1/K4:D to AES IMPORTER:IMPTT31D+ VARDRV-D	CSNBT31I / CSNET31I	01E6	ED
T31I - Permit B0:X to AES DKYGENKY:DUKPT BDK	CSNBT31I / CSNET31I	017E	ED
T31I - Permit B1 to DES KEYGENKY:DUKPT	CSNBT31I / CSNET31I	03E8	ED
T31I - Permit B3 to DES DKYGENKY	CSNBT31I / CSNET31I	03E9	ED
T31I - Permit C0:G/C/V to DES MAC/MACVER:AMEX-CSC	CSNBT31I / CSNET31I	015B	DD
T31I - Permit C0:G/C/V to DES MAC/MACVER:CVVKEY-A	CSNBT31I / CSNET31I	015A	DD
T31I - Permit D0:E/D/B to AES CIPHER:ENC/DEC/ ENC+DEC	CSNBT31I / CSNET31I	01E0	ED
T31I - Permit D3 to CIPHER:XLATE	CSNBT31I / CSNET31I	03EA	ED
T31I - Permit DES 12 to DKYGENKY:DKYL0+DMPIN	CSNBT31I / CSNET31I	0389	DD
T31I - Permit E0:N/X to DES DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	016D	DD
T31I - Permit E0:N/X to DES DKYGENKY:DKYL0+DMV	CSNBT31I / CSNET31I	016E	DD
T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMAC	CSNBT31I / CSNET31I	016F	DD
T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMV	CSNBT31I / CSNET31I	0170	DD
T31I - Permit E0:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	CSNBT31I / CSNET31I	01E7	ED
T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	0172	DD
T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DMPIN	CSNBT31I / CSNET31I	0171	DD
T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DDATA	CSNBT31I / CSNET31I	0174	DD
T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DMPIN	CSNBT31I / CSNET31I	0173	DD
T31I - Permit E1:X to AES DKYGENKY:DKYL0/L1/L2+D-SECMSG+SMPIN	CSNBT31I / CSNET31I	01E8	ED
T31I - Permit E2:N/X to DES DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	0175	DD
T31I - Permit E2:N/X to DES DKYGENKY:DKYL1+DMAC	CSNBT31I / CSNET31I	0176	DD
T31I - Permit E2:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	CSNBT31I / CSNET31I	01E9	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
T31I - Permit E3:E/B to AES CIPHER:ENCRYPT/ENC+DEC	CSNBT31I / CSNET31I	01EB	ED
T31I - Permit E3:N/E/D/B/G/X to DES ENCIPHER	CSNBT31I / CSNET31I	0177	DD
T31I - Permit E3:X to AES DKYGENKY:D-CIPHER+ENC+DEC+CBC	CSNBT31I / CSNET31I	01EA	ED
T31I - Permit E4:N/B/X to DES DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	0178	ED
T31I - Permit E4:X to AES DKYGENKY:DKYL0/L1/L2+D-CIPHER+ENC+DEC	CSNBT31I / CSNET31I	01EC	ED
T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	017A	DD
T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DEXP	CSNBT31I / CSNET31I	017B	DD
T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	0179	DD
T31I - Permit E5:X to AES DKYGENKY:DKYL0/L1/L2/D-MAC+GEN+CMAC	CSNBT31I / CSNET31I	01ED	ED
T31I - Permit F0:N/X to DES DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	03EB	DD
T31I - Permit F0:N/X to DES DKYGENKY:DKYL0+DMV	CSNBT31I / CSNET31I	03EC	DD
T31I - Permit F0:X to AES DKYGENKY:DKYL0+D-MAC+GENERATE+CMAC	CSNBT31I / CSNET31I	0504	ED
T31I - Permit F1:N/E/D/B/X to DES DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	03EE	DD
T31I - Permit F1:N/E/D/B/X to DES DKYGENKY:DKYL0+DMPIN	CSNBT31I / CSNET31I	03ED	DD
T31I - Permit F1:X to AES DKYGENKY:DKYL0+D-SECMSG+SMPIN+ANY-USE	CSNBT31I / CSNET31I	0505	ED
T31I - Permit F2:N/X to DES DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	03EF	DD
T31I - Permit F2:X to AES DKYGENKY:DKYL0+D-MAC+GENERATE+CMAC	CSNBT31I / CSNET31I	0506	ED
T31I - Permit F3:E/B to AES CIPHER:ENCRYPT/ ENCRYPT+DECRYPT	CSNBT31I / CSNET31I	0508	ED
T31I - Permit F3:N/E/D/B/G/X to DES ENCIPHER	CSNBT31I / CSNET31I	0502	DD
T31I - Permit F3:X to AES DKYGENKY:D-CIPHER+ENCRYPT+DECRYPT+CBC	CSNBT31I / CSNET31I	0507	ED
T31I - Permit F4:N/B/X to DES DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	0503	ED
T31I - Permit F4:X to AES DKYGENKY:DKYL0+D-CIPHER+ENC+DEC+CBC	CSNBT31I / CSNET31I	0509	ED
T31I - Permit K0:B to DES EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	015E	DD
T31I - Permit K0:B to DES IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	015F	DD
T31I - Permit K0:D to AES IMPORTER	CSNBT31I / CSNET31I	01E4	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
T31I - Permit K0:D to DES IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	015D	DD
T31I - Permit K0:E to AES EXPORTER	CSNBT31I / CSNET31I	01E3	ED
T31I - Permit K0:E to DES EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	015C	DD
T31I - Permit K1/K4:B to DES EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	0162	DD
T31I - Permit K1/K4:B to DES IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	0163	DD
T31I - Permit K1/K4:D to DES IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	0161	DD
T31I - Permit K1/K4:E to AES EXPORTER:EXPTT31D+ VARDRV-D	CSNBT31I / CSNET31I	01E5	ED
T31I - Permit K1/K4:E to DES EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	0160	DD
T31I - Permit M0/M1/M3:G/C/V to DES MAC/ MACVER:ANY-MAC	CSNBT31I / CSNET31I	0164	ED
T31I - Permit M6:G/C/V to AES MAC:CMAC+GENONLY/GEN/VER	CSNBT31I / CSNET31I	01E1	ED
T31I - Permit M6 to DES MAC	CSNBT31I / CSNET31I	03F0	ED
T31I - Permit M7:G/V/C to HMAC MAC: GENERATE/VERIFY	CSNBT31I / CSNET31I	017D	ED
T31I - Permit override of default wrapping method	CSNBT31I / CSNET31I	0153	ED
T31I - Permit P0:D to DES IPINENC	CSNBT31I / CSNET31I	0166	ED
T31I - Permit P0:E/D to AES PINPROT:ENC/ DEC+CBC+ISO-4	CSNBT31I / CSNET31I	01E2	ED
T31I - Permit P0:E to DES OPINENC	CSNBT31I / CSNET31I	0165	ED
T31I - Permit V0:N/G/C to DES PINGEN:NO-SPEC NOOFFSET	CSNBT31I / CSNET31I	0167	DD
T31I - Permit V0:N/V to DES PINVER:NO-SPEC NOOFFSET	CSNBT31I / CSNET31I	0168	DD
T31I - Permit V0/V1/V2:N to DES PINGEN/PINVER	CSNBT31I / CSNET31I	017C	DD
T31I - Permit V1:N/G/C to DES PINGEN:IBM-PIN/IBM-PINO NOOFFSET	CSNBT31I / CSNET31I	0169	ED
T31I - Permit V1:N/V to DES PINVER:IBM-PIN/IBM-PINO NOOFFSET	CSNBT31I / CSNET31I	016A	ED
T31I - Permit V2:N/G/C to DES PINGEN:VISA-PVV	CSNBT31I / CSNET31I	016B	ED
T31I - Permit V2:N/V to DES PINVER:VISA-PVV	CSNBT31I / CSNET31I	016C	ED
T31I - Permit version A TR-31 key blocks	CSNBT31I / CSNET31I	0150	ED
T31I - Permit version B TR-31 key blocks	CSNBT31I / CSNET31I	0151	ED
T31I - Permit version C TR-31 key blocks	CSNBT31I / CSNET31I	0152	ED
T31I - Permit version D TR-31 key blocks	CSNBT31I / CSNET31I	0386	ED
T31X - Disallow Partial DES Key Export with CV in IBMC01 OB	CSNBT31X / CSNET31X	006E	DD

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
T31X - Permit AES CIPHER, DKYGENKY:D-ALL/DCIPHER to F3:E/B/X	CSNBT31X / CSNET31X	0500	ED
T31X - Permit AES CIPHER to D0:E/D/B	CSNBT31X / CSNET31X	01D0	ED
T31X - Permit AES CIPHER to E3/E/B,DKYGENKY:D-ALL/DCIP to E3:X	CSNBT31X / CSNET31X	01DC	ED
T31X - Permit AES DKYGENKY:D-ALL/DCIPHER to E1:X	CSNBT31X / CSNET31X	01DA	ED
T31X - Permit AES DKYGENKY:D-ALL/D-CIPHER to E4:X	CSNBT31X / CSNET31X	01DD	ED
T31X - Permit AES DKYGENKY:D-ALL/DCIPHER to F1:X	CSNBT31X / CSNET31X	03FE	ED
T31X - Permit AES DKYGENKY:D-ALL/DCIPHER to F4:X	CSNBT31X / CSNET31X	0501	ED
T31X - Permit AES DKYGENKY:D-ALL/DMAC to E0:X	CSNBT31X / CSNET31X	01D9	ED
T31X - Permit AES DKYGENKY:D-ALL/D-MAC to E2:X	CSNBT31X / CSNET31X	01DB	ED
T31X - Permit AES DKYGENKY:D-ALL/DMAC to F0:X	CSNBT31X / CSNET31X	03FD	ED
T31X - Permit AES DKYGENKY:D-ALL/DMAC to F2:X	CSNBT31X / CSNET31X	03FF	ED
T31X - Permit AES DKYGENKY:D-MAC to E5:X	CSNBT31X / CSNET31X	01DE	ED
T31X - Permit AES DKYGENKY: DUKPT BDK to B0:X	CSNBT31X / CSNET31X	01CF	ED
T31X - Permit AES EXPORTER to K0:E	CSNBT31X / CSNET31X	01D3	ED
T31X - Permit AES EXPORTER to K1:E	CSNBT31X / CSNET31X	01D4	ED
T31X - Permit AES EXPORTER to K4:E	CSNBT31X / CSNET31X	01D5	ED
T31X - Permit AES IMPORTER to K0:D	CSNBT31X / CSNET31X	01D6	ED
T31X - Permit AES IMPORTER to K1:D	CSNBT31X / CSNET31X	01D7	ED
T31X - Permit AES IMPORTER to K4:D	CSNBT31X / CSNET31X	01D8	ED
T31X - Permit AES KDKGENKY: KDKTYPEA to 11:X	CSNBT31X / CSNET31X	0383	DD
T31X - Permit AES KDKGENKY: KDKTYPEB to 10:X	CSNBT31X / CSNET31X	0384	DD
T31X - Permit AES MAC: CMAC to M6:G/C/V	CSNBT31X / CSNET31X	01D1	ED
T31X - Permit AES PINPROT to P0:E/D	CSNBT31X / CSNET31X	01D2	ED
T31X - Permit any CCA DES key if INCL-CV is specified	CSNBT31X / CSNET31X	0158	ED
T31X - Permit CIPHER:XLATE to D3	CSNBT31X / CSNET31X	03E1	ED
T31X - Permit DES DATA/DATAM/CIPHER/MAC/ENCIPHER to E3:N/G/E/X	CSNBT31X / CSNET31X	01A9	DD
T31X - Permit DES DATA/DATAM/DATAMV to C0:G/C/V	CSNBT31X / CSNET31X	0184	ED
T31X - Permit DES DATA/MAC/CIPHER/ENCIPHER to F3:N/G/E/X	CSNBT31X / CSNET31X	03FA	DD
T31X - Permit DES DATA to D0:E/D/B	CSNBT31X / CSNET31X	0186	ED
T31X - Permit DES DKYGENKY, AES KDKGENKY to B3	CSNBT31X / CSNET31X	03E0	ED
T31X - Permit DES DKYGENKY:DKYL0+DALL to E0:N/X	CSNBT31X / CSNET31X	019B	DD

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
T31X - Permit DES DKYGENKY:DKYL0+DALL to E1:N/X	CSNBT31X / CSNET31X	01A1	DD
T31X - Permit DES DKYGENKY:DKYL0+DALL to E2:N/X	CSNBT31X / CSNET31X	01A6	DD
T31X - Permit DES DKYGENKY:DKYL0+DALL to E4:N/X	CSNBT31X / CSNET31X	01AB	ED
T31X - Permit DES DKYGENKY:DKYL0+DALL to E5:N/X	CSNBT31X / CSNET31X	01AF	ED
T31X - Permit DES DKYGENKY:DKYL0+DALL to F0:N/X	CSNBT31X / CSNET31X	03E6	DD
T31X - Permit DES DKYGENKY:DKYL0+DALL to F1:N/X	CSNBT31X / CSNET31X	03F7	DD
T31X - Permit DES DKYGENKY:DKYL0+DALL to F2:N/X	CSNBT31X / CSNET31X	03F9	DD
T31X - Permit DES DKYGENKY:DKYL0+DALL to F4:N/X	CSNBT31X / CSNET31X	03FC	ED
T31X - Permit DES DKYGENKY:DKYL0+DDATA to E1:N/X	CSNBT31X / CSNET31X	019F	DD
T31X - Permit DES DKYGENKY:DKYL0+DDATA to E4:N/X	CSNBT31X / CSNET31X	01AA	ED
T31X - Permit DES DKYGENKY:DKYL0+DDATA to E5:N/X	CSNBT31X / CSNET31X	01AE	DD
T31X - Permit DES DKYGENKY:DKYL0+DDATA to F1:N/X	CSNBT31X / CSNET31X	03F5	DD
T31X - Permit DES DKYGENKY:DKYL0+DDATA to F4:N/X	CSNBT31X / CSNET31X	03FB	ED
T31X - Permit DES DKYGENKY:DKYL0+DEXP to E5:N/X	CSNBT31X / CSNET31X	01AC	DD
T31X - Permit DES DKYGENKY:DKYL0+DMAC to E0:N/X	CSNBT31X / CSNET31X	0199	DD
T31X - Permit DES DKYGENKY:DKYL0+DMAC to E2:N/X	CSNBT31X / CSNET31X	01A5	DD
T31X - Permit DES DKYGENKY:DKYL0+DMAC to E5:N/X	CSNBT31X / CSNET31X	01AD	DD
T31X - Permit DES DKYGENKY:DKYL0+DMAC to F0:N/X	CSNBT31X / CSNET31X	03E4	DD
T31X - Permit DES DKYGENKY:DKYL0+DMAC to F2:N/X	CSNBT31X / CSNET31X	03F8	DD
T31X - Permit DES DKYGENKY:DKYL0+DMPIN to 12	CSNBT31X / CSNET31X	0385	DD
T31X - Permit DES DKYGENKY:DKYL0+DMPIN to E1:N/X	CSNBT31X / CSNET31X	01A0	DD
T31X - Permit DES DKYGENKY:DKYL0+DMPIN to F1:N/X	CSNBT31X / CSNET31X	03F6	DD
T31X - Permit DES DKYGENKY:DKYL0+DMV to E0:N/X	CSNBT31X / CSNET31X	019A	DD
T31X - Permit DES DKYGENKY:DKYL0+DMV to F0:N/X	CSNBT31X / CSNET31X	03E5	DD
T31X - Permit DES DKYGENKY:DKYL1+DALL to E0:N/X	CSNBT31X / CSNET31X	019E	DD
T31X - Permit DES DKYGENKY:DKYL1+DALL to E1:N/X	CSNBT31X / CSNET31X	01A4	DD
T31X - Permit DES DKYGENKY:DKYL1+DALL to E2:N/X	CSNBT31X / CSNET31X	01A8	DD
T31X - Permit DES DKYGENKY:DKYL1+DDATA to E1:N/X	CSNBT31X / CSNET31X	01A2	DD
T31X - Permit DES DKYGENKY:DKYL1+DMAC to E0:N/X	CSNBT31X / CSNET31X	019C	DD
T31X - Permit DES DKYGENKY:DKYL1+DMAC to E2:N/X	CSNBT31X / CSNET31X	01A7	DD
T31X - Permit DES DKYGENKY:DKYL1+DMPIN to E1:N/X	CSNBT31X / CSNET31X	01A3	DD
T31X - Permit DES DKYGENKY:DKYL1+DMV to E0:N/X	CSNBT31X / CSNET31X	019D	DD
T31X - Permit DES ENCIPHER/DECIPHER/CIPHER to D0:E/D/B	CSNBT31X / CSNET31X	0185	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
T31X - Permit DES EXPORTER/OKEYXLAT to K0:E	CSNBT31X / CSNET31X	0187	DD
T31X - Permit DES EXPORTER/OKEYXLAT to K1/K4:E	CSNBT31X / CSNET31X	0189	DD
T31X - Permit DES IMPORTER/IKEYXLAT to K0:D	CSNBT31X / CSNET31X	0188	DD
T31X - Permit DES IMPORTER/IKEYXLAT to K1/K4:D	CSNBT31X / CSNET31X	018A	DD
T31X - Permit DES IPINENC to P0:D	CSNBT31X / CSNET31X	0192	ED
T31X - Permit DES KEYGENKY:DUKPT, AES DKYGENKY:DUKPT to B1	CSNBT31X / CSNET31X	03DF	ED
T31X - Permit DES KEYGENKY: DUKPT to B0:N/X	CSNBT31X / CSNET31X	0180	ED
T31X - Permit DES MAC/DATA/DATAM to M0:G/C	CSNBT31X / CSNET31X	018B	DD
T31X - Permit DES MAC/DATA/DATAM to M1:G/C	CSNBT31X / CSNET31X	018D	ED
T31X - Permit DES MAC/DATA/DATAM to M3:G/C	CSNBT31X / CSNET31X	018F	ED
T31X - Permit DES MAC/MACVER:AMEX-CSC to C0:G/C/V	CSNBT31X / CSNET31X	0181	DD
T31X - Permit DES MAC/MACVER: ANY-MAC to C0:G/C/V	CSNBT31X / CSNET31X	0183	ED
T31X - Permit DES MAC/MACVER: CVV-KEYA to C0:G/C/V	CSNBT31X / CSNET31X	0182	DD
T31X - Permit DES MAC to M6	CSNBT31X / CSNET31X	03E7	ED
T31X - Permit DES MACVER/DATA/DATAMV to M0:V	CSNBT31X / CSNET31X	018C	ED
T31X - Permit DES MACVER/DATA/DATAMV to M1:V	CSNBT31X / CSNET31X	018E	ED
T31X - Permit DES MACVER/DATA/DATAMV to M3:V	CSNBT31X / CSNET31X	0190	ED
T31X - Permit DES OPINENC/IPINENC to P0:B	CSNBT31X / CSNET31X	039E	ED
T31X - Permit DES OPINENC to P0:E	CSNBT31X / CSNET31X	0191	ED
T31X - Permit DES PINGEN:NO-SPEC/IBM-PIN/IBM-PINO to V1:N/V	CSNBT31X / CSNET31X	0196	ED
T31X - Permit DES PINGEN:NO-SPEC/VISA-PVV to V2:N/C	CSNBT31X / CSNET31X	0198	ED
T31X - Permit DES PINGEN:NO-SPEC to V0:N/C	CSNBT31X / CSNET31X	0194	DD
T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N	CSNBT31X / CSNET31X	01B0	DD
T31X - Permit DES PINVER:NO-SPEC/IBM-PIN/IBM-PINO to V1:N/V	CSNBT31X / CSNET31X	0195	ED
T31X - Permit DES PINVER:NO-SPEC/VISA-PVV to V2:N/V	CSNBT31X / CSNET31X	0197	ED
T31X - Permit DES PINVER:NO-SPEC to V0:N/V	CSNBT31X / CSNET31X	0193	DD
T31X - Permit EXPORTER to K0:B	CSNBT31X / CSNET31X	02AD	ED
T31X - Permit HMAC MAC to M7:G/V/C	CSNBT31X / CSNET31X	020D	ED
T31X - Permit IMPORTER to K0:B	CSNBT31X / CSNET31X	02AE	ED
T31X - Permit SECMSG:SMKEY to K0	CSNBT31X / CSNET31X	03E3	ED
T31X - Permit SECMSG:SMPIN to P0	CSNBT31X / CSNET31X	03E2	ED

Table 677. Access control points - Callable Services (continued)

Name	Callable service	Value (Hex)	Usage
T31X - Permit version A TR-31 key blocks	CSNBT31X / CSNET31X	014D	ED
T31X - Permit version B TR-31 key blocks	CSNBT31X / CSNET31X	014E	ED
T31X - Permit version C TR-31 key blocks	CSNBT31X / CSNET31X	014F	ED
T31X - Permit version D TR-31 key blocks	CSNBT31X / CSNET31X	0382	ED
TR-34 Bind-Begin	CSNDT34B / CSNFT34B	01F0	ED
TR-34 Bind-Begin - Allow BINDCR	CSNDT34B / CSNFT34B	01F1	ED
TR-34 Bind-Begin - Allow REBINDCR	CSNDT34B / CSNFT34B	01F3	ED
TR-34 Bind-Begin - Allow UNBINDCR	CSNDT34B / CSNFT34B	01F2	ED
TR-34 Bind-Complete	CSNDT34C / CSNFT34C	01F4	ED
TR-34 Bind-Complete - Allow BINDKRDC	CSNDT34C / CSNFT34C	01F5	ED
TR-34 Bind-Complete - Allow BINDRV	CSNDT34C / CSNFT34C	01F6	ED
TR-34 Bind-Complete - Allow REBINDRV	CSNDT34C / CSNFT34C	01F8	ED
TR-34 Bind-Complete - Allow UNBINDRV	CSNDT34C / CSNFT34C	01F7	ED
TR-34 Key Distribution	CSNDT34D / CSNFT34D	01F9	ED
TR-34 Key Distribution - Allow 1PASSCRE	CSNDT34D / CSNFT34D	01FB	ED
TR-34 Key Distribution - Allow 2PASSCRE	CSNDT34D / CSNFT34D	01FA	ED
TR-34 Key Distribution - Permit AES EXPORTER to K0	CSNDT34D / CSNFT34D	0244	ED
TR-34 Key Distribution - Permit AES EXPORTER to K1	CSNDT34D / CSNFT34D	0245	ED
TR-34 Key Distribution - Permit AES IMPORTER to K0	CSNDT34D / CSNFT34D	0246	ED
TR-34 Key Distribution - Permit AES IMPORTER to K1	CSNDT34D / CSNFT34D	0247	ED
TR-34 Key Distribution - Permit DES EXPORTER to K0 or K1	CSNDT34D / CSNFT34D	0242	ED
TR-34 Key Distribution - Permit DES IMPORTER to K0 or K1	CSNDT34D / CSNFT34D	0243	ED
TR-34 Key Receive	CSNDT34R / CSNFT34R	01FC	ED
TR-34 Key Receive - Allow 1PASSRCV	CSNDT34R / CSNFT34R	01FE	ED
TR-34 Key Receive - Allow 2PASSRCV	CSNDT34R / CSNFT34R	01FD	ED
TR-34 Key Receive - Allow wrapping override keywords	CSNDT34R / CSNFT34R	01DF	ED
TR-34 Key Receive - Permit AES EXPORTER	CSNDT34R / CSNFT34R	024A	ED
TR-34 Key Receive - Permit AES EXPORTER with EXPTT31D	CSNDT34R / CSNFT34R	024C	ED
TR-34 Key Receive - Permit AES IMPORTER	CSNDT34R / CSNFT34R	024B	ED
TR-34 Key Receive - Permit AES IMPORTER with IMPTT31D	CSNDT34R / CSNFT34R	024D	ED
TR-34 Key Receive - Permit DES EXPORTER	CSNDT34R / CSNFT34R	0248	ED

Table 677. Access control points - Callable Services (continued)			
Name	**Callable service**	**Value (Hex)**	**Usage**
TR-34 Key Receive - Permit DES IMPORTER	CSNDT34R / CSNFT34R	0249	ED
Transaction Validation - Generate	CSNBTRV / CSNETRV	0291	ED
Transaction Validation - Verify CSC-3	CSNBTRV / CSNETRV	0292	ED
Transaction Validation - Verify CSC-4	CSNBTRV / CSNETRV	0293	ED
Transaction Validation - Verify CSC-5	CSNBTRV / CSNETRV	0294	ED
Trusted Block Create - Activate an inactive block	CSNDTBC / CSNFTBC	0310	ED
Trusted Block Create - Create Block in inactive form	CSNDTBC / CSNFTBC	030F	ED
Trusted Block Create - Disallow triple-length MAC key	CSNDTBC / CSNFTBC	032E	DD, SC
Unique Key Derive	CSNBUKD / CSNEUKD	01C8	ED
Unique Key Derive - Allow PIN-DATA processing	CSNBUKD / CSNEUKD	01C9	DD
Unique Key Derive - K3IPEK	CSNBUKD / CSNEUKD	0335	DD
Unique Key Derive - Override default wrapping	CSNBUKD / CSNEUKD	01CA	ED
VISA CVV Generate	CSNBCSG / CSNECSG	00DF	ED
VISA CVV Verify	CSNBCSV / CSNECSV	00E0	ED

There are relationships between certain access control points. A controlling access control point is required to be enabled before subordinate access control points can enabled. The TKE workstation will enable the controlling access control point when a subordinate access control point is enabled.

- To use Data Key Export - Unrestricted, the Data Key Export access control point must be enabled.
- To use Data Key Import - Unrestricted, the Data Key Import access control point must be enabled.
- Diversified Key Generate - single length or same halves requires either Diversified Key Generate - TDES-ENC or Diversified Key Generate - TDES-DEC be enabled.
- To use Key Export - Unrestricted, the Key Export access control point must be enabled.
- To use Key Import - Unrestricted, the Key Import access control point must be enabled.
- To use Key Part Import - Unrestricted, the Key Part Import - First key part and Key Part Import - Middle and Final access control points must be enabled.
- To use T31X - Permit PINGEN/PINVER to V0/V1/V2:N, the TR31 Export - Permit version A TR-31 key blocks access control point must be enabled.
- To use Unique Key Derive - Allow PIN-DATA processing or Unique Key Derive - Override default wrapping access control points, Unique Key Derive access control point must be enabled.
- To use SET Block Decompose - PIN ext IPINENC or PIN ex OPINENC, the SET Block Decompose access control point must be enabled.
- To use PKA Key Generate - Permit Regeneration Data, the PKA Key Generate access control point must be enabled.
- To use PKA Key Generate - Permit Regeneration Data Retain, the PKA Key Generate and PKA Key Generate - Clone access control points must be enabled.
- To use PKA Key Generate - Clear or PKA Key Generate - Clone, the PKA Key Generate access control point must be enabled.
- To use any of the following access control points, the ECC Diffie-Hellman access control point must be enabled:

- ECC Diffie-Hellman - Allow PASSTHRU
- ECC Diffie-Hellman - Allow key wrap override
- ECC Diffie-Hellman - Allow Prime Curve 192
- ECC Diffie-Hellman - Allow Prime Curve 224
- ECC Diffie-Hellman - Allow Prime Curve 256
- ECC Diffie-Hellman - Allow Prime Curve 384
- ECC Diffie-Hellman - Allow Prime Curve 521
- ECC Diffie-Hellman - Allow BP Curve 160
- ECC Diffie-Hellman - Allow BP Curve 192
- ECC Diffie-Hellman - Allow BP Curve 224
- ECC Diffie-Hellman - Allow BP Curve 256
- ECC Diffie-Hellman - Allow BP Curve 320
- ECC Diffie-Hellman - Allow BP Curve 384
- ECC Diffie-Hellman - Allow BP Curve 512
- ECC Diffie-Hellman - Prohibit weak key generate

Appendix H. Impact of compliance mode on callable services

When compliant-tagged key tokens are used, the request is processed according to the compliance mode in effect. A subset of callable services support compliant-tagged key tokens. Any attempt to use a compliant-tagged key token in a cryptographic operation within a service that does not accept compliant-tagged key tokens results in a failure.

PCI-HSM 2016 compliance mode

Table 678 on page 1695 shows a list of callable services that are not compliant with PCI-HSM 2016 compliance mode. For brevity, only the 31-bit callable services are listed.

Table 678. Callable services not compliant with PCI-HSM 2016

Callable service	Name	Possible compliant alternative or alternatives
CSNBCKC	CVV Key Combine	Create double-length keys instead of single-length keys. This eliminates a need for the service. **Note:** Though this service does not accept or create compliant-tagged key tokens, the key tokens that are created by this service can be converted to compliant-tagged key tokens.
CSNBCKI	Clear Key Import	1. Use the TKE workstation to create a compliant-tagged key token from key parts. or 2. Use the Key Generate (CSNBKGN) callable service to create a compliant-tagged key token with a random key value.
CSNBCKM	Multiple Clear Key Import	1. Use the TKE workstation to create a compliant-tagged key token from key parts. or 2. Use the Key Generate (CSNBKGN) callable service to create a compliant-tagged key token with a random key value.

Table 678. Callable services not compliant with PCI-HSM 2016 (continued)		
Callable service	**Name**	**Possible compliant alternative or alternatives**
CSNBCVE	Cryptographic Variable Encipher	Create double-length keys instead of single-length keys. This eliminates a need for the service.
CSNBCVT	Control Vector Translate	1. Recreate the source key token with the wanted control vector. or 2. Wrap the source key such that the wanted attributes are bound to the key. The Key Import/Export (CSNBKIM/ CSNBKEX) or TR-31 Import/Export (CSNBT31I/ CSNBT31X) callable services can be used.
CSNBDKM	Data Key Import	DATA keys are not compliant. Create compliant-tagged CIPHER or MAC keys that can be imported by using the Key Import (CSNBKIM) or TR-31 Import (CSNBT31I) callable services.
CSNBDKX	Data Key Export	DATA keys are not compliant. Create compliant-tagged CIPHER or MAC keys that can be exported by using the Key Export (CSNBKEX) or TR-31 Translate (CSNBT31X) callable services.
CSNBKET	Key Encryption Translate	None.
CSNBKPI	Key Part Import	Use the TKE workstation to create a compliant-tagged key token from key parts.
CSNBKPI2	Key Part Import2	Use the TKE workstation to create a compliant-tagged key token from key parts.
CSNBSKI	Secure Key Import	1. Use the TKE workstation to create a compliant-tagged key token from key parts. or 2. Use the Key Generate (CSNBKGN) callable service to create a compliant-tagged key token with a random key value.

Table 678. Callable services not compliant with PCI-HSM 2016 (continued)

Callable service	Name	Possible compliant alternative or alternatives
CSNBSKI2	Secure Key Import2	1. Use the TKE workstation to create a compliant-tagged key token from key parts. or 2. Use the Key Generate (CSNBKGN) callable service to create a compliant-tagged key token with a random key value.
CSNBSKM	Multiple Secure Key Import	1. Use the TKE workstation to create a compliant-tagged key token from key parts. or 2. Use the Key Generate (CSNBKGN) callable service to create a compliant-tagged key token with a random key value.
CSNBSKY	Secure Messaging for Keys	None.
CSNDPKX	PKA Public Key Extract	Use the CSNDPIC service to create a certificate signing request from a compliant-tagged private key token. Create a certificate from the CSR and use that in place of a public key token.
CSNDRKX	Remote Key Export	Use the TR34 and TR31 set of services to exchange compliant-tagged keys with a remote device.
CSNDSBC	SET Block Compose	None.
CSNDSBD	SET Block Decompose	None.
CSNDSXD	Symmetric Key Export with Data	The TR-31 Translate (CSNBT31X) callable service can be used to export a compliant-tagged symmetric key token with data.
CSNDTBC	Trusted Block Create	Use the TR34 and TR31 set of services to exchange compliant-tagged keys with a remote device.

In addition to the callable services above, certain restrictions are placed on compliant-tagged key tokens regardless of the service.

DES Key Tokens and Key Blocks

- Only internal keys may be compliant-tagged.

- Single-length keys cannot be compliant-tagged.
- Key Encrypting Keys (including master keys) must be at least as strong as the keys they protect. Therefore, a KEK with replicated key halves cannot wrap a key with unique key halves.
- Compliant-tagged key tokens are restricted in terms of the verification patterns that can be calculated. Only the ENC-ZERO and CMACZERO verification patterns are allowed. See the Key Test (CSNBKYT/CSNEKYT), Key Test Extended (CSNBKYTX/CSNEKYTX), and Key Test2 (CSNBKYT2/CSNEKYT2) callable services for more detail. In addition, for the Key Test and Key Extended services, the KEY-ENCD keyword is required.
- Compliant-tagged keys are restricted from performing certain PIN block translation operations. For more information, see Table 681 on page 1703.
- Generally, compliant-tagged keys cannot be used with non-compliant-tagged keys. The exceptions are:
  - The Cipher Text Translate2 (CSNBCTT2/CSNDCTT2) callable service.
  - Compliant-tagged keys may be used with an X.509 certificate which is compliant.
- Compliant-tagged keys cannot be used with the following RSA-encrypted key formats: PKCSAOEP, PKCS-1.2, ZEROPAD.

DES CCA Key Tokens

- Only the enhanced wrapping method can be used.
- Default DATA key tokens (including zero CV DATA key tokens) cannot be compliant-tagged. They are multi-use keys capable of performing both encipherment and MACing. One alternative is to upgrade to CIPHER and/or MAC keys. Currently, triple-length DATA keys with unique key parts have no migration path to becoming compliant-tagged without downgrading the key to a double-length key.
- NOCV-KEKs cannot be compliant-tagged.
- Compliant-tagged key tokens are defined by two features of the key token:
  1. The compliant-tag bit in the key attributes, which is the CV for DES fixed-length key tokens, and
  2. The key derivation function (KDF) value which indicates what generation of compliance is applicable.

  The KDF value appears at the end of the truncated Master Key Verification Pattern (MKVP) section. For additional information, see Appendix B, “Key token formats,” on page 1501.

  Key tokens with the compliant-tag bit on in the control vector and a key derivation function (KDF) of less than X'03' are no longer considered compliant-tagged. They are referred to as DES KDF 01 or 02 tokens throughout the ICSF publications. They were either created on a Crypto Express adapter which does not have the May 2021 or later licensed internal code (LIC) or by the Diversified Key Generate (CSNBDKG and CSNEDKG) or Unique Key Derive (CSNBUKD and CSNEUKD) services using an input DES KDF 01 or 02 token. A DES KDF 01 key token can only be used with other DES KDF 01 tokens. Beginning with the May 2021or later licensed internal code (LIC), a DES KDF 02 token can only be used with other DES KDF 02 key tokens. The only exception to this is a DES KDF 01 or 02 key token that can also be used with an X.509 certificate or with any other key token in the Cipher Text Translate2 (CSNBCTT2and CSNECTT2) callable service. It is recommended that DES KDF 01 and 02 tokens be migrated using the Key Translate2 (CSNBKTR2 ad CSNEKTR2) service with the COMP-TAG keyword. The output will be a key token with the compliant-tag bit set in the control vector and a KDF of X'03' or greater. Only DES tokens with a KDF of X'03' or greater are referred to as compliant-tagged key tokens. Key tokens without the compliant-tag bit set or DES KDF 01 or 02 tokens are referred to as non-compliant-tagged tokens and this is reflected in the ICSF output (ICSF displays as well as SMF records do not show DES KDF 01 or 02 tokens as compliant-tagged). The CSFCMPLC, CSFCMPCC, and CSFCMPTC samples may help with identifying DES KDF 01 or 02 tokens and migrating them. Though DES KDF 01 or 02 tokens are not compliant-tagged, a coprocessor in compliance mode is required to use them.

DES TR-31 Key Blocks

- Key blocks with exportability S cannot be compliant-tagged.

AES Key Tokens and Key Blocks

- Key Encrypting Keys (including master keys) must be at least as strong as the keys they protect.
- Compliant-tagged key tokens are restricted in terms of the verification patterns that can be calculated. Only the CMACZERO verification pattern is allowed.
- Generally, compliant-tagged key tokens cannot be used with non-compliant-tagged key tokens. The exceptions are:
  - The Cipher Text Translate2 (CSNBCTT2/CSNDCTT2) callable service.
  - Compliant-tagged key tokens may be used with an X.509 certificate which is compliant.
- Compliant-tagged key tokens are restricted from performing certain PIN block translation operations. For more information, see Table 681 on page 1703.
- Compliant-tagged keys cannot be used with the following RSA-encrypted key formats: PKCSAOEP, PKCS-1.2, ZEROPAD.

AES CCA Key Tokens

- Only internal version 05 key tokens may become compliant-tagged [AES version 04 DATA key tokens may be migrated to version 05 using the Key Translate2 (CSNBKTR2/CSNEKTR2) callable service].
- Compliant-tagged keys may not have a variable-length payload (V0PYLD). Only fixed-length payloads are allowed (V1PYLD).

AES TR-31 Key Blocks

- Key blocks with exportability S cannot be compliant-tagged.

RSA Key Tokens

- Only internal RSA key tokens with private key sections X'30' or X'31' and associated data versions X'04' or X'05' may become compliant-tagged. This corresponds to keywords RSAAESM2 and RSAAESC2 on the PKA Key Token Build (CSNDPKB/CSNFPKB) callable service. Other private key sections may be migrated using the PKA Key Translate (CSNDPKT/CSNFPKT) callable service.
- Compliant-tagged key tokens must be single usage, or in other words, may not perform both signatures and key encipherment. See "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 1147 for details.
- Compliant-tagged keys must have a minimum modulus size of 2048 bits.
- Key Encrypting Keys (including master keys) must be at least as strong as the keys they protect.
- Compliant-tagged key tokens cannot be used with non-compliant-tagged key tokens. The exception is compliant-tagged key tokens may be used with an X.509 certificate which is compliant.
- An RSA public key token may not be used to wrap a compliant-tagged symmetric key token.
- Compliant-tagged keys cannot be used with the following RSA-encrypted key formats: PKCSAOEP, PKCS-1.2, ZEROPAD.
- Compliant-tagged keys cannot use the SHA-1 hash method.

X.509 certificates (RSA)

An X.509 certificate cannot be compliant-tagged. However, it may be used with compliant-tagged keys if the X.509 certificate is considered compliant and PKI validation is active for the request.PKI validation is active if the request specifies or defaults to the PKI-CHK keyword. A certificate is considered compliant if it satisfies all of the following:

1. It is single usage (any combination of the following signature usages may be allowed by the certificate and it is still considered single usage: digitalSignature, nonRepudiation, keyCertSign, cRLSign).

2. It has a minimum modulus size of 2048 bits, and
3. The signature algorithm does not specify the SHA-1 hashing method.

One way to create a certificate with specific key usage attributes is to use the Public Infrastructure Create (CSNDPIC/CSNFPIC) callable service to create a certificate signing request (CSR) with the desired key usage attributes. This CSR can then be signed by a Certificate Authority (CA) to create a certificate with the desired key usage attributes. The RACDCERT GENCERT command may be used to create a certificate from a CSR.

When used with a compliant-tagged key, certificates:

- Must be at least as strong as the keys they protect.
- Cannot use the following RSA-encrypted key formats: PKCSAOEP, PKCS-1.2, ZEROPAD.
- Cannot use the SHA-1 hash method.

Table 679 on page 1700 shows a list of services that are both compliant with PCI-HSM 2016 and accept key tokens which could become compliant-tagged, but currently do not support compliant-tagged key tokens.

Table 679. Callable services that do not support compliant-tagged key tokens

Callable service	**Name**
CSNDEDH	ECC Diffie-Hellman

Table 680 on page 1700 shows the list of services that support compliant-tagged key tokens in cryptographic operations when running in PCI-HSM 2016 compliance mode. See the required hardware table of each individual service description for more detail.

Table 680. Callable services that support compliant-tagged keys in cryptographic operations

Callable service	**Name**
CSNBAPG	Authentication Parameter Generate
CSNBCPA	Clear PIN Generate Alternate
CSNBCPE	Clear PIN Encrypt
CSNBCSG	VISA CVV Service Generate
CSNBCSV	VISA CVV Service Verify
CSNBCTT2	Cipher Text Translate2
CSNBDCM	Derive ICC MK
CSNBDCU2	DK PRW Card Number Update2
CSNBDDK	Diversify Directed Key
CSNBDDPG	DK Deterministic PIN Generate
CSNBDEC	Decipher
CSNBDKG	Diversified Key Generate
CSNBDKG2	Diversified Key Generate2
CSNBDMP	DK Migrate PIN
CSNBDPC	DK PIN Change
CSNBDPCG	DK PRW CMAC Generate
CSNBDPMT	DK PAN Modify in Transaction

Table 680. Callable services that support compliant-tagged keys in cryptographic operations (continued)	
Callable service	**Name**
CSNBDPNU	DK PRW Card Number Update
CSNBDPT	DK PAN Translate
CSNBDPV	DK PIN Verify
CSNBDRG2	DK Random PIN Generate2
CSNBDRP	DK Regenerate PRW
CSNBDRPG	DK Random PIN Generate
CSNBDSK	Derive Session Key
CSNBEAC	EMV Transaction (ARQC/ARPC) Service
CSNBENC	Encipher
CSNBEPG	Encrypted PIN Generate
CSNBESC	EMV Scripting Service
CSNBEVF	EMV Verification Functions
CSNBFLD	Field Level Decipher
CSNBFLE	Field Level Encipher
CSNBFPED	FPE Decipher
CSNBFPEE	FPE Encipher
CSNBFPET	FPE Translate
CSNBGIM	Generate Issuer MK
CSNBKEX	Key Export
CSNBKGN	Key Generate
CSNBKGN2	Key Generate2
CSNBKIM	Key Import
CSNBKRR2	CKDS Key Record Read2
CSNBKTR	Key Translate
CSNBKTR2	Key Translate2
CSNBKYT	Key Test
CSNBKYT2	Key Test2
CSNBKYTX	Key Test Extended
CSNBMGN	MAC Generate
CSNBMGN2	MAC Generate2
CSNBMVR	MAC Verify
CSNBMVR2	MAC Verify2
CSNBPCU	PIN Change/Unblock
CSNBPEX	Prohibit Export

Table 680. Callable services that support compliant-tagged keys in cryptographic operations (continued)	
Callable service	**Name**
CSNBPEXX	Prohibit Export Extended
CSNBPFO	Recover PIN from Offset
CSNBPGN	Clear PIN Generate
CSNBPTR	Encrypted PIN Translate
CSNBPTR2	Encrypted PIN Translate2
CSNBPTRE	Encrypted PIN Translate Enhanced
CSNBPVR	Encrypted PIN Verify
CSNBRKA	Restrict Key Attribute
CSNBSAD	Symmetric Algorithm Decipher
CSNBSAE	Symmetric Algorithm Encipher
CSNBSPN	Secure Messaging for PINs
CSNBSYD	Symmetric Key Decipher
CSNBSYE	Symmetric Key Encipher
CSNBT31C	TR-31 Create
CSNBT31I	TR-31 Import
CSNBT31X	TR-31 Translate
CSNBTRV	Transaction Validation
CSNBUKD	Unique Key Derive
CSNDDSG	Digital Signature Generate
CSNDDSV	Digital Signature Verify
CSNDPIC	Public Infrastructure Certificate
CSNDPKD	PKA Decrypt
CSNDPKE	PKA Encrypt
CSNDPKG	PKA Key Generate
CSNDPKI	PKA Key Import
CSNDPKT	PKA Key Translate
CSNDSYG	Symmetric Key Generate
CSNDSYI	Symmetric Key Import
CSNDSYI2	Symmetric Key Import2
CSNDSYX	Symmetric Key Export
CSNDT34B	TR-34 Key Bind-Begin
CSNDT34D	TR-34 Key Distribution
CSNDT34R	TR-34 Key Receive

Table 681. Using compliant-tagged keys to translate between PIN block formats in cryptographic operations

Translate from:	**ECI-2 (4 digit PIN)**	**ECI-3 (4-6 digit PIN)**	**ISO-0, ANSI X9.8, VISA-1, VISA-4, and ECI-1**	**ISO-1 and ECI-4**	**ISO-2**	**ISO-3**	**VISA-2 (4-6 digit PIN)**	**VISA-3 (4-12 digit PIN)**	**3621 (4-12 digit PIN)**	**3624 (4-12 digit PIN)**	**4704-EPP (4-12 digit PIN)**
ECI-2	Y	Y	N	N	N	N	Y	Y	Y	Y	Y
ECI-3	If PIN is 4 digits.	Y	N	N	N	N	Y	Y	Y	Y	Y
ISO-0, ANSI X9.8, VISA-1, VISA-4, and ECI-1	N	N	If not changing PAN.	N	Y	Y	N	N	N	N	N
ISO-1 and ECI-4	N	N	Y	Y	Y	Y	N	N	N	N	N
ISO-2	N	N	N	N	Y	N	N	N	N	N	N
ISO-3	N	N	If not changing PAN.	N	Y	Y	N	N	N	N	N
VISA-2	If PIN is 4 digits.	Y	N	N	N	N	Y	Y	Y	Y	Y
VISA-3	If PIN is 4 digits.	If PIN is 4-6 digits.	N	N	N	N	If PIN is 4-6 digits.	Y	Y	Y	Y
3621	If PIN is 4 digits.	If PIN is 4-6 digits.	N	N	N	N	If PIN is 4-6 digits.	Y	Y	Y	Y
3624	If PIN is 4 digits.	If PIN is 4-6 digits.	N	N	N	N	If PIN is 4-6 digits.	Y	Y	Y	Y
4704-EPP	If PIN is 4 digits.	If PIN is 4-6 digits.	N	N	N	N	If PIN is 4-6 digits.	Y	Y	Y	Y

Appendix I. Resource names for CCA and ICSF entry points

Table 682. Resource names for CCA and ICSF entry points

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
	31-bit	64-bit	31-bit	64-bit		
Authentication Parameter Generate	CSNBAPG	CSNEAPG	CSFAPG	CSFAPG6	CSFAPG	CSFAPG
Cipher Text Translate2	CSNBCTT2	CSNECTT2	CSFCTT2	CSFCTT26	CSFCTT2	CSFCTT2
Cipher Text Translate2	CSNBCTT3	CSNECTT3	CSFCTT3	CSFCTT36	CSFCTT3	CSFCTT3
CKDS Key Record Create	CSNBKRC	CSNEKRC	CSFKRC	CSFKRC6	CSFKRC	CSFKRC
CKDS Key Record Create2	CSNBKRC2	CSNEKRC2	CSFKRC2	CSFKRC26	CSFKRC2	CSFKRC2
CKDS Key Record Delete	CSNBKRD	CSNEKRD	CSFKRD	CSFKRD6	CSFKRD	CSFKRD
CKDS Key Record Read	CSNBKRR	CSNEKRR	CSFKRR	CSFKRR6	CSFKRR	CSFKRR
CKDS Key Record Read2	CSNBKRR2	CSNEKRR2	CSFKRR2	CSFKRR26	CSFKRR2	CSFKRR2
CKDS Key Record Write	CSNBKRW	CSNEKRW	CSFKRW	CSFKRW6	CSFKRW	CSFKRW
CKDS Key Record Write2	CSNBKRW2	CSNEKRW2	CSFKRW2	CSFKRW26	CSFKRW2	CSFKRW2
Clear Key Import	CSNBCKI	CSNECKI	CSFCKI	CSFCKI6	CSFCKI	CSFCKI
Clear PIN Encrypt	CSNBCPE	CSNECPE	CSFCPE	CSFCPE6	CSFCPE	CSFCPE
Clear PIN Generate	CSNBPGN	CSNEPGN	CSFPGN	CSFPGN6	CSFPGN	CSFPGN
Clear PIN Generate Alternate	CSNBCPA	CSNECPA	CSFCPA	CSFCPA6	CSFCPA	CSFCPA
Control Vector Generate	CSNBCVG	CSNECVG	CSFCVG	CSFCVG6	N/A	N/A
Control Vector Translate	CSNBCVT	CSNECVT	CSFCVT	CSFCVT6	CSFCVT	CSFCVT
Coordinated KDS Administration	N/A	N/A	CSFCRC	CSFCRC6	CSFCRC	N/A
Cryptographic Usage Statistic	N/A	N/A	CSFSTAT	CSFSTAT6	N/A	N/A
Cryptographic Variable Encipher	CSNBCVE	CSNECVE	CSFCVE	CSFCVE6	CSFCVE	CSFCVE

Table 682. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
CVV Key Combine	CSNBCKC	CSNECKC	CSFCKC	CSFCKC6	CSFCKC	CSFCKC
Data Key Export	CSNBDKX	CSNEDKX	CSFDKX	CSFDKX6	CSFDKX	CSFDKX
Data Key Import	CSNBDKM	CSNEDKM	CSFDKM	CSFDKM6	CSFDKM	CSFDKM
Decipher	CSNBDEC	CSNEDEC	CSFDEC	CSFDEC6	CSFDEC	CSFDEC
Decipher	CSNBDEC1	CSNEDEC1	CSFDEC1	CSFDEC16	CSFDEC1	CSFDEC1
Decode	CSNBDCO	CSNEDCO	CSFDCO	CSFDCO6	CSFDCO	CSFDCO
Derive ICC MK	CSNBDCM	CSNEDCM	CSFDCM	CSFDCM6	CSFDCM	CSFDCM
Derive Session Key	CSNBDSK	CSNEDSK	CSFDSK	CSFDSK6	CSFDSK	CSFDSK
Digital Signature Generate	CSNDDSG	CSNFDSG	CSFDSG	CSFDSG6	CSFDSG	CSFDSG
Digital Signature Verify	CSNDDSV	CSNFDSV	CSFDSV	CSFDSV6	CSFDSV	CSFDSV
Diversified Key Generate	CSNBDKG	CSNEDKG	CSFDKG	CSFDKG6	CSFDKG	CSFDKG
Diversified Key Generate2	CSNBDKG2	CSNEDKG2	CSFDKG2	CSFDKG26	CSFDKG2	CSFDKG2
Diversify Directed Key	CSNBDDK	CSNEDDK	CSFDDK	CSFDDK6	CSFDDK	CSFDDK
DK Deterministic PIN Generate	CSNBDDPG	CSNEDDPG	CSFDDPG	CSFDDPG6	CSFDDPG	CSFDDPG
DK Migrate PIN	CSNBDMP	CSNEDMP	CSFDMP	CSFDMP6	CSFDMP	CSFDMP
DK PAN Modify in Transaction	CSNBDPMT	CSNEDPMT	CSFDPMT	CSFDPMT6	CSFDPMT	CSFDPMT
DK PAN Translate	CSNBDPT	CSNEDPT	CSFDPT	CSFDPT6	CSFDPT	CSFDPT
DK PIN Change	CSNBDPC	CSNEDPC	CSFDPC	CSFDPC6	CSFDPC	CSFDPC
DK PIN Verify	CSNBDPV	CSNEDPV	CSFDPV	CSFDPV6	CSFDPV	CSFDPV
DK PRW Card Number Update	CSNBDPNU	CSNEDPNU	CSFDPNU	CSFDPNU6	CSFDPNU	CSFDPNU
DK PRW Card Number Update2	CSNBDCU2	CSNBECU2	CSFDCU2	CSFDCU26	CSFDCU2	CSFDCU2
DK PRW CMAC Generate	CSNBDPCG	CSNEDPCG	CSFDPCG	CSFDPCG6	CSFDPCG	CSFDPCG
DK Random PIN Generate	CSNBDRPG	CSNEDRPG	CSFDRPG	CSFDRPG6	CSFDRPG	CSFDRPG
DK Random PIN Generate2	CSNBDRG2	CSNBERG2	CSFDRG2	CSFDRG26	CSFDRG2	CSFDRG2
DK Regenerate PRW	CSNBDRP	CSNEDRP	CSFDRP	CSFDRP6	CSFDRP	CSFDRP
ECC Diffie-Hellman	CSNDEDH	CSNFEDH	CSFEDH	CSFEDH6	CSFEDH	CSFEDH

Table 682. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
EMV Scripting Service	CSNBESC	CSNEESC	CSFESC	CSFESC6	CSFESC	CSFESC
EMV Transaction (ARQC/ARPC) Service	CSNBEAC	CSNEEAC	CSFEAC	CSFEAC6	CSFEAC	CSFEAC
EMV Verification Functions	CSNBEVF	CSNEEVF	CSFEVF	CSFEVF6	CSFEVF	CSFEVF
Encipher	CSNBENC	CSNEENC	CSFENC	CSFENC6	CSFENC	CSFENC
Encipher	CSNBENC1	CSNEENC1	CSFENC1	CSFENC16	CSFENC1	CSFENC1
Encode	CSNBECO	CSNEECO	CSFECO	CSFECO6	CSFECO	CSFECO
Encrypted PIN Generate	CSNBEPG	CSNEEPG	CSFEPG	CSFEPG6	CSFEPG	CSFEPG
Encrypted PIN Translate	CSNBPTR	CSNEPTR	CSFPTR	CSFPTR6	CSFPTR	CSFPTR
Encrypted PIN Translate2	CSNBPTR2	CSNEPTR2	CSFPTR2	CSFPTR26	CSFPTR2	CSFPTR2
Encrypted PIN Translate Enhanced	CSNBPTRE	CSNEPTRE	CSFPTRE	CSFPTRE6	CSFPTRE	CSFPTRE
Encrypted PIN Verify	CSNBPVR	CSNEPVR	CSFPVR	CSFPVR6	CSFPVR	CSFPVR
Encrypted PIN Verify2	CSNBPVR2	CSNEPVR2	CSFPVR2	CSFPVR26	CSFPVR2	CSNBPVR2
Field Level Decipher	CSNBFLD	CSNEFLD	CSFFLD	CSFFLD6	N/A	N/A
Field Level Encipher	CSNBFLE	CSNEFLE	CSFFLE	CSFFLE6	N/A	N/A
Format Preserving Algorithms Decipher	CSNBFFXD	CSNEFFXD	CSFFFXD	CSFFFXD6	CSFFFXD	CSFFFXD
Format Preserving Algorithms Encipher	CSNBFFXE	CSNEFFXE	CSFFFXE	CSFFFXE6	CSFFFXE	CSFFFXE
Format Preserving Algorithms Translate	CSNBFFXT	CSNEFFXT	CSFFFXT	CSFFFXT6	CSFFFXT	CSFFFXT
FPE Decipher	CSNBFPED	CSNEFPED	CSFFPED	CSFFPED6	CSFFPED	CSFFPED
FPE Encipher	CSNBFPEE	CSNEFPEE	CSFFPEE	CSFFPEE6	CSFFPEE	CSFFPEE
FPE Translate	CSNBFPET	CSNEFPET	CSFFPET	CSFFPET6	CSFFPET	CSFFPET
Generate Issuer MK	CSNBGIM	CSNEGIM	CSFGIM	CSFGIM6	CSFGIM	CSFGIM
HMAC Generate	CSNBHMG	CSNEHMG	CSFHMG	CSFHMG6	CSFHMG	CSFHMG
HMAC Generate	CSNBHMG1	CSNEHMG1	CSFHMG1	CSFHMG16	CSFHMG1	CSFHMG1
HMAC Verify	CSNBHMV	CSNEHMV	CSFHMV	CSFHMV6	CSFHMV	CSFHMV
HMAC Verify	CSNBHMV1	CSNEHMV1	CSFHMV1	CSFHMV16	CSFHMV1	CSFHMV1
ICSF Multi-Purpose Service	N/A	N/A	CSFMPS	CSFMPS6	CSFMPS	CSFMPS
ICSF Query Algorithm	N/A	N/A	CSFIQA	CSFIQA6	CSFIQA	N/A

Table 682. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
ICSF Query Facility	N/A	N/A	CSFIQF	CSFIQF6	CSFIQF	N/A
ICSF Query Facility2	N/A	N/A	CSFIQF2	CSFIQF26	N/A	CSFIQF2
Key Data Set List	N/A	N/A	CSFKDSL	CSFKDSL6	CSFKDSL	CSFKDSL
Key Data Set Metadata Read	N/A	N/A	CSFKDMR	CSFKDMR6	CSFKDMR	CSFKDMR
Key Data Set Metadata Write	N/A	N/A	CSFKDMW	CSFKDMW6	CSFKDMW	CSFKDMW
Key Data Set Record Retrieve	N/A	N/A	CSFRRT	CSFRRT6	CSFRRT (see notes)	N/A
Key Data Set Update	N/A	N/A	CSFKDU	CSFKDU6	CSFKDU (see notes)	N/A
Key Encryption Translate	CSNBKET	CSNEKET	CSFKET	CSFKET6	CSFKET	CSFKET
Key Export	CSNBKEX	CSNEKEX	CSFKEX	CSFKEX6	CSFKEX	CSFKEX
Key Generate	CSNBKGN	CSNEKGN	CSFKGN	CSFKGN6	CSFKGN	CSFKGN
Key Generate2	CSNBKGN2	CSNEKGN2	CSFKGN2	CSFKGN26	CSFKGN2	CSFKGN2
Key Import	CSNBKIM	CSNEKIM	CSFKIM	CSFKIM6	CSFKIM	CSFKIM
Key Part Import	CSNBKPI	CSNEKPI	CSFKPI	CSFKPI6	CSFKPI	CSFKPI
Key Part Import2	CSNBKPI2	CSNEKPI2	CSFKPI2	CSFKPI26	CSFKPI2	CSFKPI2
Key Test	CSNBKYT	CSNEKYT	CSFKYT	CSFKYT6	CSFKYT	CSFKYT
Key Test2	CSNBKYT2	CSNEKYT2	CSFKYT2	CSFKYT26	CSFKYT2	CSFKYT2
Key Test Extended	CSNBKYTX	CSNEKYTX	CSFKYTX	CSFKYTX6	CSFKYTX	CSFKYTX
Key Token Build	CSNBKTB	CSNEKTB	CSFKTB	CSFKTB6	N/A	N/A
Key Token Build2	CSNBKTB2	CSNEKTB2	CSFKTB2	CSFKTB26	N/A	N/A
Key Token Wrap	N/A	N/A	CSFWRP	CSFWRP6	CSFWRP	N/A
Key Translate	CSNBKTR	CSNEKTR	CSFKTR	CSFKTR6	CSFKTR	CSFKTR
Key Translate2	CSNBKTR2	CSNEKTR2	CSFKTR2	CSFKTR26	CSFKTR2	CSFKTR2
MAC Generate	CSNBMGN	CSNEMGN	CSFMGN	CSFMGN6	CSFMGN	CSFMGN
MAC Generate	CSNBMGN1	CSNEMGN1	CSFMGN1	CSFMGN16	CSFMGN1	CSFMGN1
MAC Generate2	CSNBMGN2	CSNEMGN2	CSFMGN2	CSFMGN26	CSFMGN2	CSFMGN2
MAC Generate2	CSNBMGN3	CSNEMGN3	CSFMGN3	CSFMGN36	CSFMGN3	CSFMGN3
MAC Verify	CSNBMVR	CSNEMVR	CSFMVR	CSFMVR6	CSFMVR	CSFMVR
MAC Verify	CSNBMVR1	CSNEMVR1	CSFMVR1	CSFMVR16	CSFMVR1	CSFMVR1
MAC Verify2	CSNBMVR2	CSNEMVR2	CSFMVR2	CSFMVR26	CSFMVR2	CSFMVR2
MAC Verify2	CSNBMVR3	CSNEMVR3	CSFMVR3	CSFMVR36	CSFMVR3	CSFMVR3

Table 682. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
MDC Generate	CSNBMDG	CSNEMDG	CSFMDG	CSFMDG6	CSFMDG	CSFMDG
MDC Generate	CSNBMDG1	CSNEMDG1	CSFMDG1	CSFMDG16	CSFMDG1	CSFMDG1
Multiple Clear Key Import	CSNBCKM	CSNECKM	CSFCKM	CSFCKM6	CSFCKM	CSFCKM
Multiple Secure Key Import	CSNBSKM	CSNESKM	CSFSKM	CSFSKM6	CSFSKM	CSFSKM
One-Way Hash Generate	CSNBOWH	CSNEOWH	CSFOWH	CSFOWH6	CSFOWH	CSFOWH
One-Way Hash Generate	CSNBOWH1	CSNEOWH1	CSFOWH1	CSFOWH16	CSFOWH1	CSFOWH1
PCI Interface	N/A	N/A	CSFPCI	CSFPCI6	CSFPCI	CSFPCI
PIN Change/Unblock	CSNBPCU	CSNEPCU	CSFPCU	CSFPCU6	CSFPCU	CSFPCU
PKA Decrypt	CSNDPKD	CSNFPKD	CSFPKD	CSFPKD6	CSFPKD	CSFPKD
PKA Encrypt	CSNDPKE	CSNFPKE	CSFPKE	CSFPKE6	CSFPKE	CSFPKE
PKA Key Generate	CSNDPKG	CSNFPKG	CSFPKG	CSFPKG6	CSFPKG	CSFPKG
PKA Key Import	CSNDPKI	CSNFPKI	CSFPKI	CSFPKI6	CSFPKI	CSFPKI
PKA Key Token Build	CSNDPKB	CSNFPKB	CSFPKB	CSFPKB6	N/A	N/A
PKA Key Token Change	CSNDKTC	CSNFKTC	CSFPKTC	CSFPKTC6	CSFPKTC	CSFPKTC
PKA Key Translate	CSNDPKT	CSNFPKT	CSFPKT	CSFPKT6	CSFPKT	CSFPKT
PKA Public Key Extract	CSNDPKX	CSNFPKX	CSFPKX	CSFPKX6	CSFPKX	CSFPKX
PKCS #11 Derive Key	N/A	N/A	CSFPDVK	CSFPDVK6	CSF1DVK[1]	N/A
PKCS #11 Derive Multiple Keys	N/A	N/A	CSFPDMK	CSFPDMK6	CSF1DMK[1]	N/A
PKCS #11 Generate Keyed MAC	N/A	N/A	CSFPHMG	CSFPHMG6	CSF1HMG[1]	N/A
PKCS #11 Generate Key Pair	N/A	N/A	CSFPGKP	CSFPGKP6	CSF1GKP[1]	N/A
PKCS #11 Generate Secret Key	N/A	N/A	CSFPGSK	CSFPGSK6	CSF1GSK[1]	N/A
PKCS #11 Get Attribute Value	N/A	N/A	CSFPGAV	CSFPGAV6	CSF1GAV[1]	N/A
PKCS #11 One-Way Hash, Sign, or Verify	N/A	N/A	CSFPOWH	CSFPOWH6	CSFOWH	N/A
PKCS #11 Private Key Sign	N/A	N/A	CSFPPKS	CSFPPKS6	CSF1PKS[1]	N/A

Table 682. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
PKCS #11 Private Key Structure Decrypt	N/A	N/A	CSFPPD2	CSFPPD26	CSFPKD	N/A
PKCS #11 Private Key Structure Sign	N/A	N/A	CSFPPS2	CSFPPS26	CSFDSG	N/A
PKCS #11 Pseudo-Random Function	N/A	N/A	CSFPPRF	CSFPPRF6	CSFRNG	N/A
PKCS #11 Public Key Structure Encrypt	N/A	N/A	CSFPPE2	CSFPPE26	CSFPKE	N/A
PKCS #11 Public Key Structure Verify	N/A	N/A	CSFPPV2	CSFPPV26	CSFDSV	N/A
PKCS #11 Public Key Verify	N/A	N/A	CSFPPKV	CSFPPKV6	CSF1PKV[1]	N/A
PKCS #11 Secret Key Decrypt	N/A	N/A	CSFPSKD	CSFPSKD6	CSF1SKD[1]	N/A
PKCS #11 Secret Key Encrypt	N/A	N/A	CSFPSKE	CSFPSKE6	CSF1SKE[1]	N/A
PKCS #11 Secret Key Reencrypt	N/A	N/A	CSFPSKR	CSFPSKR6	CSF1SKR[1]	N/A
PKCS #11 Set Attribute Value	N/A	N/A	CSFPSAV	CSFPSAV6	CSF1SAV[1]	N/A
PKCS #11 Token Record Create	N/A	N/A	CSFPTRC	CSFPTRC6	CSF1TRC[1]	N/A
PKCS #11 Token Record Delete	N/A	N/A	CSFPTRD	CSFPTRD6	CSF1TRD[1]	N/A
PKCS #11 Token Record List	N/A	N/A	CSFPTRL	CSFPTRL6	CSF1TRL[1]	N/A
PKCS #11 Unwrap Key	N/A	N/A	CSFPUWK	CSFPUWK6	CSF1UWK[1]	N/A
PKCS #11 Verify Keyed MAC	N/A	N/A	CSFPHMV	CSFPHMV6	CSF1HMV[1]	N/A
PKCS #11 Wrap Key	N/A	N/A	CSFPWPK	CSFPWPK6	CSF1WPK[1]	N/A
PKDS Key Record Create	CSNDKRC	CSNFKRC	CSFPKRC	CSFPKRC6	CSFPKRC	CSFPKRC
PKDS Key Record Delete	CSNDKRD	CSNFKRD	CSFPKRD	CSFPKRD6	CSFPKRD	CSFPKRD
PKDS Key Record Read	CSNDKRR	CSNFKRR	CSFPKRR	CSFPKRR6	CSFPKRR	CSFPKRR
PKDS Key Record Read2	CSNDKRR2	CSNFKRR2	CSFPRR2	CSFPRR26	CSFPRR2	CSFPRR2

Table 682. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
PKDS Key Record Write	CSNDKRW	CSNFKRW	CSFPKRW	CSFPKRW6	CSFPKRW	CSFPKRW
Prohibit Export	CSNBPEX	CSNEPEX	CSFPEX	CSFPEX6	CSFPEX	CSFPEX
Prohibit Export Extended	CSNBPEXX	CSNEPEXX	CSFPEXX	CSFPEXX6	CSFPEXX	CSFPEXX
Public Infrastructure Certificate	CSNDPIC	CSNFPIC	CSFPIC	CSFPIC6	CSFPIC	CSFPIC
Random Number Generate	CSNBRNG	CSNERNG	CSFRNG	CSFRNG6	CSFRNG	CSFRNG
Random Number Generate	CSNBRNGL	CSNERNGL	CSFRNGL	CSFRNGL6	CSFRNGL	CSFRNGL
Recover PIN from Offset	CSNBPFO	CSNEPFO	CSFPFO	CSFPFO6	CSFPFO	CSFPFO
Remote Key Export	CSNDRKX	CSNFRKX	CSFRKX	CSFRKX6	CSFRKX	CSFRKX
Restrict Key Attribute	CSNBRKA	CSNERKA	CSFRKA	CSFRKA6	CSFRKA	CSFRKA
Retained Key Delete	CSNDRKD	CSNFRKD	CSFRKD	CSFRKD6	CSFRKD	CSFRKD
Retained Key List	CSNDRKL	CSNFRKL	CSFRKL	CSFRKL6	CSFRKL	CSFRKL
SAF ACEE Selection	N/A	N/A	CSFACEE	CSFACEE6	N/A (see notes)	N/A (see notes)
Secure Key Import	CSNBSKI	CSNESKI	CSFSKI	CSFSKI6	CSFSKI	CSFSKI
Secure Key Import2	CSNBSKI2	CSNESKI2	CSFSKI2	CSFSKI26	CSFSKI2	CSFSKI2
Secure Messaging for Keys	CSNBSKY	CSNESKY	CSFSKY	CSFSKY6	CSFSKY	CSFSKY
Secure Messaging for PINs	CSNBSPN	CSNESPN	CSFSPN	CSFSPN6	CSFSPN	CSFSPN
SET Block Compose	CSNDSBC	CSNFSBC	CSFSBC	CSFSBC6	CSFSBC	CSFSBC
SET Block Decompose	CSNDSBD	CSNFSBD	CSFSBD	CSFSBD6	CSFSBD	CSFSBD
Symmetric Algorithm Decipher	CSNBSAD	CSNESAD	CSFSAD	CSFSAD6	CSFSAD	N/A
Symmetric Algorithm Decipher	CSNBSAD1	CSNESAD1	CSFSAD1	CSFSAD16	CSFSAD1	N/A
Symmetric Algorithm Encipher	CSNBSAE	CSNESAE	CSFSAE	CSFSAE6	CSFSAE	N/A
Symmetric Algorithm Encipher	CSNBSAE1	CSNESAE1	CSFSAE1	CSFSAE16	CSFSAE1	N/A
Symmetric Key Decipher	CSNBSYD	CSNESYD	CSFSYD	CSFSYD6	N/A	N/A

Table 682. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
Symmetric Key Decipher	CSNBSYD1	CSNESYD1	CSFSYD1	CSFSYD16	N/A	N/A
Symmetric Key Encipher	CSNBSYE	CSNESYE	CSFSYE	CSFSYE6	N/A	N/A
Symmetric Key Encipher	CSNBSYE1	CSNESYE1	CSFSYE1	CSFSYE16	N/A	N/A
Symmetric Key Export	CSNDSYX	CSNFSYX	CSFSYX	CSFSYX6	CSFSYX	CSFSYX
Symmetric Key Export with Data	CSNDSXD	CSNFSXD	CSFSXD	CSFSXD6	CSFSXD	CSFSXD
Symmetric Key Generate	CSNDSYG	CSNFSYG	CSFSYG	CSFSYG6	CSFSYG	CSFSYG
Symmetric Key Import	CSNDSYI	CSNFSYI	CSFSYI	CSFSYI6	CSFSYI	CSFSYI
Symmetric Key Import2	CSNDSYI2	CSNFSYI2	CSFSYI2	CSFSYI26	CSFSYI2	CSFSYI2
Symmetric MAC Generate	CSNBSMG	CSNESMG	CSFSMG	CSFSMG6	N/A	CSFSMG
Symmetric MAC Generate	CSNBSMG1	CSNESMG1	CSFSMG1	CSFSMG16	N/A	CSFSMG1
Symmetric MAC Verify	CSNBSMV	CSNESMV	CSFSMV	CSFSMV6	N/A	CSFSMV
Symmetric MAC Verify	CSNBSMV1	CSNESMV1	CSFSMV1	CSFSMV16	N/A	CSFSMV1
TR-31 Create	CSNBT31C	CSNET31C	CSFT31C	CSFT31C6	CSFT31C	CSFT31C
TR-31 Import	CSNBT31I	CSNET31I	CSFT31I	CSFT31I6	CSFT31I	CSFT31I
TR-31 Optional Data Build	CSNBT31O	CSNET31O	CSFT31O	CSFT31O6	N/A	N/A
TR-31 Optional Data Read	CSNBT31R	CSNET31R	CSFT31R	CSFT31R6	N/A	N/A
TR-31 Parse	CSNBT31P	CSNET31P	CSFT31P	CSFT31P6	N/A	N/A
TR-31 Translate	CSNBT31X	CSNET31X	CSFT31X	CSFT31X6	CSFT31X	CSFT31X
TR-34 Bind-Begin	CSNDT34B	CSNFT34B	CSFT34B	CSFT34B6	CSFT34B	CSFT34B
TR-34 Bind-Complete	CSNDT34C	CSNFT34C	CSFT34C	CSFT34C6	CSFT34C	CSFT34C
TR-34 Key Distribution	CSNDT34D	CSNFT34D	CSFT34D	CSFT34D6	CSFT34D	CSFT34D
TR-34 Key Receive	CSNDT34R	CSNFT34R	CSFT34R	CSFT34R6	CSFT34R	CSFT34R
Transaction Validation	CSNBTRV	CSNETRV	CSFTRV	CSFTRV6	CSFTRV	CSFTRV
Trusted Block Create	CSNDTBC	CSNFTBC	CSFTBC	CSFTBC6	CSFTBC	CSFTBC
Unique Key Derive	CSNBUKD	CSNEUKD	CSFUKD	CSFUKD6	CSFUKD	CSFUKD

Table 682. Resource names for CCA and ICSF entry points (continued)						
Descriptive service name	**CCA entry point names**		**ICSF entry point names**		**SAF resource name**	**Callable service exit name**
VISA CVV Service Generate	CSNBCSG	CSNECSG	CSFCSG	CSFCSG6	CSFCSG	CSFCSG
VISA CVV Service Verify	CSNBCSV	CSNECSV	CSFCSV	CSFCSV6	CSFCSV	CSFCSV

Notes:

- Key Data Set Update (CSFKDU and CSFKDU6) and Key Data Set Record Retrieve (CSFRRT and CSFRRT6) will only be granted access with an explicitly defined covering profile.
- SAF ACEE Selection (CSFACEE and CSFACEE6) does not have SAF checking or callable service exit support on its own. The service specified in the *service_name* parameter determines SAF checking and callable service exit capability.
- N/A is shown in a column when the callable service:
  - Does not have CCA entry points (CCA entry point names columns).
  - Does not call SAF to determine access to a CSFSERV resource (SAF resource name column).
  - Does not allow a callable service exit to be defined (Callable service exit name column).
- [1] CSF1*xxx* is just another name for the CSFP*xxx* service.

Appendix J. Cryptographic hardware engines and software

The following are details about the cryptographic hardware engines and software used by ICSF:

CEX*n*C adapter:

- CCA coprocessor.
- Firmware.
- Executes within the adapter.

CEX*n*P adapter:

- EP11 coprocessor.
- Firmware.
- Executes within the adapter.

CEX*n*A adapter:

- Accelerator.
- Firmware.
- Executes within the adapter.

CP Assist for Cryptographic Functions (CPACF):

- Cryptographic hardware engines.
- Executes on System z CPU.

Software:

- ICSF code.
- Executes on System z CPU.

IBM Common Cryptographic Architecture (CCA)

This topic describes the details about:

- Symmetric key services: “Symmetric key algorithms and processing” on page 1715.
- Asymmetric key services: “Asymmetric key algorithms and processing” on page 1716.
- Hashing services: “Hashing algorithms and processing” on page 1716.

Symmetric key algorithms and processing

- A clear key is not wrapped by another key.
- A secure key is wrapped by a key.
- An operational key is a secure key wrapped by a CCA master key.
- A protected key is a secure key wrapped by the CPACF master key (HPSK).

DES and TDES

- Data-encryption: Clear, operational, and CPACF protected keys.
- Message authentication: Operational keys.
- Key management: Operational keys.
- Financial services: Operational keys.

AES

- Data-encryption: Clear, operational, and CPACF protected keys.
- Message authentication: Operational keys.
- Key management: Operational keys.
- Financial services: Operational keys.

HMAC

- Message authentication: Clear and operational keys.

Asymmetric key algorithms and processing

- A public key is always in the clear.
- A clear private key is not wrapped by any key.
- An operational private key of an asymmetric key is wrapped by a CCA master key.

RSA

- Signature: Clear and operational keys.
- Key management: Clear and operational keys.

EC

- Signature: Clear, operational, and protected keys.

Dilithium

- Signature: Clear and operational keys.

Diffie-Hellman

- Key management: Clear and operational keys.

Hashing algorithms and processing

SHA-1
CPACF only.

SHA-2
CPACF only.

SHA-3/SHAKE
CPACF only.

MD5
Software only.

RIPEMD-160
Software only.

Hardware support

Table 683 on page 1717 shows which cryptographic hardware engine or software is used for the processing of the algorithm.

P
Primary engine.

O
Optional engine. Used if available when algorithm and rule array keywords permit its use.

1
Coprocessor required to export the operational key as a protected key.

Table 683. Cryptographic functions used by ICSF				
Algorithm	**CPACF**	**CEX*n*C**	**CEX*n*A**	**Software**
DES/3DES/AES – Clear key	P			
DES/3DES/AES – Secure key		P		
DES/3DES/AES – Protected key	P	1		
HMAC – Clear key	P			
HMAC – Secure key		P		
SHA-1/SHA-2/SHA-3	P			
MD5/RIPEMD-160				P
Dilithium - Clear and secure private key		P		
ECC - Clear private key	O	P		
ECC - Secure private key		P		
ECC - Protected private key	P	1		
RSA – Clear private key		P	O	
RSA – Secure private key		P		
Dilithium - Public key		P		
ECC – Public key	O	P		
RSA – Public key		P	O	
DH		P		

Appendix K. AES-DUKPT reference information

AES-DUKPT derivation data

The following data structure is used to specify input parameters for both initial derivation keys and working keys. The initial terminal DUKPT key or initial key is used together with the transaction counter to derive the transaction key, which is also called the working key. Once the transaction key has been derived, the terminal does not preserve any information that could be used to derive the transaction key after the transaction has been completed.

In the following table, the derivation data is specified. For the most part, initial terminal key value ranges are a subset of the overall value ranges. One exception to that rule is the value for the key usage indicator field, X'8001', which can only be used when deriving an initial terminal key.

Table 684. AES-DUKPT derivation data

Offset	Length of field (bytes)	Field name	Description
0	1	Version	Version ID of this table structure. Allowed value: X'01'.
1	1	Key block counter	A counter that is incremented for each 16-byte block of keying material generated for a pair of encryption and MAC keys. Starts at 1 for each key being generated. Allowed values: X'01' – X'02'. For example, an AES 256 bit key is 32-bytes and requires two 16-byte key blocks with a key block counter of X'02'. An AES 128 bit key is 16-bytes and would only require one 16-byte key block with a key block counter of X'01'.

Table 684. AES-DUKPT derivation data (continued)

Offset	Length of field (bytes)	Field name	Description
2	2	Key usage indicator	Indicates how the key to be derived is to be used. The initial terminal key is always a key derivation key. Allowed values: **X'0002'** Key Encryption Key. **X'1000'** PIN Encryption. **X'2000'** Message Authentication, generation. **X'2001'** Message Authentication, verification. **X'2002'** Message Authentication, both generation and verification. **X'3000'** Data Encryption, encryption. **X'3001'** Data Encryption, decryption. **X'3002'** Data Encryption, both encryption and decryption. **X'8000'** Key Derivation. **X'8001'** Key Derivation Initial Key (Note: This value cannot be used for working keys. This is the only value that is allowed for an initial terminal key). **Note:** X'2000', X'2001', X'3000', and X'3001' do not represent complementary keys. A key derived using X'2000' will not be the same key as is derived for the X'2001' usage value. Instead, the values represent the viewpoint of the terminal operator (for example, whether the terminal will use the key for that purpose).

Table 684. AES-DUKPT derivation data (continued)

Offset	Length of field (bytes)	Field name	Description
4	2	Algorithm indicator	Indicates the encipherment algorithm that is going to use the derived key. Allowed values: **X'0000'** 2-key TDES. **X'0001'** 3-key TDES. **X'0002'** AES 128 bit (Note: This is the value range that is allowed for an initial terminal key). **X'0003'** AES 192 bit (Note: This is the value range that is allowed for an initial terminal key). **X'0004'** AES 256 bit (Note: This is the value range that is allowed for an initial terminal key). **X'0005'** HMAC.
6	2	Length	Length, in bits, of the keying material being generated. Allowed values: **X'0080'** If 128 bits is being generated (AES-128 (Note: This is the only value that is allowed for an initial terminal key), 2TDES, or 128-bit HMAC key). **X'00C0'** If 192 bits is being generated (AES-192 (Note: This is the only value that is allowed for an initial terminal key), 3TDES, or 192-bit HMAC key). **X'0100'** If 256 bits is being generated (AES-256 (Note: This is the only value that is allowed for an initial terminal key) or 256-bit HMAC key).
8	8	Initial key ID	The terminal's initial key ID, the leftmost 64 bits of the key serial number. Allowed range: X'0000000000000000' to X'FFFFFFFFFFFFFFFF'.
16	4	Transaction counter	The 32-bit transaction counter. Allowed range: X'00000000' to X'FFFFFFFF'.

Supported CCA key types for AES-DUKPT derived working keys

The following table lists the types of CCA keys that can be derived using the AES-DUKPT algorithm:

Table 685. Supported CCA keys types for AES-DUKPT derived working keys

Derived working key	AES key type	2-Key TDES/3-Key TDES key type
PIN Encryption	PINPROT	IPINENC/OPINENC
MAC	MAC	MAC / MACVER
Key encrypting key	IMPORTER/EXPORTER	IMPORTER/EXPORTER
Key derivation key	DKYGENKY	KEYGENKY (2-Key TDES only)
Data encryption	CIPHER	CIPHER/ENCIPHER/DECIPHER
HMAC	N/A	N/A

AES-DUKPT allowed derived working key sizes

ANSI X9.24 specifies that working keys shall be the same strength or weaker than the key from which they are derived. The following table shows allowed working key sizes per AES Base Derivation Key size:

Table 686. AES-DUKPT allowed derived working key sizes

Derived working key	AES-128 base derivation key	AES-192 base derivation key	AES-256 base derivation key
2TDES	Allowed	Allowed	Allowed
3TDES	Allowed	Allowed	Allowed
AES-128	Allowed	Allowed	Allowed
AES-192	Not Allowed	Allowed	Allowed
AES-256	Not Allowed	Not Allowed	Allowed
HMAC-128	Allowed	Allowed	Allowed
HMAC-192	Not Allowed	Allowed	Allowed
HMAC-256	Not Allowed	Not Allowed	Allowed

Appendix L. CCA release levels

The following tables list the CCA release level used in the required hardware tables for all the CCA services:

- CCA release levels for IBM z16: Table 687 on page 1723
- CCA release levels for IBM z15: Table 688 on page 1723
- CCA release levels for IBM z14: Table 689 on page 1725
- CCA release levels for IBM z13: Table 690 on page 1726

CCA release
The short designation of the release of the licensed internal code (LIC). This designation is used in the access control table. It is the same as the release level used by the IBM 4767/4768/4769PCIe Cryptographic Coprocessors.

ICSF release and APAR number (if applicable)
The ICSF release and APAR number (if applicable) indicates where the support for the code was introduced. For APARs, there may be multiple CCA releases involved. The APAR number is listed with the oldest release of ICSF that was changed for that APAR.

Crypto Express adapter
The adapters for which the code is available.

CCA licensed internal code (LIC) release
The date that the CCA code was made available and the MCL number.

Table 687. CCA release levels for IBM z16

CCA release	ICSF release and APAR number (if applicable)	Crypto Express adapter	CCA licensed internal code (LIC) release
8.1	HCR77D1 OA61978	CEX8C	May 2023 Driver D51C MCL P30750.006
8.0	HCR77D2 OA61609	CEX8C	May 2022 Driver D51C MCL P30750.002
7.4	HCR77D2 OA61609	CEX7C	May 2022 Driver D51C MCL P30748.001
6.7	HCR77D2 OA61609	CEX6C	May 2022 Driver D51C MCL P30746.001

Table 688. CCA release levels for IBM z15

CCA release	ICSF release and APAR number (if applicable)	Crypto Express adapter	CCA licensed internal code (LIC) release
7.4	HCR77D1 OA61253	CEX7C	September 2021 Driver D41C MCL P46646.017
6.7	HCR77D1 OA61253	CEX6C	September 2021 Driver D41C MCL P46644.012

Table 688. CCA release levels for IBM z15 (continued)

CCA release	ICSF release and APAR number (if applicable)	Crypto Express adapter	CCA licensed internal code (LIC) release
7.3	HCR77D1 OA60318	CEX7C	May 2021 Driver 41C MCL 46646.014
6.6	HCR77D1 OA60318	CEX6C	May 2021 Driver 41C MCL 46644.010
5.7	HCR77D1 OA60318	CEX5C	May 2021 Driver 41C MCL 46642.009
7.2	HCR77D1 OA59593	CEX7C	September 2020 Driver 41C MCL 46646.011
6.5	HCR77D1 OA59593	CEX6C	September 2020 Driver 41C MCL P46644.007
7.1	HCR77C1 OA58880	CEX7C	June 2020 Driver D41C MCL P46646.008
6.4	HCR77C1 OA58880	CEX6C	June 2020 Driver D41C MCL P46644.006
5.6	HCR77C1 OA58880	CEX5C	June 2020 Driver D41C MCL P46642.005
7.0	HCR77C1 OA58306	CEX7C	November 2019 Driver D41C MCL P46646.004
6.3	HCR77C1 OA58306	CEX6C	November 2019 Driver D36C MCL P41456.005 Driver D36C MCL P41456.006
5.5	HCR77C1 OA58306	CEX5C	November 2019 Driver D41C MCL P46642.003
7.0	HCR77D1 z/OS V2R2-V2R4	CEX7C	September 2019
6.3	HCR77D1 z/OS V2R2-V2R4	CEX6C	September 2019
5.5	HCR77D1 z/OS V2R2-V2R4	CEX5C	September 2019

Table 689. CCA release levels for the IBM z14

CCA release	ICSF release and APAR number (if applicable)	Crypto Express adapter	CCA licensed internal code (LIC) release
6.7	HCR77D1 OA61253	CEX6C	September 2021 Driver D36C MCL P41458.012
6.6	HCR77D1 OA60318	CEX6C	July 2021 Driver 36C MCL 41458.011
5.7	HCR77D1 OA60318	CEX5C	July 2021 Driver 36C MCL 41456.010
6.5	HCR77D1 OA59593 OA60355	CEX6C	October 2020 Driver 36C MCL P41458.010
6.4	HCR77C1 OA58880	CEX6C	June 2020 Driver D36C MCL P41458.009
5.6	HCR77C1 OA58880	CEX5C	June 2020 Driver D36C MCL P41456.008
6.3	HCR77C1 OA58306	CEX6C	November 2019 Driver D41C MCL P46644.003
5.5	HCR77C1 OA58306	CEX5C	November 2019 Driver D36C MCL P41456.005 Driver D36C MCL P41456.006
6.3	HCR77D0 OA57089	CEX6C	July 2019 Driver D36C MCL P41458.004
5.5	HCR77D0 OA57089	CEX5C	July 2019 Driver D36C MCL P41456.004
6.3	HCR77D0 OA57088	CEX6C	July 2019 Driver D36C MCL P41458.004
5.5	HCR77D0 OA57088	CEX5C	July 2019 Driver D36C MCL P41456.004
6.2	HCR77D0 z/OS V2R2-V2R4	CEX6C	December 2018
6.1	HCR77C1 OA55184	CEX5C	December 2018 Driver D36C MCL P41458.002

Table 689. CCA release levels for the IBM z14 (continued)

CCA release	ICSF release and APAR number (if applicable)	Crypto Express adapter	CCA licensed internal code (LIC) release
5.4	HCR77C1 OA55184	CEX5C	December 2018 Driver D32L MCL P42641.004
6.0	HCR77C1 z/OS V2R1-V2R3	CEX6C	September 2017

Table 690. CCA release levels for the IBM z13

CCA release	ICSF release and APAR number (if applicable)	Crypto Express adapter	CCA licensed internal code (LIC) release
5.7	HCR77D1 OA60318	CEX5C	July 2021 Driver D27I MCL 08449.024
5.6	HCR77C1 OA58880	CEX5C	June 2020 Driver D27I MCL P08449.022
5.5	HCR77C1 OA58306	CEX5C	November 2019 Driver D27I MCL P08449.020
5.5	HCR77D0 OA57089	CEX5C	July 2019 Driver D27I MCL P08449.019
5.5	HCR77D0 OA57088	CEX5C	July 2019 Driver D27I MCL P08449.019
5.4	HCR77C1 OA55184	CEX5C	December 2018 Driver D27I MCL P08449.019
5.3	HCR77C0 z/OS V2R1-V2R3	CEX5C	October 2016
5.2	HCR77B1 z/OS V1R13-V2R2	CEX5C	March 2016
5.1	HCR77B1 OA49064	CEX5C	July 2015
5.0	HCR77B0 z/OS V1R13-V2R2	CEX5C	February 2015

Appendix M. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Glossary

This glossary defines terms and abbreviations used in Integrated Cryptographic Service Facility (ICSF).

This glossary includes terms and definitions from:

- The American National Standard Dictionary for Information Technology, ANSI INCITS 172, by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The Information Technology Vocabulary, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Definitions specific to the Integrated Cryptographic Services Facility are labeled “In ICSF.”

access method services (AMS)
The facility used to define and reproduce VSAM key-sequenced data sets (KSDS).

Advanced Encryption Standard (AES)
In computer security, the National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) algorithm.

AES
Advanced Encryption Standard.

American National Standard Code for Information Interchange (ASCII)
The standard code using a coded character set consisting of 7-bit characters (8 bits including parity check) that is used for information exchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

ANSI X9.19
An ANSI standard that specifies an optional double-MAC procedure which requires a double-length MAC key.

application program
A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll.

A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

application program interface (API)
A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

In ICSF, a callable service.

asymmetric cryptography
Synonym for public key cryptography.

authentication pattern
An 8-byte pattern that ICSF calculates from the master key when initializing the cryptographic key data set. ICSF places the value of the authentication pattern in the header record of the cryptographic key data set.

authorized program facility (APF)
A facility that permits identification of programs authorized to use restricted functions.

callable service
A predefined sequence of instructions invoked from an application program, using a CALL instruction. In ICSF, callable services perform cryptographic functions and utilities.

CBC
Cipher block chaining.

CCA
Common Cryptographic Architecture.

CCF
Cryptographic Coprocessor Feature.

CDMF
Commercial Data Masking Facility.

CEDA
A CICS transaction that defines resources online. Using CEDA, you can update both the CICS system definition data set (CSD) and the running CICS system.

Central Credit Committee
The official English name for *Zentraler Kreditausschuss*, also known as ZKA. ZKA was founded in 1932 and was renamed in August 2011 to *Die Deutsche Kreditwirtschaft*, also known as DK. DK is an association of the German banking industry. The hybrid term in English for DK is 'German Banking Industry Committee'.

CEX5A
Crypto Express5 Accelerator

CEX5C
Crypto Express5 CCA Coprocessor

CEX5P
Crypto Express5 PKCS #11 Coprocessor

CEX6A
Crypto Express6 Accelerator

CEX6C
Crypto Express6 CCA Coprocessor

CEX6P
Crypto Express6 PKCS #11 Coprocessor

CEX7A
Crypto Express7 Accelerator

CEX7C
Crypto Express7 CCA Coprocessor

CEX7P
Crypto Express7 PKCS #11 Coprocessor

CEX8A
Crypto Express8 Accelerator

CEX8C
Crypto Express8 CCA Coprocessor

CEX8P
Crypto Express8 PKCS #11 Coprocessor

checksum
The sum of a group of data associated with the group and used for checking purposes. (T)

In ICSF, the data used is a key part. The resulting checksum is a two-digit value you enter when you enter a master key part.

Chinese Remainder Theorem (CRT)
A mathematical theorem that defines a format for the RSA private key that improves performance.

CICS
Customer Information Control System.

cipher block chaining (CBC)
A mode of encryption that uses the data encryption algorithm and requires an initial chaining vector. For encipher, it exclusively ORs the initial block of data with the initial control vector and then enciphers it. This process results in the encryption both of the input block and of the initial control vector that it uses on the next input block as the process repeats. A comparable chaining process works for decipher.

ciphertext
In computer security, text produced by encryption.

Synonym for enciphered data.

CKDS
Cryptographic Key Data Set.

clear key
Any type of encryption key not protected by encryption under another key.

CMOS
Complementary metal oxide semiconductor.

coexistence mode
An ICSF method of operation during which CUSP or PCF can run independently and simultaneously on the same ICSF system. A CUSP or PCF application program can run on ICSF in this mode if the application program has been reassembled.

Commercial Data Masking Facility (CDMF)
A data-masking algorithm using a DES-based kernel and a key that is shortened to an effective key length of 40 DES key-bits. Because CDMF is not as strong as DES, it is called a masking algorithm rather than an encryption algorithm. Implementations of CDMF, when used for data confidentiality, are generally exportable from the USA and Canada.

Common Cryptographic Architecture: Cryptographic Application Programming Interface
Defines a set of cryptographic functions, external interfaces, and a set of key management rules that provide a consistent, end-to-end cryptographic architecture across different IBM platforms.

compatibility mode
An ICSF method of operation during which a CUSP or PCF application program can run on ICSF without recompiling it. In this mode, ICSF cannot run simultaneously with CUSP or PCF.

complementary keys
A pair of keys that have the same clear key value, are different but complementary types, and usually exist on different systems.

console
A part of a computer used for communication between the operator or maintenance engineer and the computer. (A)

control-area split
In systems with VSAM, the movement of the contents of some of the control intervals in a control area to a newly created control area in order to facilitate insertion or lengthening of a data record when there are no remaining free control intervals in the original control area.

control block
A storage area used by a computer program to hold control information. (I) Synonymous with control area.

The circuitry that performs the control functions such as decoding microinstructions and generating the internal control signals that perform the operations requested. (A)

control interval
A fixed-length area of direct-access storage in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records pointed to by an entry in the sequence-set index record. The control interval is the unit of information that VSAM transmits

to or from direct access storage. A control interval always comprises an integral number of physical records.

control interval split
In systems with VSAM, the movement of some of the stored records in a control interval to a free control interval to facilitate insertion or lengthening of a record that does not fit in the original control interval.

control statement input data set
A key generator utility program data set containing control statements that a particular key generator utility program job will process.

control statement output data set
A key generator utility program data set containing control statements to create the complements of keys created by the key generator utility program.

control vector
In ICSF, a mask that is exclusive ORed with a master key or a transport key before ICSF uses that key to encrypt another key. Control vectors ensure that keys used on the system and keys distributed to other systems are used for only the cryptographic functions for which they were intended.

CPACF
CP Assist for Cryptographic Functions

CP Assist for Cryptographic Functions
Implemented on all IBM servers to provide AES and DES encryption and SHA-1 secure hashing.

cross memory mode
Synchronous communication between programs in different address spaces that permits a program residing in one address space to access the same or other address spaces. This synchronous transfer of control is accomplished by a calling linkage and a return linkage.

CRT
Chinese Remainder Theorem.

Crypto Express5 Coprocessor
An asynchronous cryptographic coprocessor available on IBM z13 and IBM z13s.

Crypto Express6 Coprocessor
An asynchronous cryptographic coprocessor available on IBM z14 and IBM z14 ZR1.

Crypto Express7 Coprocessor
An asynchronous cryptographic coprocessor available on IBM z15.

Crypto Express8 Coprocessor
An asynchronous cryptographic coprocessor available on IBM z16.

cryptographic adapter (4764, 4765, and 4767)
An expansion board that provides a comprehensive set of cryptographic functions for the network security processor and the workstation in the TSS family of products.

cryptographic coprocessor
A tamper responding, programmable, cryptographic PCI card, containing CPU, encryption hardware, RAM, persistent memory, hardware random number generator, time of day clock, infrastructure firmware, and software.

cryptographic key data set (CKDS)
A data set that contains the encrypting keys used by an installation.

In ICSF, a VSAM data set that contains all the cryptographic keys. Besides the encrypted key value, an entry in the cryptographic key data set contains information about the key.

cryptography
The transformation of data to conceal its meaning.

In computer security, the principles, means, and methods for encrypting plaintext and decrypting ciphertext.

In ICSF, the use of cryptography is extended to include the generation and verification of MACs, the generation of MDCs and other one-way hashes, the generation and verification of PINs, and the generation and verification of digital signatures.

CUSP (Cryptographic Unit Support Program)
The IBM cryptographic offering, program product 5740-XY6, using the channel-attached 3848. CUSP is no longer in service.

CUSP/PCF conversion program
A program, for use during migration from CUSP or PCF to ICSF, that converts a CUSP or PCF cryptographic key data set into a ICSF cryptographic key data set.

Customer Information Control System (CICS)
An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user written application programs. It includes facilities for building, using, and maintaining databases.

CVC
Card verification code used by MasterCard.

CVV
Card verification value used by VISA.

data encryption algorithm (DEA)
In computer security, a 64-bit block cipher that uses a 64-bit key, of which 56 bits are used to control the cryptographic process and 8 bits are used for parity checking to ensure that the key is transmitted properly.

data encryption standard (DES)
In computer security, the National Institute of Standards and Technology (NIST) Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm.

data key or data-encrypting key
A key used to encipher, decipher, or authenticate data.

In ICSF, a 64-bit encryption key used to protect data privacy using the DES algorithm. AES data keys are now supported by ICSF.

data set
The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

data-translation key
A 64-bit key that protects data transmitted through intermediate systems when the originator and receiver do not share the same key.

DEA
Data encryption algorithm.

decipher
To convert enciphered data in order to restore the original data. (T)

In computer security, to convert ciphertext into plaintext by means of a cipher system.

To convert enciphered data into clear data. Contrast with encipher. Synonymous with decrypt.

decode
To convert data by reversing the effect of some previous encoding. (I) (A)

In ICSF, to decipher data by use of a clear key.

decrypt
See decipher.

DES
Data Encryption Standard.

diagnostics data set
A key generator utility program data set containing a copy of each input control statement followed by a diagnostic message generated for each control statement.

digital signature
In public key cryptography, information created by using a private key and verified by using a public key. A digital signature provides data integrity and source nonrepudiation.

Digital Signature Algorithm (DSA)
A public key algorithm for digital signature generation and verification used with the Digital Signature Standard.

Digital Signature Standard (DSS)
A standard describing the use of algorithms for digital signature purposes. One of the algorithms specified is DSA (Digital Signature Algorithm).

Dilithium
A quantum-safe cryptographic algorithm. Also known as LI2.

DK
Die Deutsche Kreditwirtschaft (German Banking Industry Committee). Formerly known as ZKA.

domain
That part of a network in which the data processing resources are under common control. (T)

In ICSF, an index into a set of master key registers.

DSA
Digital Signature Algorithm.

DSS
Digital Signature Standard.

ECB
Electronic codebook.

ECC
Elliptic Curve Cryptography.

ECI
Eurocheque International S.C., a financial institution consortium that has defined three PIN block formats.

EID
Environment Identification.

electronic codebook (ECB) operation
A mode of operation used with block cipher cryptographic algorithms in which plaintext or ciphertext is placed in the input to the algorithm and the result is contained in the output of the algorithm.

A mode of encryption using the data encryption algorithm, in which each block of data is enciphered or deciphered without an initial chaining vector. It is used for key management functions and the encode and decode callable services.

electronic funds transfer system (EFTS)
A computerized payment and withdrawal system used to transfer funds from one account to another and to obtain related financial data.

encipher
To scramble data or to convert data to a secret code that masks the meaning of the data to any unauthorized recipient. Synonymous with encrypt.

Contrast with decipher.

enciphered data
Data whose meaning is concealed from unauthorized users or observers.

encode
To convert data by the use of a code in such a manner that reconversion to the original form is possible. (T)

In computer security, to convert plaintext into an unintelligible form by means of a code system.

In ICSF, to encipher data by use of a clear key.

encrypt
See encipher.

exit
To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T)

In ICSF, a user-written routine that receives control from the system during a certain point in processing—for example, after an operator issues the START command.

exportable form
A condition a key is in when enciphered under an exporter key-encrypting key. In this form, a key can be sent outside the system to another system. A key in exportable form cannot be used in a cryptographic function.

exporter key-encrypting key
A 128-bit key used to protect keys sent to another system. A type of transport key.

file
A named set of records stored or processed as a unit. (T)

GBP
German Bank Pool.

German Bank Pool (GBP)
A German financial institution consortium that defines specific methods of PIN calculation.

German Banking Industry Committee
A hybrid term in English for *Die Deutsche Kreditwirtschaft*, also known as DK, an association of the German banking industry. Prior to August 2011, DK was named ZKA for *Zentraler Kreditausschuss*, or Central Credit Committee. ZKA was founded in 1932.

hashing
An operation that uses a one-way (irreversible) function on data, usually to reduce the length of the data and to provide a verifiable authentication value (checksum) for the hashed data.

header record
A record containing common, constant, or identifying information for a group of records that follows.

ICSF
Integrated Cryptographic Service Facility.

importable form
A condition a key is in when it is enciphered under an importer key-encrypting key. A key is received from another system in this form. A key in importable form cannot be used in a cryptographic function.

importer key-encrypting key
A 128-bit key used to protect keys received from another system. A type of transport key.

initial chaining vector (ICV)
A 64-bit random or pseudo-random value used in the cipher block chaining mode of encryption with the data encryption algorithm.

initial program load (IPL)
The initialization procedure that causes an operating system to commence operation.

The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction.

The process of loading system programs and preparing a system to run jobs.

input PIN-encrypting key
A 128-bit key used to protect a PIN block sent to another system or to translate a PIN block from one format to another.

installation exit
See exit.

Integrated Cryptographic Service Facility (ICSF)
A licensed program that runs under MVS/System Product 3.1.3, or higher, or OS/390 Release 1, or higher, or z/OS, and provides access to the hardware cryptographic feature for programming applications. The combination of the hardware cryptographic feature and ICSF provides secure high-speed cryptographic services.

International Organization for Standardization
An organization of national standards bodies from many countries, established to promote the development of standards to facilitate the international exchange of goods and services and to develop cooperation in intellectual, scientific, technological, and economic activity. ISO has defined certain standards relating to cryptography and has defined two PIN block formats.

ISO
International Organization for Standardization.

job control language (JCL)
A control language used to identify a job to an operating system and to describe the job's requirements.

key-encrypting key (KEK)
In computer security, a key used for encryption and decryption of other keys.

In ICSF, a master key or transport key.

key generator utility program (KGUP)
A program that processes control statements for generating and maintaining keys in the cryptographic key data set.

key output data set
A key generator utility program data set containing information about each key that the key generator utility program generates except an importer key for file encryption.

key part
A 32-digit hexadecimal value that you enter for ICSF to combine with other values to create a master key or clear key.

key part register
A register in a cryptographic coprocessor that accumulates key parts as they are entered via TKE.

key store policy
Ensures that only authorized users and jobs can access secure key tokens that are stored in one of the ICSF key stores - the CKDS or the PKDS.

key store policy controls
Resources that are defined in the XFACILIT class. A control can verify the caller has authority to use a secure token and identify the action to take when the secure token is not stored in the CKDS or PKDS.

Kyber
A quantum-safe cryptographic algorithm.

LI2
Abbreviation for the Dilithium quantum-safe algorithm.

linkage
The coding that passes control and parameters between two routines.

load module
All or part of a computer program in a form suitable for loading into main storage for execution. A load module is usually the output of a linkage editor. (T)

LPAR mode
The central processor mode that enables the operator to allocate the hardware resources among several logical partitions.

MAC generation key
A 64-bit or 128-bit key used by a message originator to generate a message authentication code sent with the message to the message receiver.

MAC verification key
A 64-bit or 128-bit key used by a message receiver to verify a message authentication code received with a message.

magnetic tape
A tape with a magnetizable layer on which data can be stored. (T)

master key
In computer security, the top-level key in a hierarchy of key-encrypting keys.

ICSF uses master keys to encrypt operational keys. Master keys are known only to the cryptographic coprocessors and are maintained in tamper proof cryptographic coprocessors.

master key concept
The idea of using a single cryptographic key, the master key, to encrypt all other keys on the system.

master key register
A register in the cryptographic coprocessors that stores the master key that is active on the system.

master key variant
A key derived from the master key by use of a control vector. It is used to force separation by type of keys on the system.

MD5
Message Digest 5. A hash algorithm.

message authentication code (MAC)
The cryptographic result of block cipher operations on text or data using the cipher block chain (CBC) mode of operation.

In ICSF, a MAC is used to authenticate the source of the message, and verify that the message was not altered during transmission or storage.

modification detection code (MDC)
A 128-bit value that interrelates all bits of a data stream so that the modification of any bit in the data stream results in a new MDC.

In ICSF, an MDC is used to verify that a message or stored data has not been altered.

multiple encipherment
The method of encrypting a key under a double-length key-encrypting key.

new master key register
A register in a cryptographic coprocessor that stores a master key before you make it active on the system.

NIST
U.S. National Institute of Science and Technology.

NOCV processing
Process by which the key generator utility program or an application program encrypts a key under a transport key itself rather than a transport key variant.

noncompatibility mode
An ICSF method of operation during which CUSP or PCF can run independently and simultaneously on the same z/OS, OS/390, or MVS system. You cannot run a CUSP or PCF application program on ICSF in this mode.

nonrepudiation
A method of ensuring that a message was sent by the appropriate individual.

OAEP
Optimal asymmetric encryption padding.

offset
The process of exclusively ORing a counter to a key.

old master key register
A register in a cryptographic coprocessor that stores a master key that you replaced with a new master key.

operational form
The condition of a key when it is encrypted under the master key so that it is active on the system.

output PIN-encrypting key
A 128-bit key used to protect a PIN block received from another system or to translate a PIN block from one format to another.

PAN
Personal Account Number.

parameter
Data passed between programs or procedures.

parmlib
A system parameter library, either SYS1.PARMLIB or an installation-supplied library.

partitioned data set (PDS)
A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

Personal Account Number (PAN)
A Personal Account Number identifies an individual and relates that individual to an account at a financial institution. It consists of an issuer identification number, customer account number, and one check digit.

personal identification number (PIN)
The 4- to 12-digit number entered at an automatic teller machine to identify and validate the requester of an automatic teller machine service. Personal identification numbers are always enciphered at the device where they are entered, and are manipulated in a secure fashion.

Personal Security card
An ISO-standard “smart card” with a microprocessor that enables it to perform a variety of functions such as identifying and verifying users, and determining which functions each user can perform.

PIN block
A 64-bit block of data in a certain PIN block format. A PIN block contains both a PIN and other data.

PIN generation key
A 128-bit key used to generate PINs or PIN offsets algorithmically.

PIN key
A 128-bit key used in cryptographic functions to generate, transform, and verify the personal identification numbers.

PIN offset
For 3624, the difference between a customer-selected PIN and an institution-assigned PIN. For German Bank Pool, the difference between an institution PIN (generated with an institution PIN key) and a pool PIN (generated with a pool PIN key).

PIN verification key
A 128-bit key used to verify PINs algorithmically.

PKA
Public Key Algorithm.

PKCS
Public Key Cryptographic Standards (RSA Data Security, Inc.)

PKDS
Public key data set (PKA cryptographic key data set).

plaintext
Data in normal, readable form.

primary space allocation
An area of direct access storage space initially allocated to a particular data set or file when the data set or file is defined. See also secondary space allocation.

private key
In computer security, a key that is known only to the owner and used with a public key algorithm to decrypt data or generate digital signatures. The data is encrypted and the digital signature is verified using the related public key.

processor complex
A configuration that consists of all the machines required for operation.

Processor Resource/Systems Manager
Enables logical partitioning of the processor complex, may provide additional byte-multiplexer channel capability, and supports the VM/XA System Product enhancement for Multiple Preferred Guests.

Programmed Cryptographic Facility (PCF)
An IBM licensed program that provides facilities for enciphering and deciphering data and for creating, maintaining, and managing cryptographic keys.

The IBM cryptographic offering, program product 5740-XY5, using software only for encryption and decryption. This product is no longer in service; ICSF is the replacement product.

PR/SM
Processor Resource/Systems Manager.

public key
In computer security, a key made available to anyone who wants to encrypt information using the public key algorithm or verify a digital signature generated with the related private key. The encrypted data can be decrypted only by use of the related private key.

public key algorithm (PKA)
In computer security, an asymmetric cryptographic process in which a public key is used for encryption and digital signature verification and a private key is used for decryption and digital signature generation.

public key cryptography
In computer security, cryptography in which a public key is used for encryption and a private key is used for decryption. Synonymous with asymmetric cryptography.

QSA
Quantum-safe algorithm. Examples include CRYSTALS-Dilithium Digital Signature Algorithm and CRYSTALS-Kyber key encapsulation mechanism (KEM).

RACE Integrity Primitives Evaluatiuon Message Digest
A hash algorithm.

RDO
Resource definition online.

record chaining
When there are multiple cipher requests and the output chaining vector (OCV) from the previous encipher request is used as the input chaining vector (ICV) for the next encipher request.

Resource Access Control Facility (RACF)
An IBM licensed program that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.

retained key
A private key that is generated and retained within the secure boundary of the Crypto Express adapter.

return code
A code used to influence the execution of succeeding instructions. (A)

A value returned to a program to indicate the results of an operation requested by that program.

Rivest-Shamir-Adleman (RSA) algorithm
A process for public key cryptography that was developed by R. Rivest, A. Shamir, and L. Adleman.

RMF
Resource Manager Interface.

RMI
Resource Measurement Facility.

RSA
Rivest-Shamir-Adleman.

RSA-PSS
RSA-Probabilistic Signature Scheme. RSA-PSS is a signature scheme that is based on the RSA cryptosystem and provides increased security assurance. It was added in version 2.1 of PKCS #1.

SAF
System Authorization Facility.

save area
Area of main storage in which contents of registers are saved. (A)

secondary space allocation
In systems with VSAM, area of direct access storage space allocated after primary space originally allocated is exhausted. See also primary space allocation.

Secure Electronic Transaction
A standard created by Visa International and MasterCard for safe-guarding payment card purchases made over open networks.

secure key
A key that is encrypted under a master key. When ICSF uses a secure key, it is passed to a cryptographic coprocessor where the coprocessor decrypts the key and performs the function. The secure key never appears in the clear outside of the cryptographic coprocessor.

Secure Sockets Layer
A security protocol that provides communications privacy over the Internet by allowing client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

sequential data set
A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape.

SET
Secure Electronic Transaction.

SHA (Secure Hash Algorithm, FIPS 180)
(Secure Hash Algorithm, FIPS 180) The SHA (Secure Hash Algorithm) family is a set of related cryptographic hash functions designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST). The first member of the family, published in 1993, is officially called SHA. However, today, it is often unofficially called SHA-0 to avoid confusion with its successors. Two years later, SHA-1, the first successor to SHA, was published. Four more variants, have since been published with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).

SHA-1 (Secure Hash Algorithm 1, FIPS 180)
A hash algorithm required for use with the Digital Signature Standard.

SHA-2 (Secure Hash Algorithm 2, FIPS 180)
Four additional variants to the SHA family, with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).

SHA-3 (Secure Hash Algorithm 3, FIPS 202)
SHA-3 is a subset of the cryptographic primitive family Keccak and is used to build instances of Permutation-Based Hash and Extendable-Output Functions (see also SHAKE). Because of the successful attacks on MD5, SHA-0, and SHA-1, NIST perceived a need for an alternative, dissimilar cryptographic hash, which became SHA-3.

SHA-224
One of the SHA-2 algorithms.

SHA-256
One of the SHA-2 algorithms.

SHA-384
One of the SHA-2 algorithms.

SHA-512
One of the SHA-2 algorithms.

SHA3-224
An instance of the SHA-3 algorithm that provides a Permutation-Based Hash.

SHA3-256
An instance of the SHA-3 algorithm that provides a Permutation-Based Hash.

SHA3-384
An instance of the SHA-3 algorithm that provides a Permutation-Based Hash.

SHA3-512
An instance of the SHA-3 algorithm that provides a Permutation-Based Hash.

SHAKE (combination of Secure Hash Algorithm and Keccak)
A set of Extendable-Output Functions defined in FIPS PUB 202.

SHAKE128
An instance of the SHA-3 algorithm that provides an Extendable-Output Function.

SHAKE256
An instance of the SHA-3 algorithm that provides an Extendable-Output Function.

smart card
A plastic card that has a microchip capable of storing data or process information.

special secure mode
An alternative form of security that allows you to enter clear keys with the key generator utility program or generate clear PINs.

SSL
Secure Sockets Layer.

supervisor state
A state during which a processing unit can execute input/output and other privileged instructions.

System Authorization Facility (SAF)
An interface to a system security system like the Resource Access Control Facility (RACF).

system key
A key that ICSF creates and uses for internal processing.

System Management Facility (SMF)
A base component of z/OS that provides the means for gathering and recording information that can be used to evaluate system usage.

TDEA
Triple Data Encryption Algorithm.

TKE
Trusted key entry.

Transaction Security System
An IBM product offering including both hardware and supporting software that provides access control and basic cryptographic key-management functions in a network environment. In the workstation environment, this includes the 4755 Cryptographic Adapter, the Personal Security Card, the 4754 Security Interface Unit, the Signature Verification feature, the Workstation Security Services Program, and the AIX Security Services Program/6000. In the host environment, this includes the 4753 Network Security Processor and the 4753 Network Security Processor MVS Support Program.

transport key
A key used to protect keys distributed from one system to another. A transport key can be an AES or DES key-encrypting key (importer or exporter).

transport key variant
A key derived from a transport key by use of a control vector. It is used to force separation by type for keys sent between systems.

TRUE
Task-related User Exit (CICS). The CICS-ICSF Attachment Facility provides a CSFATRUE and CSFATREN routine.

UAT
UDX Authority Table.

UDF
User-defined function.

UDK
User-derived key.

UDP
User Developed Program.

UDX
User Defined Extension.

verification pattern
An 8-byte pattern that ICSF calculates from the key parts you enter when you enter a master key or clear key. You can use the verification pattern to verify that you have entered the key parts correctly and specified a certain type of key.

Virtual Storage Access Method (VSAM)
An access method for indexed or sequential processing of fixed and variable-length records on direct-access devices. The records in a VSAM data set or file can be organized in logical sequence by means of a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by means of relative-record number.

Virtual Telecommunications Access Method (VTAM)
An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VISA
A financial institution consortium that has defined four PIN block formats and a method for PIN verification.

VISA PIN Verification Value (VISA PVV)
An input to the VISA PIN verification process that, in practice, works similarly to a PIN offset.

3621
A model of an IBM Automatic Teller Machine that has a defined PIN block format.

3624
A model of an IBM Automatic Teller Machine that has a defined PIN block format and methods of PIN calculation.

4764
The IBM 4764 PCI-X Cryptographic Coprocessor processor provides a secure programming and hardware environment where AES, DES, and RSA processes are performed.

4765
The IBM 4765 PCIe Cryptographic Coprocessor processor provides a secure programming and hardware environment where AES, DES, ECC, and RSA processes are performed.

4767
The IBM 4767 PCIe Cryptographic Coprocessor processor provides a secure programming and hardware environment where AES, DES, ECC, and RSA processes are performed.

Index

Numerics

1-pass key transport 1069
2-pass key transport 1066
3621 PIN block format 610, 1633
3624 PIN block format 610, 1633
4700-PAD processing rule 487, 488, 496, 497
4704-EPP PIN block format 610

A

accessibility
  contact IBM 1727
accessing
  callable service 9
  invocation requirements 9
AES-DUKPT allowed derived working key sizes 1722
AES-DUKPT derivation data 1719
AES-DUKPT reference information 1719
AES-DUKPT supported CCA key types 1722
affinity (IEAAFFN callable service) 9
ALET (alternate entry point)
  format 4
algorithm
  3624 PIN generation 1635
  3624 PIN verification 1638
  GBP PIN generation 1636
  GBP PIN verification 1640
  GBP-PIN 716
  GBP-PINO 716
  IBM-PIN 716
  IBM-PINO 716
  PIN offset generation 1637
  PIN, detailed 1635
  PIN, general 64
  PVV generation 1641
  PVV verification 1642
  VISA PIN 1641
  VISA-PVV 638, 716
  VISAPVV4 716
ANSI 9.9-1 algorithm 543
ANSI INCITS 106 processing rule 1643
ANSI X9.19 optional double MAC procedure 543
ANSI X9.23 processing rule 487, 488, 496, 497, 1644
ANSI X9.8 683
ANSI X9.8 PIN block format 1632
ASCII to EBCDIC conversion
  table 1657
assistive technologies 1727
asym_encrypted_key parameter
  Remote Key Export callable service 399
asym_encrypted_key_length parameter
  Remote Key Export callable service 399
Australian Payment Network)
  support 37
authenticating messages 543
Authentication Parameter Generate (CSNBAPG)
Authentication Parameter Generate (CSNBAPG) *(continued)*
  format 620
  syntax 620
Authentication Parameter Generate callable service (CSNBAPG)
  parameters 620

B

BIND 1062

C

c_variable_encrypting_key_identifier parameter
  Cryptographic Variable Encipher callable service 129
call
  successful 11
  unsuccessful 11
callable service
  Authentication Parameter Generate (CSNBAPG and CSNEAPG) 64
  Authentication Parameter Generate (CSNBAPG) 620
  character/nibble conversion (CSNBXBC and CSNBXCB) 1267
  Cipher Text Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3) 471
  CKDS key record create (CSNBKRC) 57, 1202
  CKDS key record create2 (CSNBKRC2 and CSNEKRC2) 58
  CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2) 1203
  CKDS key record delete (CSNBKRD) 58, 1205
  CKDS key record read (CSNBKRR) 58, 1207
  CKDS key record read2 (CSNBKRR2 and CSNEKRR2) 58
  CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2) 1208
  CKDS key record write (CSNBKRW) 58, 1213
  CKDS key record write2 (CSNBKRW2 and CSNEKRW2) 58
  CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2) 1215
  clear key import (CSNBCKI) 37
  Clear Key Import (CSNBCKI) 116
  clear PIN encrypt (CSNBCPE) 64
  Clear PIN Encrypt (CSNBCPE) 624
  clear PIN generate (CSNBPGN) 65
  Clear PIN Generate (CSNBPGN) 629
  clear PIN generate alternate (CSNBCPA) 65
  Clear PIN Generate Alternate (CSNBCPA) 634
  code conversion (CSNBXAE) 72
  code conversion (CSNBXBC) 72
  code conversion (CSNBXCB) 72
  code conversion (CSNBXEA and CSNBXAE) 1269
  code conversion (CSNBXEA) 72
  coding examples
    C 1619
    COBOL 1621

callable service *(continued)*
coding examples *(continued)*
High Level Assembler 1623
PL/I 1625
control vector generate (CSNBCVG) 38, 118
control vector translate callable service (CSNBCVT) 38
Control Vector Translate callable service (CSNBCVT) 124
Coordinated KDS Administration (CSFCRC and CSFCRC6) 1217
coordinated KDS administration callable services (CSFCRC and CSFCRC6) 58
Crypto Usage Statistic (CSFSTAT) 1271
cryptographic variable encipher (CSNBCVE) 38
Cryptographic Variable Encipher (CSNBCVE) 128
CSFxxxx format 4
CSNBxxxx format 4
CVV Key Combine (CSNBCKC and CSNECKC) 640
data key export (CSNBDKX) 38
Data Key Export (CSNBDKX) 130
data key import (CSNBDKM) 38
Data Key Import (CSNBDKM) 133
decipher (CSNBDEC or CSNBDEC1) 484
decode (CSNBDCO) 490
definition 3, 15
Derive ICC MK (CSNBDCM) 136
Derive Session Key (CSNBDSK) 143
Digital Signature Generate (CSNDDSG) 96, 1110
Digital Signature Verify (CSNDDSV) 96, 1119
diversified key generate (CSNBDKG) 38
Diversified Key Generate (CSNBDKG) 151
Diversified Key Generate2 (CSNBDKG2) 41, 162
Diversify Directed Key (CSNBDDK and CSNEDDK) 36
Diversify Directed Key (CSNBDDK and CSNEDDK)) 170
DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG) 36, 850
DK Migrate PIN (CSNBDMP and CSNEDMP) 36
DK Migrate PIN callable service (CSNBDMP) 857
DK PAN Modify in Transaction (CSNBDPMT) 36
DK PAN Modify in Transaction callable service (CSNBDPMT) 863
DK PAN Translate (CSNBDPT and CSNEDPT) 36, 871
DK PIN Change (CSNBDPC) 36
DK PIN Change callable service (CSNBDPC) 878
DK PIN Verify (CSNBDPV) 36
DK PIN Verify callable service (CSNBDPV) 892
DK PRW Card Number Update (CSNBDPNU and CSNEDPNU) 36, 897
DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2) 36, 904
DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG) 36, 912
DK Random PIN Generate (CSNBDRPG) 37
DK Random PIN Generate callable service (CSNBDRPG) 916
DK Random PIN Generate2 (CSNBDRG2 and CSNEDRG2) 37
DK Random PIN Generate2 callable service (CSNBDRG2) 922
DK Regenerate PRW (CSNBDRP and CSNEDRP) 37, 930
ECC Diffie-Hellman (CSNDEDH and CSNFEDH) 182
EMV Scripting Service (CSNBESC) 646
EMV Transaction (ARQC/ARPC) Service (CSNBEAC) 657
EMV Verification Functions (CSNBEVF) 665
callable service *(continued)*
encipher (CSNBENC or CSNBENC1) 492
encode (CSNBECO) 499
encrypted PIN generate (CSNBEPG) 65, 671
encrypted PIN translate (CSNBPTR) 65, 678
encrypted PIN translate enhanced (CSNBPTRE) 701
encrypted PIN translate2 (CSNBPTR2) 65, 685
encrypted PIN verification (CSNBPVR) 66
encrypted PIN verification (CSNBPVR2) 66
encrypted PIN verify (CSNBPVR) 713
encrypted PIN verify2 (CSNBPVR2) 720
Field Level Decipher(CSNBFLD) 730
Field Level Encipher (CSNBFLE) 739
format 1327
Format Preserving Algorithms Decipher (CSNBFFXD) 749
Format Preserving Algorithms Encipher (CSNBFFXE) 754
Format Preserving Algorithms Translate (CSNBFFXT) 760
FPE Decipher (CSNBFPED) 67, 767
FPE Encipher (CSNBFPEE) 67, 776
FPE Translate (CSNBFPET) 67, 786
Generate Issuer MK callable service (CSNBGIM) 197
get attribute value (CSFPGAV) 1355
HMAC Generate (CSNBHMG, CSNEHMG, CSNBHMG1 and CSNEHMG1) 61, 545
HMAC Verify (CSNBHMV, CSNEHMV, CSNBHMV1 and CSNEHMV1) 61, 550
ICSF Multi-Purpose Service (CSFMPS and CSFMPS6) 59, 98, 1221
ICSF Query Algorithm (CSFIQA) 72, 1273
ICSF Query Facility (CSFIQF) 72
ICSF Query Facility2 (CSFIQF2) 72
ICSF Query Service (CSFIQF) 1278
ICSF Query Service (CSFIQF2) 1316
IEAAFFN (affinity) 9
installation-defined 15
invoking a 3
Key Data Set List (CSFKDSL and CSFKDL6) 59, 98
Key data set list (CSFKDSL and CSFKDSL6) 1225
Key data set metadata read (CSFKDMR and CSFKDMR6) 1239
Key Data Set Metadata Read (CSFKDMR and CSFKDMR6) 59, 98
Key data set metadata write (CSFKDMW and CSFKDMW6) 1246
Key Data Set Metadata Write (CSFKDMW and CSFKDMW6) 59, 98
Key Data Set Record Retrieve (CSFRRT and CSFRRT6) 1252
Key Data Set Update (CSFKDU and CSFKDU6) 1254
Key Encryption Translate (CSNBKET and CSNEKET) 204
key encryption translate callable service (CSNBKET) 38
key export (CSNBKEX) 38, 207
key generate (CSNBKGN) 38, 73, 211
Key Generate (CSNBKGN) 41
key generate2 (CSNBKGN2 and CSNEKGN2) 41, 42
Key Generate2 (CSNBKGN2 and CSNEKGN2) 224
key import (CSNBKIM) 39, 237
key part import (CSNBKPI) 39
Key Part Import (CSNBKPI) 242
key part import2 (CSNBKPI2 and CSNEKPI2) 41–43
Key Part Import2 (CSNBKPI2 and CSNEKPI2) 247

callable service *(continued)*
Key Test (CSNBKYT) 252
key test (CSNBKYT and CSNBKYTX) 39
Key Test Extended (CSNBKYTX) 267
Key Test2 (CSNBKYT2 and CSNEKYT2) 256
key token build (CSNBKTB) 39
Key Token Build (CSNBKTB) 271
key token build2 (CSNBKTB2 and CSNEKTB2) 41
Key Token Build2 (CSNBKTB2 and CSNEKTB2) 297
Key Token Wrap (CSFWRP) 1325
key translate (CSNBKTR) 39
Key Translate (CSNBKTR) 343
Key Translate2 (CSNBKTR2 and CSNEKTR2) 346
link edit step 12
MAC Generate (CSNBMGN or CSNBMGN1) 61, 555
MAC Generate2 (CSNBMGN2, CSNBMGN3, CSNEMGN2, and CSNEMGN3) 62, 562
MAC Verify (CSNBMVR or CSNBMVR1) 62, 567
MAC Verify2 (CSNBMVR2, CSNBMVR3, CSNEMVR2, and CSNEMVR3) 62, 574
MDC Generate (CSNBMDG or CSNBMDG1) 63, 579
multiple clear key import (CSNBCKM) 39
Multiple Clear Key Import (CSNBCKM) 355
multiple secure key import (CSNBSKM) 39
Multiple Secure Key Import (CSNBSKM) 358
one-way hash generate (CSNBOWH and CSNBOWH1) 63
One-Way Hash Generate (CSNBOWH, CSNEOWH and CSNBOWH1) 584
overview 3
PCI interface (CSFPCI) 1327
PIN change/unblock (CSNBPCU) 66
PIN Change/Unblock (CSNBPCU) 795
PKA decrypt (CSNDPKD) 59
PKA encrypt (CSNDPKE) 59
PKA key generate (CSNDPKG) 97
PKA Key Generate (CSNDPKG) 1131
PKA key import (CSNDPKI) 97
PKA Key Import (CSNDPKI) 1140
PKA key token build (CSNDPKB) 97, 1147
PKA key token change (CSNDKTC and CSNFKTC) 97
PKA Key Token Change (CSNDKTC) 1165
PKA Key Translate (CSNDPKT) 1169
PKA public key extract (CSNDPKX) 97, 1181
PKCS #11 Derive key (CSFPDVK) 1347
PKCS #11 Derive multiple keys (CSFPDMK) 1339
PKCS #11 Generate Keyed MAC (CSFPHMG) 1364
PKCS #11 Generate secret key (CSFPGSK) 1360
PKCS #11 One-Way Hash, Sign, or Verify (CSFPOWH) 1372
PKCS #11 Private Key Sign (CSFPPKS) 1379
PKCS #11 Private Key Structure Decrypt (CSFPPD2) 1429
PKCS #11 Private Key Structure Sign(CSFPPS2) 1432
PKCS #11 Pseudo-Random Function (CSFPPRF) 1386
PKCS #11 Public Key Structure Encrypt (CSFPPE2) 1434
PKCS #11 Public Key Structure Verify (CSFPPV2) 1437
PKCS #11 Public Key Verify (CSFPPKV) 1382
PKCS #11 Secret Key Decrypt (CSFPSKD) 1391
PKCS #11 Secret Key Encrypt (CSFPSKE) 1396
PKCS #11 Secret Key Reencrypt (CSFPSKR) 1403
PKCS #11 Set Attribute Value (CSFPSAV) 1389
PKCS #11 Unwrap Key (CSFPUWK) 1417
PKCS #11 Verify Keyed MAC (CSFPHMV) 1368
PKCS #11 Wrap Key (CSFPWPK) 1422
callable service *(continued)*
PKDS key record create (CSNDKRC) 1257
PKDS key record delete (CSNDKRD) 1259
PKDS key record read (CSNDKRR) 1261
PKDS key record read2 (CSNFKRR) 1261
PKDS key record write (CSNDKRW) 1263
PPKCS #11 Generate key pair (CSFPGKP) 1357
prohibit export (CSNBPEX) 40
Prohibit Export (CSNBPEX) 384
prohibit export extended (CSNBPEXX) 40
Prohibit Export Extended (CSNBPEXX) 386
Public Infrastructure Certificate (CSNDPIC and CSNFPIC) 1184
random number generate (CSNBRNG) 40
Random Number Generate (CSNBRNG) 388
Recover PIN from Offset (CSNBPFO) 806
Recover PIN From Offset (CSNBPFO) 66
remote key export (CSNDRKX) 40
Remote Key Export (CSNDRKX) 393
restrict key attribute (CSNBRKA and CSNERKA) 40, 42, 43
Restrict Key Attribute (CSNBRKA and CSNERKA) 402
Retained Key Delete (CSNDRKD) 1191
Retained Key List (CSNDRKL) 1194
secure key import (CSNBSKI) 40
Secure Key Import (CSNBSKI) 408
Secure Key Import2 (CSNBSKI2 and CSNESKI2) 412
Secure Messaging for Keys (CSNBSKY) 811
Secure Messaging for PINs (CSNBSPN) 816
security considerations 9
sequences 72
SET block compose (CSNDSBC) 100
SET Block Compose (CSNDSBC) 823
SET block decompose (CSNDSBD) 100
SET Block Decompose (CSNDSBD) 828
symmetric algorithm decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1) 501
symmetric key decipher (CSNBSYD and CSNBSYD1) 521
symmetric key encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1) 509
symmetric key encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1) 531
symmetric key export (CSNDSYX) 40
Symmetric Key Export (CSNDSYX) 417
Symmetric Key Export with Data (CSNDSXD) 426
symmetric key generate (CSNDSYG) 40
Symmetric Key Generate (CSNDSYG) 430
symmetric key import (CSNDSYI) 41
Symmetric Key Import (CSNDSYI) 439
Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2) 444
Symmetric MAC generate (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1) 589
Symmetric MAC Generate Callable Service (CSNBSMG, CSNBSMG1, CSNESMG and CSNESMG1) 62
Symmetric MAC verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1) 594
Symmetric MAC Verify Callable Service (CSNBSMV, CSNBSMV1, CSNESMV and CSNESMV1) 62
syntax 3
Token Record Create (CSFPTRC) 1407
Token Record Delete (CSFPTRD) 1411
Token Record List (CSFPTRL) 1413
TR-31 Create (CSNBT31C and CSNET31C) 938
TR-31 import (CSNBT31I and CSNET31I) 959

callable service *(continued)*
TR-31 Optional Data Build (CSNBT31O and CSNET31O) 998
TR-31 Optional Data Read (CSNBT31R and CSNET31R) 1001
TR-31 Parse (CSNBT31P and CSNET31P) 1004
TR-31 Translate (CSNBT31X and CSNET31X) 1007
TR-34 99
TR-34 Bind-Begin (CSNDT34B and CSNFT34B) 1072
TR-34 Bind-Complete (CSNDT34C and CSNFT34C) 1079
TR-34 Key Distribution (CSNDT34D and CSNFT34D) 1086
TR-34 Key Receive (CSNDT34R and CSNFT34R) 1098
transaction validation 66
Transaction Validation (CSNBTRV) 834
translating ciphertext 60
trusted block create (CSNDTBC) 41, 450
Unique Key Derive (CSNBUKD and CSNEUKD) 454
using key types and key forms 11
VISA CVV Service Generate (CSNBCSG) 838
VISA CVV Service Verify (CSNBCSV) 843
with ALETs (alternate entry point) 4
X9.9 data editing (CSNB9ED) 72, 1322
callable services
deprecated 10
CBC processing rule 487, 488, 496, 497
CCA
release levels 1723
CCA DES
control vectors 16
certificate length parameter
Remote Key Export callable service 395
certificate parameter
Remote Key Export callable service 396
certificate_parms parameter
Remote Key Export callable service 396
certificate_parms_length parameter
Remote Key Export callable service 396
chaining vector length parameter
One-Way Hash Generate callable service 588
Symmetric MAC generate callable service 592
Symmetric MAC verify callable service 597
chaining vector parameter
decipher callable service 488
encipher callable service 497
MAC Generate callable service 560
MAC verify callable service 572
MDC Generate callable service 582
One-Way Hash Generate callable service 588
symmetric MAC generate callable service 593
Symmetric MAC verify callable service 597
changing control vectors 1613
character/nibble conversion callable service (CSNBXBC and CSNBXCB)
format 1267
parameters 1267
syntax 1267
character/nibble conversion callable services (CSNBXBC and CSNBXCB)
overview 72
choosing between
CSNBCTT2 and CSNBCTT3 472
CSNBDEC and CSNBDEC1 485
CSNBENC and CSNBENC1 493
choosing between *(continued)*
CSNBMDG and CSNBMDG1 580
CSNBMGN and CSNBMGN1 556
CSNBMVR and CSNBMVR1 568
CSNBSYD and CSNBSYD1 522, 589
CSNBSYE and CSNBSYE1 533
CSNESAE and CSNESAE1 510
cipher block chaining (CBC)
mode 470
cipher feedback (CFB)
mode 470
cipher text id parameter
decipher callable service 507
encipher callable service 518
Cipher Text Translate2 Callable Service (CSNBCTT or CSNBCTT1)
using 76
ciphertext
Cryptographic Variable Encipher callable service 129
deciphering 60, 469
encoding 499
field 531, 542
translating 60, 471
ciphertext id parameter
decipher callable service 489, 529
encipher callable service 498, 539
ciphertext parameter
decipher callable service 487
decode callable service 492
encipher callable service 497
encode callable service 501
CKDS (cryptographic key data set)
record format 1537, 1538
CKDS key record create callable service (CSNBKRC)
format 1202
overview 57
parameters 1202
syntax 1202
CKDS key record delete callable service (CSNBKRD)
format 1205
parameters 1205
syntax 1205
CKDS key record read callable service (CSNBKRR)
format 1207
overview 58
parameters 1207
syntax 1207
CKDS key record write callable service (CSNBKRW)
format 1213
overview 58
parameters 1213
syntax 1213
clear key
deciphering data with 490
definition 28
enciphering 408
enciphering data with 499
encoding and decoding data with 60
protecting 469
clear key import callable service (CSNBCKI)
overview 37
Clear Key Import callable service (CSNBCKI)
format 116
parameters 116

Clear Key Import callable service (CSNBCKI) *(continued)*
syntax 116
clear key length parameter
multiple clear key import callable service 361
Multiple Clear Key Import callable service 357
clear key parameter
Clear Key Import callable service 117
decode callable service 491
encode callable service 500
multiple clear key import callable service 361
Multiple Clear Key Import callable service 357
Secure Key Import callable service 409
Clear PIN Encrypt callable service (CSNBCPE)
format 625
syntax 625
Clear PIN Encrypt service (CSNBCPE)
parameters 625
clear PIN generate alternate callable service (CSNBCPA)
overview 65
Clear PIN Generate Alternate callable service (CSNBCPA)
format 634
parameters 634
syntax 634
Clear PIN Generate callable service (CSNBPGN)
format 629
parameters 629
syntax 629
clear PIN generate key identifier parameter 635
Clear PIN Generate key identifier parameter
Clear PIN Generate callable service 630
clear text id parameter
decipher callable service 489, 507, 529
encipher callable service 498, 518, 539
clear text parameter
decipher callable service 489
decode callable service 492
encipher callable service 495
encode callable service 501
code conversion callable service (CSNBXEA and CSNBXAE)
format 1269
parameters 1269
syntax 1269
code conversion callable services (CSNBXEA and CSNBXAE)
overview 72
code table parameter
character/nibble conversion callable service 1268
code conversion callable service 1270
coding examples
C 1619
COBOL 1621
High Level Assembler 1623
PL/I 1625
Rexx 1627
compliance mode 11
compliance mode impact 1695
Compliant-tagged key tokens 19, 104
contact
z/OS 1727
control information
for Digital Signature Generate 1111–1113
for Digital Signature Verify 1121–1123
for Diversified Key Generate 152
for Key Test 253
control information *(continued)*
for Key Test Extended 269
for MAC Generate 558
for MAC verify 596
for MAC Verify 570
for MDC Generate 582
for Multiple Clear Key Import 356, 357
for multiple secure key import 642, 1002
for Multiple Secure Key Import 360, 361
for One-Way Hash Generate 586, 587
for PKA key token build 1150
for symmetric algorithm encipher 3, 503, 504, 512–514
for symmetric key encipher 524, 525, 535, 536
for symmetric key import 446
for Symmetric Key Import 440, 441
for symmetric MAC generate 592
Random Number Generate callable service 390, 391
control vector
Base Bit Map 1605
description 1603
key form bits, 'fff' 1608
value 1603–1605
control vector generate (CSNBCVG)
parameters 118
control vector generate callable service (CSNBCVG)
format 118
overview 38
syntax 118
control vector parameter
control vector generate callable service 123
control vector translate callable service (CSNBCVT)
overview 38
parameters 546, 551, 1203, 1209, 1215
Control Vector Translate callable service (CSNBCVT)
format 124
parameters 124
syntax 124
control vectors
CCA DES 16
control vectors, changing 1613
Coordinated KDS Administration callable service (CSFCRC and CSFCRC6) 1217
coordinated KDS administration callable services (CSFCRC and CSFCRC6)
overview 58
cryptographic feature
description xliii
cryptographic hardware engines 1715
cryptographic key data set (CKDS)
held keys 23
storing keys 37, 57, 115
cryptographic software 1715
Cryptographic Variable Encipher (CSNBCVE)
parameters 128
cryptographic variable encipher callable service (CSNBCVE)
overview 38
Cryptographic Variable Encipher callable service (CSNBCVE)
format 128
syntax 128
CRYSTALS-Dilithium digital signature algorithm 93
CSFACEE callable service 1321
CSFCRC callable service 1217
CSFCRC6 callable service 1217

CSFIQA callable service 1273
CSFIQF callable service 1278
CSFIQF2 1278, 1316, 1321
CSFIQF2 callable service 1316
CSFKDMR callable service 1239
CSFKDMR6 callable service 1239
CSFKDMW callable service 1246
CSFKDMW6 callable service 1246
CSFKDSL callable service 1225
CSFKDSL6 callable service 1225
CSFKDU callable service 1254
CSFKDU6 callable service 1254
CSFMPS callable service 1221
CSFMPS6 callable service 1221
CSFPCI callable service 1327
CSFPDMK callable service 1339
CSFPDVK callable service 1347
CSFPGAV callable service 1355
CSFPGKP callable service 1357
CSFPGSK callable service 1360
CSFPHMG callable service 1364
CSFPHMV callable service 1368
CSFPOWH callable service 1372
CSFPPD2 callable service 1429
CSFPPE2 callable service 1434
CSFPPKS callable service 1379
CSFPPKV callable service 1382
CSFPPRF callable service 1386
CSFPPS2 callable service 1432
CSFPPV2 callable service 1437
CSFPSAV callable service 1389
CSFPSKD callable service 1391
CSFPSKE callable service 1396
CSFPSKR callable service 1403
CSFPTRC callable service 1407
CSFPTRD callable service 1411
CSFPTRL callable service 1413
CSFPUWK callable service 1417
CSFPWPK callable service 1422
CSFRRT callable service 1252
CSFRRT6 callable service 1252
CSFSTAT callable service 1271
CSFxxxx format 4
CSNB9ED callable service 1322
CSNBAPG callable service 620
CSNBCKC and CSNECKC callable services 640
CSNBCKI callable service 116
CSNBCKM callable service 355
CSNBCPA callable service 634
CSNBCPE callable service 624
CSNBCSG callable service 838
CSNBCSV callable service 843
CSNBCTT2, CSNBCTT3, CSNECTT2, or CSNECTT3 callable service 471
CSNBCVE callable service 128
CSNBCVG callable service 118
CSNBCVT callable service 124
CSNBDCO callable service 490
CSNBDDK and CSNEDDK callable services 170
CSNBDEC or CSNBDEC1 callable service 484
CSNBDKG callable service 151
CSNBDKM callable service 133
CSNBDKX callable service 130
CSNBECO callable service 499
CSNBENC or CSNBENC1 callable service 492
CSNBEPG callable service 671
CSNBHMG, CSNEHMG, CSNBHMG1 and CSNEHMG1 callable services 545
CSNBHMV, CSNEHMV, CSNBHMV1 and CSNEHMV1 callable services 550
CSNBKET and CSNEKET callable services 204
CSNBKEX callable service 207
CSNBKGN callable service 211
CSNBKGN2 and CSNEKGN2 callable services 224
CSNBKIM callable service 237
CSNBKPI callable service 242
CSNBKPI2 and CSNEKPI2 callable services 247
CSNBKRC callable service 1202
CSNBKRC2 and CSNEKRC2 callable services 1203
CSNBKRD callable service 1205
CSNBKRR callable service 1207
CSNBKRR2 and CSNEKRR2 callable services 1208
CSNBKRW callable service 1213
CSNBKRW2 and CSNEKRW2 callable services 1215
CSNBKTB callable service 271
CSNBKTB2 and CSNEKTB2 callable services 297
CSNBKTR callable service 343
CSNBKTR2 and CSNEKTR2 callable services 346
CSNBKYT callable service 252
CSNBKYT2 and CSNEKYT2 callable services 256
CSNBKYTX callable service 267
CSNBMDG or CSNBMDG1 callable service 579
CSNBMGN or CSNBMGN1 callable service 555
CSNBMGN2, CSNBMGN3, CSNEMGN2, and CSNEMGN3 callable services 562
CSNBMVR or CSNBMVR1 callable service 567
CSNBMVR2, CSNBMVR3, CSNEMVR2, and CSNEMVR3 callable services 574
CSNBOWH, CSNEOWH and CSNBOWH1 callable services 584
CSNBPCU callable service 795
CSNBPEX callable service 384
CSNBPEXX callable service 386
CSNBPFO callable service 806
CSNBPGN callable service 629
CSNBPTR callable service 678, 685
CSNBPTRE callable service 701
CSNBPVR callable service 713
CSNBPVR2 callable service 720
CSNBRKA and CSNERKA callable services 402
CSNBRNG callable service 388
CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1 501
CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1 callable service 509
CSNBSKI callable service 408
CSNBSKI2 and CSNESKI2 callable services 412
CSNBSKM callable service 358
CSNBSKY callable service 811
CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1 callable service 589
CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1 callable service 594
CSNBSPN callable service 816
CSNBSYD and CSNBSYD1 callable service 521
CSNBSYE and CSNBSYE1 callable service 531
CSNBT31C and CSNET31C callable services 938
CSNBT31I and CSNET31I callable services 959
CSNBT31O and CSNET31O callable services 998

CSNBT31P and CSNET31P callable services 1004
CSNBT31R and CSNET31R callable services 1001
CSNBT31X and CSNET31X callable services 1007
CSNBTRV callable service 834
CSNBUKD and CSNEUKD callable service 454
CSNBXAE callable service 1269
CSNBXBC callable service 1267
CSNBXCB callable service 1267
CSNBXEA callable service 1269
CSNBxxxx format 4
CSNDDSG callable service 1110
CSNDDSV callable service 1119
CSNDEDH and CSNFEDH callable services 182
CSNDKRC callable service 1257
CSNDKRD callable service 1259
CSNDKRR callable service 1261
CSNDKRW callable service 1263
CSNDKTC callable service 1165
CSNDPIC and CSNFPIC callable service 1184
CSNDPKB callable service 1147
CSNDPKD callable service 365
CSNDPKE callable service 374
CSNDPKG callable service 1131
CSNDPKI callable service 1140
CSNDPKT callable service 1169
CSNDPKX callable service 1181
CSNDRKD callable service 1191
CSNDRKL callable service 1194
CSNDSBC callable service 823
CSNDSBD callable service 828
CSNDSXD callable service 426
CSNDSYG callable service 430
CSNDSYI callable service 439
CSNDSYI2 callable service 444
CSNDSYX callable service 417
CSNDT34B and CSNFT34B callable services 1072
CSNDT34C and CSNFT34C callable services 1079
CSNDT34D and CSNFT34D callable services 1086
CSNDT34R and CSNFT34R callable services 1098
CSNDTBC callable service 450
CSNECKI 116
CSNECKM 355
CSNEKGN 212
CSNEOWH 584
CSNERNG 389, 1271
CSNFKRC 1258
CSNFKRD 1259
CSNFKRR callable service 1261
CSNFPKB 1149
CSNFPKD 365
CSNFPKE 374
CSNFPKG 1132
CSNFPKX 1181
CSNFRKD 1191
CSNFRKL 1194
CSNFSYI2 callable service 444
CUSP processing rule 487, 488, 496, 497, 1644
CVV Key Combine callable service (CSNBCKC and CSNECKC) 640

D

data
  deciphering 484
data *(continued)*
  enciphering 492
  enciphering and deciphering 59
  encoding and decoding 60
  protecting 469
data array parameter
  Clear PIN Generate Alternate callable service 638
  Clear PIN Generate callable service 631
  encrypted PIN generate callable service 674
  Encrypted PIN Verify callable service 717
data integrity
  ensuring 60
  verifying 543
data key
  exporting 130
  importing 116
  reenciphering 130
data key export callable service (CSNBDKX)
  overview 38
Data Key Export callable service (CSNBDKX)
  format 130
  parameters 130
  syntax 130
data key import callable service (CSNBDKM)
  overview 38
Data Key Import callable service (CSNBDKM)
  format 133
  parameters 133
  syntax 133
data length parameter
  Digital Signature Verify callable service 1124
  Diversified Key Generate callable service 156
data parameter
  Digital Signature Verify callable service 1124
data space
  callable services that use data in data spaces 4
data_decrypting_key identifier parameter
  Diversified Key Generate callable service 157
decipher callable service (CSNBDEC or CSNBDEC1)
  format 485
  syntax 485
deciphering
  data 469, 484
  data with clear key 490
decode callable service (CSNBDCO)
  format 490
  parameters 490
  syntax 490
deprecated callable services 10
Derive ICC MK
  description 67
Derive ICC MK (CSNBDCM)
  overview 136
Derive Session Key
  description 68
Derive Session Key (CSNBDSK)
  overview 143
DES algorithm 469
DES internal key token format 1502
Digital Signature Generate callable service (CSNDDSG)
  format 1110
  overview 96
  parameters 1110
  syntax 1110

Digital Signature Verify callable service (CSNDDSV)
format 1119
overview 96
parameters 1119
syntax 1119
diversified key generate callable service (CSNBDKG)
overview 38
Diversified Key Generate callable service (CSNBDKG)
format 151
parameters 151
syntax 151
Diversified Key Generate2 (CSNBDKG2)
overview 162
DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG)
overview 850
DK Migrate PIN callable service (CSNBDMP)
overview 857
DK PAN Modify in Transaction callable service (CSNBDPMT)
overview 863
DK PAN Translate (CSNBDPT and CSNEDPT)
overview 871
DK PIN Change callable service (CSNBDPC)
overview 878
DK PIN methods 849
DK PIN Verify callable service (CSNBDPV)
overview 892
DK PRW Card Number Update (CSNBDPNU and CSNEDPNU)
overview 897
DK PRW Card Number Update2 (CSNBDCU2 and CSNEDCU2)
overview 904
DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG)
overview 912
DK Random PIN Generate callable service (CSNBDRPG)
overview 916
DK Random PIN Generate2 callable service (CSNBDRG2)
overview 922
DK Regenerate PRW (CSNBDRP and CSNEDRP)
overview 930

E

EBCDIC to ASCII conversion
table 1657
ECC algorithm 93
ECI-1 683
ECI-2 PIN block format 610, 1633
ECI-3 PIN block format 610, 1633
ECI-4 683
electronic code book (ECB)
mode 470
Elliptic Curve Cryptography (ECC) 93
EMV Scripting Service
description 68
EMV Scripting Service (CSNBESC)
overview 646
EMV simplification 67
EMV Transaction (ARQC/ARPC) Service
description 68
EMV Transaction (ARQC/ARPC) Service (CSNBEAC)
EMV Transaction (ARQC/ARPC) Service (CSNBEAC) *(continued)*
overview 657
EMV Verification
description 69
EMV Verification Functions (CSNBEVF)
overview 665
encipher callable service (CSNBENC or CSNBENC1)
format 494
parameters 494
syntax 494
enciphered
key 211, 410, 469
under master key 237
enciphering
data 469, 492
string with clear key 499
encode callable service (CSNBECO)
format 499
parameters 499
syntax 499
encrypted PIN block parameter
Clear PIN Generate Alternate callable service 636
Encrypted PIN Verify callable service 716
encrypted PIN generate callable service (CSNBEPG)
format 672
syntax 672
encrypted PIN generate service (CSNBEPG)
parameters 672
encrypted PIN translate callable service (CSNBPTR)
extraction rules 1633
format 678
parameters 679
syntax 678
encrypted PIN translate enhanced callable service (CSNBPTRE)
format 702
parameters 703
syntax 702
encrypted PIN translate2 callable service (CSNBPTR2)
format 686
parameters 686
syntax 686
encrypted PIN verification callable service (CSNBPVR)
extraction rules 1633
encrypted PIN verify callable service (CSNBPVR)
format 713
parameters 713
syntax 713
encrypted PIN verify2 callable service (CSNBPVR2)
format 720
parameters 721
syntax 720
ensuring data integrity and authenticity 60
EX key form 74
examples of callable services 1619
EXEX key form 76
exit data 7
exit data length 7
exit, installation 7
exportable key form
definition 16
generating 74
exporter key identifier parameter
key export callable service 209

exporter key-encrypting key
  any DES key 207
  enciphering data key 130
exporting keys
  trusted blocks 50
external key token
  PKA 103
extra_data parameter
  Remote Key Export callable service 400
extra_data_length parameter
  Remote Key Export callable service 400
extraction rules, PIN 1633

F

FEATURE=CRYPTO keyword
  SCHEDULE macro 9
feedback xlvii
Field Level Decipher (CSNBFLD)
  overview 730
Field Level Encipher (CSNBFLE)
  overview 739
form parameter
  Random Number Generate callable service 389
format control 612
Format Preserving Algorithms Decipher (CSNBFFXD and CSNEFFXD) 67
Format Preserving Algorithms Decipher (CSNBFFXD)
  overview 749
Format Preserving Algorithms Encipher (CSNBFFXE and CSNEFFXE) 67
Format Preserving Algorithms Encipher (CSNBFFXE)
  overview 754
Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT) 67
Format Preserving Algorithms Translate (CSNBFFXT)
  overview 760
format preserving encryption 614
Format preserving encryption 66
formats, PIN 64
formatting methods 1109
FPE Decipher (CSNBFPED)
  overview 767
FPE Encipher (CSNBFPEE)
  overview 776
FPE Translate (CSNBFPET)
  overview 786
functions of
  cryptographic keys 15
  ICSF 15

G

GBP-PIN algorithm 716
GBP-PINO algorithm 716
Generate Issuer MK
  description 69
Generate Issuer MK callable service (CSNBGIM)
  overview 197
generated key identifier 1 parameter
  key generate callable service 218
generated key identifier 2 parameter
  key generate callable service 219
generated key identifier parameter
  Diversified Key Generate callable service 157
generating encrypted keys 211
generating key identifier parameter
  Diversified Key Generate callable service 155
generating keys
  remote key export 53
German Banking Pool PIN algorithm 1636
get attribute value callable service (CSFPGAV)
  format 1355
  parameters 1355
  syntax 1355

H

hardware support 1716
hash length parameter
  One-Way Hash Generate callable service 588
hash parameter
  One-Way Hash Generate callable service 588
HEXDIGIT PIN extraction method keyword 610–612
high-level languages 4
how to use information xliii

I

IBM 3624 629, 713
IBM 4700 processing rule 1644
IBM GBP 629, 713
IBM-4700 PIN block format 1633
IBM-PIN algorithm 716
IBM-PINO algorithm 716
ICSF
  functions 15
ICSF Multi-Purpose Service (CSFMPS and CSFMPS6) 1221
ICSF Query Algorithm (CSFIQA)
  parameters 1273
  syntax 1273
ICSF Query Algorithm (CSFIQA))
  format 1273
ICSF Query Algorithm Service (CSFIQA)
  overview 72
ICSF Query Facility (CSFIQF)
  parameters 1278
  syntax 1278
ICSF Query Facility (CSFIQF))
  format 1278
ICSF Query Facility Service (CSFIQF)
  overview 72
ICSF Query Facility2 (CSFIQF2)
  overview 72
  parameters 1316
  syntax 1316
ICSF Query Facility2(CSFIQF2)
  format 1316
IEAAFFN callable service (affinity) 9
IM key form 74
IMEX key form 75
IMIM key form 74
importable key form
  definition 16

importable key form *(continued)*
generating 74
imported key identifier length parameter
Multiple Secure Key Import callable service 362
imported key identifier parameter
Multiple Secure Key Import callable service 362
importer key identifier parameter
key import callable service 239
Secure Key Import callable service 410
importer key-encrypting key
enciphering clear key 408, 410
importer_key_identifier parameter
Remote Key Export callable service 398
importer_key_length parameter
Remote Key Export callable service 398, 399
importing a non-exportable key 386
INBK PIN 604, 629
INBK-PIN 713
Information Protection System (IPS) 1645
initial chaining vector (ICV)
description 470, 1643
initialization vector parameter
Cryptographic Variable Encipher callable service 129
decipher callable service 487
encipher callable service 495
Key Token Build callable service 279
input KEK key identifier parameter
Key Translate callable service 344
input PAN data parameter
Encrypted PIN Verify2 callable service 727
input PIN block length parameter
Encrypted PIN Verify2 callable service 726
input PIN block parameter
Encrypted PIN Verify2 callable service 726
input PIN key identifier length parameter
encrypted PIN translate enhanced callable service 705
input PIN key identifier parameter
encrypted PIN translate enhanced callable service 706
input PIN profile length parameter
encrypted PIN translate enhanced callable service 708
Encrypted PIN Verify2 callable service 725
input PIN profile parameter
Clear PIN Generate Alternate callable service 636
encrypted PIN translate callable service 680
encrypted PIN translate enhanced callable service 708
Encrypted PIN Verify callable service 715
Encrypted PIN Verify2 callable service 725
input PIN-encrypting key identifier length parameter
Encrypted PIN Verify2 callable service 723
input PIN-encrypting key identifier parameter
encrypted PIN translate callable service 679
Encrypted PIN Verify callable service 714
Encrypted PIN Verify2 callable service 723
input_block parameter
trusted block create callable service 452
input_block_identifier parameter
trusted block create callable service 452
installation exit
post-processing 7
preprocessing 7
installation-defined callable service 15
Integrated Cryptographic Service Facility (ICSF)
description xliii
Integrity 1590
Interbank PIN 86, 604, 629, 713
internal key token
aes; 1502
DES 1502, 1503
invocation requirements 9
IPINENC key type 679
IPS processing rule 487, 488, 496, 497, 1645
ISO-0 PIN block format 610
ISO-1 PIN block format 610, 1632
ISO-2 PIN block format 610, 1632
ISO-3 PIN block format 610, 1632
ISO-4 PIN block format 1632

J

JCL statements, sample 12

K

KEK key identifer parameter
Control Vector Translate callable service 125
KEK key identifier 1 parameter
key generate callable service 217
KEK key identifier 2 parameter
key generate callable service 218
KEK key identifier parameter
Key Test Extended callable service 270
Prohibit Export Extended callable service 387
key array parameter
Control Vector Translate callable service 125
key array right parameter
Control Vector Translate callable service 125
Key data set list (CSFKDSL and CSFKDSL6) 1225
Key Data Set management
callable services 1199
Key data set metadata read (CSFKDMR and CSFKDMR6) 1239
Key data set metadata write (CSFKDMW and CSFKDMW6) 1246
Key Data Set Record Retrieve (CSFRRT and CSFRRT6) 1252
Key Data Set Update (CSFKDU and CSFKDU6) 1254
key encrypting key identifier parameter 434
key encryption translate callable service (CSNBKET)
overview 38
key export callable service (CSNBKEX)
format 207
overview 38
parameters 207
syntax 207
key flow 17
key form
combinations for a key pair 220
combinations with key type 220
definition 16
exportable 16, 17
importable 16, 17
operational 16
key form parameter
key generate callable service 213
Secure Key Import callable service 410
key generate callable service (CSNBKGN)
format 211
overview 37

key generate callable service (CSNBKGN) *(continued)*
parameters 211
syntax 211
using 73
key generator utility program (KGUP)
description 37
key identifier
PKA keys 103
key identifier length parameter
Multiple Clear Key Import callable service 357
Symmetric MAC generate callable service 591
symmetric MAC verify callable service 596
key identifier parameter
Clear Key Import callable service 117
decipher callable service 486
encipher callable service 495
Key Test callable service 254
Key Test Extended callable service 269
MAC Generate callable service 557
Multiple Clear Key Import callable service 357
Secure Key Import callable service 410
Symmetric MAC generate callable service 591
symmetric MAC verify callable service 596
key import callable service (CSNBKIM)
format 237
overview 39
parameters 237
syntax 237
key label
security considerations 9
key length parameter
key generate callable service 214
key pair 220
key part import callable service (CSNBKPI)
overview 39
Key Part Import callable service (CSNBKPI)
format 242
parameters 242
syntax 242
key record delete callable service (CSNBKRD)
overview 58
key test callable service (CSNBKYT and CSNBKYTX)
overview 39
Key Test callable service (CSNBKYT)
parameters 252
Key Test callable services (CSNBKYT)
format 252
syntax 252
Key Test Extended callable service (CSNBKYTX)
parameters 267
Key Test Extended callable services (CSNBKYTX)
syntax 267
Key Test Extended callable services (CSNBKYTX)
format 267
key token
aes; internal 1502
DES
internal 1502
DES internal 1503
PKA 100
key token build callable service (CSNBKTB and CSNEKTB)
overview 41
key token build callable service (CSNBKTB)
overview 39
Key Token Build callable service (CSNBKTB)
format 271
parameters 271
syntax 271
Key Token Wrap (CSFWRP)
overview 1325
Key Translate (CSNBKTR)
parameters 343
key translate callable service (CSNBKTR)
overview 39
Key Translate callable service (CSNBKTR)
format 343
syntax 343
Key Translate2 callable service (CSNBKTR2 and CSNEKTR2)
format 347
parameters 347
syntax 347
key type 1 74–76
key type 1 parameter
key generate callable service 216
key type 2 74–76
key type 2 parameter
key generate callable service 216
key type parameter
key export callable service 208
key import callable service 238
Key Token Build callable service 273
Secure Key Import callable service 409
key value structure length parameter 1153
key value structure parameter 1153
key_check_length parameter
Remote Key Export callable service 400
key_check_parameters parameter
Remote Key Export callable service 400
key_check_parameters_length parameter
Remote Key Export callable service 400
key_check_value parameter
Remote Key Export callable service 401
key_identifier
Random Number Generate callable service 391
key_identifier_length
Random Number Generate callable service 391
key-encrypting key
exporter 130, 207
importer 408
keyboard
navigation 1727
PF keys 1727
shortcut keys 1727
keys
clear 28, 408
create
values for keys 41
creating 11
cryptographic, functions of 15
data key
exporting 130
importing 116
reenciphering 130
double-length 75, 76
enciphered 410
export
values for keys 40

keys *(continued)*
forms 16
generating
encrypted 211
values for keys 40
held in applications 23
held in CKDS 23
managing 115
pair 74–76
parity 116
PIN-encrypting key 678, 685, 701
PKA master
Key Management Master Key (KMMK) 94
Signature Master Key (SMK) 94
possible forms 38
protecting 469
reenciphered 237
reenciphering 207
separation 16
single-length 74
types of 21, 23, 26, 27
using 11
VISA PVV
generating 634

L

languages, high-level 4
large data object 1644
linking callable services 12
local enciphered key token parameter 435

M

MAC
Generate callable service 61
Generate2 callable service 62
length keywords 558, 570, 592, 596
managing 61
Verify callable service 62
Verify2 callable service 62
MAC Generate callable service (CSNBMGN or CSNBMGN1)
format 556
parameters 557
syntax 556
mac length parameter
Symmetric MAC generate callable service 593
symmetric MAC verify callable service 598
mac parameter
MAC Generate callable service 560
MAC verify callable service 572
Symmetric MAC generate callable service 593
symmetric MAC verify callable service 598
MAC Verify callable service (CSNBMVR or CSNBMVR1)
format 568
parameters 569
syntax 568
managing keys 115
managing the CKDS
callable services 57
managing TR-31 symmetric keys 937
managing TR-34 symmetric keys 1059
managing X9.143 (TR-31) symmetric keys 937
mask array left parameter
Control Vector Translate callable service 125
mask array preparation 1613
mask array right parameter
Control Vector Translate callable service 125
master key
enciphered key 237
possible effect on internal key tokens 17
master key verification pattern 1501
MDC
generate callable service 63
length keywords 582
managing 63
mdc parameter
MDC Generate callable service 583
message authentication
definition 61, 62
message authentication code (MAC)
description 543
generating 543, 555, 589
verifying 543, 567, 594
messages
authenticating 543
metadata
key data set records 1199
migration consideration
return codes from PCF macros 7
mode, special secure 10
modes of operation 469
modification detection
definition 63
modification detection code (MDC)
generating 544, 579
verifying 544
multiple clear key import callable service (CSNBCKM)
overview 39, 42
Multiple Clear Key Import callable service (CSNBCKM)
format 355
parameters 355
syntax 355
multiple secure key import callable service (CSNBSKM and CSNESKM)
overview 42
multiple secure key import callable service (CSNBSKM)
overview 39
Multiple Secure Key Import callable service (CSNBSKM)
format 359
parameters 359
syntax 359

N

navigation
keyboard 1727
null key token 17
Null key tokens 1501

O

object ion key (OPK) 1599
one-way hash generate callable service (CSNBOWH and CSNBOWH1)
overview 63

One-Way Hash Generate callable service (CSNBOWH, CSNEOWH and CSNBOWH1)
format 584
parameters 584
syntax 584
OP key form 74
operational key form
definition 16
generating 73
OPEX key form 75
OPIM key form 74
OPINENC key type 679
OPK, object protection key 1599
OPOP key form 74
output chaining vector (OCV)
description 1643
output KEK key identifier parameter
Key Translate callable service 344
output PIN key identifier length parameter
encrypted PIN translate enhanced callable service 706
output PIN key identifier parameter
encrypted PIN translate enhanced callable service 707
output PIN profile parameter
encrypted PIN translate callable service 682
encrypted PIN translate enhanced callable service 709
output PIN-encrypt translation key identifier parameter
encrypted PIN translate callable service 679
overview of callable services 3

P

pad character parameter
encipher callable service 497
Key Token Build callable service 279
pad digit
format 612
PADDIGIT PIN extraction method keyword 610–612
padding schemes 485, 493
PADEXIST PIN extraction method keyword 610–612
pair of keys 74–76
PAN data in parameter
encrypted PIN translate callable service 680
encrypted PIN translate enhanced callable service 709
PAN data length parameter
encrypted PIN translate enhanced callable service 709
PAN data out parameter
encrypted PIN translate callable service 682
PAN data parameter
Clear PIN Encrypt callable service 627
Clear PIN Generate Alternate callable service 636
encrypted PIN generate callable service 675
encrypted PIN verify callable service 715
PAN key identifier length parameter
encrypted PIN translate enhanced callable service 707
PAN key identifier parameter
encrypted PIN translate enhanced callable service 708
parameter
attribute definitions 5
definitions 6
direction 5
exit data 7
exit data length 7
reason code 6
return code 6
parameter *(continued)*
type 5
parity of key
adjusting 253, 269
EVEN 390
ODD 390
Payload Format 21
PCF
key separation 16
macros 7
migration consideration 7
PCI interface callable service (CSFPCI)
parameters 1327
syntax 1327
performance considerations 9
personal account number (PAN)
for encrypted PIN translate 680
for encrypted PIN translate enhanced 709
for encrypted PIN verify 715
personal authentication
definition 64
personal identification number (PIN)
3624 PIN generation algorithm 1635
3624 PIN verification algorithm 1638
algorithm value 638, 716
algorithms 64, 604, 629
block format 605, 678, 685, 701
clear PIN encrypt callable service 64
clear PIN generate alternate callable service 65
Clear PIN Generate Alternate callable service 634
definition 64
description 601, 849
detailed algorithms 1635
encrypted generation callable service 65
encrypting key 605, 678, 685, 701
extraction rules 1633
formats 64
GBP PIN verification algorithm 1640
generating
from encrypted PIN block 603
generation callable service 65, 629
German Banking Pool PIN algorithm 1636
managing 64
PIN offset generation algorithm 1637
PVV generation algorithm 1641
PVV verification algorithm 1642
translating 604
translation callable service 65, 678, 685, 701
translation of, in networks 602
using 602
verification callable service 66, 713, 720
verifying 603, 713, 720
VISA PIN algorithm 1641
PIN block format
3621 1633
3624 1633
additional names 683
ANSI X9.8 1632
detail 1632
ECI-2 1633
ECI-3 1633
format values 610
IBM-4700 1633
ISO-1 1632

PIN block format *(continued)*
ISO-2 1632
ISO-3 1632
ISO-4 1632
PIN extraction method keywords 610
VISA-2 1632
VISA-3 1633
PIN block in parameter
encrypted PIN translate callable service 680
PIN block out parameter
encrypted PIN translate callable service 682
PIN Change/Unblock
format 796
syntax 796
PIN Change/Unblock (CSNBPCU)
parameters 797
PIN check length parameter
Clear PIN Encrypt callable service 626
Clear PIN Generate callable service 631
Encrypted PIN Verify callable service 717
Encrypted PIN Verify2 callable service 723
PIN encryption key identifier parameter 635
PIN encryting key identifier parameter
Clear PIN Encrypt callable service 625
PIN generating key identifier parameter
encrypted PIN generate callable service 673
PIN length parameter
clear PIN generate callable service 626
Clear PIN Generate callable service 631
encrypted PIN generate callable service 674
PIN notation 1631
PIN profile
description 680, 715
PIN profile parameter
encrypted PIN generate callable service 675
PIN validation value (PVV) 629
PIN verifying key identifier parameter
Encrypted PIN Verify callable service 715
PINBLOCK PIN extraction method keyword 610–612
PINLEN04 PIN extraction method keyword 610–612
PINLEN12 PIN extraction method keyword 610–612
PKA Decrypt callable service 365
PKA decrypt callable service (CSNDPKD)
overview 59
PKA Encrypt callable service 374
PKA encrypt callable service (CSNDPKE)
overview 59
PKA external key token 103
PKA Key Generate callable service (CSNDPKG)
format 1131
parameters 1131
syntax 1131
PKA key import callable service (CSNDPKI)
overview 97
PKA Key Import callable service (CSNDPKI)
format 1140
parameters 1140
syntax 1140
PKA key token 100
PKA key token build callable service (CSNDPKB)
format 1147
overview 97
parameters 1147
syntax 1147
PKA Key Token Change (CSNDKTC)
parameters 1166
PKA key token change callable service (CSNDKTC and CSNFKTC)
overview 97
PKA Key Token Change callable service (CSNDKTC) 1165
PKA key translate callable service (CSNDPKT)
parameters 1169
syntax 1169
PKA Key Translate callable service (CSNDPKT)
format 1169
PKA public key extract callable service (CSNDPKX)
format 1181
overview 97
parameters 1181
syntax 1181
PKA public key identifier length parameter 1123
PKA public key identifier parameter 1123
PKA92 key format and encryption process 1649
pkcs #11
using 109
PKCS #11
callable services 109, 1337, 1429
key structure callable services 1429
objects 1339
tokens 1339
using 1339, 1429
PKCS #11 Private Key Structure Decrypt (CSFPPD2)
format 1429
parameters 1429
syntax 1429
PKCS #11 Private Key Structure Sign (CSFPPS2)
parameters 1432
syntax 1432
PKCS #11 Private Key Structure Sign(CSFPPS2)
format 1432
PKCS #11 Public Key Structure Encrypt (CSFPPE2)
format 1434
parameters 1434
syntax 1434
PKCS #11 Public Key Structure Verify (CSFPPV2)
format 1437
parameters 1437
syntax 1437
PKDS key record create callable service (CSNDKRC)
format 1258
parameters 1258
syntax 1258
PKDS key record delete callable service (CSNDKRD)
format 1259
parameters 1260
syntax 1259
PKDS key record read callable service (CSNDKRR)
format 1261
parameters 1262
syntax 1261
PKDS key record read2 callable service (CSNFKRR) 1261
PKDS key record write callable service (CSNDKRW)
format 1263
parameters 1264
syntax 1263
plaintext
enciphering 469
encoding 499

plaintext *(continued)*
field 531, 542
plaintext parameter
Cryptographic Variable Encipher callable service 129
post-processing exit 7
preprocessing exit 7
privacy 59
private key name length parameter 1162
private key name parameter 1162
processing rule
4700-PAD 487, 488, 496, 497
ANSI INCITS 106 1643
ANSI X9.23 487, 488, 496, 497, 1644
CBC 487, 488, 496, 497
cipher 1643
cipher last block 1644
CUSP 1644
CUSP/IPS 487, 488, 496, 497
decipher 487, 488
encipher 496, 497
GBP-PIN 631
GBP-PINO 631
IBM 4700 1644
IBM-PIN 631
IBM-PINO 631
INBK-PIN 631
IPS 1645
segmenting 1644
VISA-PVV 631
Prohibit Export (CSNBPEX) 384
prohibit export callable service (CSNBPEX)
overview 40
Prohibit Export callable service (CSNBPEX)
format 384
syntax 384
prohibit export extended callable service (CSNBPEXX)
overview 40
Prohibit Export Extended callable service (CSNBPEXX)
format 386
parameters 386
syntax 386
protecting data and keys 469

R

RACF authorization 9
random number generate callable service (CSNBRNG)
overview 40
Random Number Generate callable service (CSNBRNG)
format 388
parameters 388
syntax 388
random number parameter
Key Test callable service 254
Key Test Extended callable service 270
Random Number Generate callable service 392
random_number_length
Random Number Generate callable service 391
reason codes 6, 11
reason codes for ICSF
for return code 0 (0) 1442
for return code 10 (16) 1499
for return code 4 (4) 1444
for return code 8 (8) 1447
reason codes for ICSF *(continued)*
for return code C (12) 1489
REBIND 1065
record chaining 1645
Recover PIN from Offset (CSNBPFO)
format 806
parameters 806
syntax 806
reenciphered
key 237
reenciphering
data-encrypting key 130
PIN block 678, 685, 701
reference PAN data parameter
Encrypted PIN Verify2 callable service 727
reference PIN block length parameter
Encrypted PIN Verify2 callable service 727
reference PIN block parameter
Encrypted PIN Verify2 callable service 727
reference PIN encrypting key identifier length parameter
Encrypted PIN Verify2 callable service 724
reference PIN encrypting key identifier parameter
Encrypted PIN Verify2 callable service 724
reference PIN profile length parameter
Encrypted PIN Verify2 callable service 726
reference PIN profile parameter
Encrypted PIN Verify2 callable service 726
reference_PIN_rule_array parameter
Encrypted PIN Verify2 callable service 722
reference_PIN_rule_array_count parameter
Encrypted PIN Verify2 callable service 722
related publications xlv
remote key distribution
benefits 56
scenario 54
remote key export
exporting keys 50
generating keys 53
remote key export callable service (CSNDRKX)
overview 40
Remote Key Export callable service (CSNDRKX)
format 393
parameters 393
syntax 393
remote key loading
example 44
remote key-loading
CCA API changes 49
reserved data length parameter
symmetric MAC generate callable service 593
symmetric MAC verify callable service 598
reserved data parameter
Symmetric MAC generate callable service 593
symmetric MAC verify callable service 598
reserved parameter
control vector generate callable service 123, 344
retained key delete callable service (CSNDRKD)
overview 99
Retained Key Delete callable service (CSNDRKD)
format 1191
parameters 1191
syntax 1191
retained key list callable service (CSNDRKL)
overview 99

Retained Key List callable service (CSNDRKL)
format 1194
parameters 1194
syntax 1194
retained private keys
overview 99
return codes
from PCF macros
migration consideration 7
returned PVV parameter 638
returned result parameter
Clear PIN Generate callable service 632
Rivest-Shamir-Adleman (RSA) algorithm 93
RKX key token 49
RKX key-token 1508
RSA algorithm 93
RSA enciphered key length parameter
Symmetric Key Generate callable service 436
Symmetric Key Import callable service 441
RSA enciphered key parameter
Symmetric Key Generate callable service 436
Symmetric Key Import callable service 442
RSA private key identifier 442
RSA private key identifier length 442
RSA public key identifier length parameter
for Symmetric Key Generate 435
RSA public key identifier parameter 435
rule array count parameter
Clear PIN encrypt callable service 125, 673
Clear PIN Encrypt callable service 626
Clear PIN Generate Alternate callable service 637
Clear PIN Generate callable service 630
Control Vector Translate callable service 126
decipher callable service 487
Digital Signature Verify callable service 1120
Diversified Key Generate callable service 152
encipher callable service 496
encrypted PIN translate callable service 680
encrypted PIN translate enhanced callable service 703
Encrypted PIN Verify callable service 716
Encrypted PIN Verify2 callable service 721
Key Test callable service 253
Key Test Extended callable service 268
Key Token Build callable service 274
MAC Generate callable service 558
MDC Generate callable service 581
One-Way Hash Generate callable service 585
PKA key generate callable service 1171
PKA Key Generate callable service 1133
PKA Key Import callable service 1142
PKA key token build callable service 1150
PKA public key extract callable service 1182
Symmetric Key Export callable service 420
Symmetric Key Generate callable service 431
Symmetric Key Import callable service 440
Symmetric MAC generate callable service 591
Symmetric MAC verify callable service 596
trusted block create callable service 451
rule array parameter
Clear PIN Encrypt callable service 626
Clear PIN Generate Alternate callable service 637
Clear PIN Generate callable service 630
control vector generate callable service 120
control vector translate callable service 184
rule array parameter *(continued)*
Control Vector Translate callable service 126
decipher callable service 487
Digital Signature Verify callable service 1121
Diversified Key Generate callable service 152
encipher callable service 496
encrypted PIN generate callable service 674
encrypted PIN translate callable service 680
encrypted PIN translate enhanced callable service 703
encrypted PIN verify callable service 716
Encrypted PIN Verify2 callable service 721
Key Test callable service 253
Key Test Extended callable service 269
Key Token Build callable service 275
MAC Generate callable service 558
MDC Generate callable service 582
One-Way Hash Generate callable service 586
PKA key generate callable service 1171
PKA Key Generate callable service 1133
PKA key token build callable service 1150
PKA public key extract callable service 1182
Random Number Generate callable service 390
Symmetric Key Export callable service 420
Symmetric Key Generate callable service 431
Symmetric Key Import callable service 440
Symmetric MAC generate callable service 592
symmetric MAC verify callable service 596
trusted block create callable service 451
rule_array_count
ICSF query service callable service 1274, 1279, 1317
Random Number Generate callable service 390
rule_id parameter
Remote Key Export callable service 398
rule_id_length parameter
Remote Key Export callable service 398

S

SAF ACEE Selection (CSFACEE)
format 1321
parameters 1321
SAF ACEE Selection (CSFACEE2)
syntax 1321
sample JCL statements 12
SCHEDULE macro
FEATURE=CRYPTO keyword 9
SCSFSTUB module 12
section sequence, trusted block 1589
secure key import callable service (CSNBSKI)
overview 40
Secure Key Import callable service (CSNBSKI)
format 408
parameters 408
syntax 408
secure messaging
overview 71
secure messaging for keys callable service (CSNBSKY)
format 1166
parameters 834
syntax 1166
Secure Messaging for Keys callable service (CSNBSKY)
format 811
parameters 812
syntax 811

Secure Messaging for PINs callable service (CSNBSPN)
format 816
parameters 817
syntax 816
Secure Sockets Layer (SSL) 59
security considerations 9
segmenting
control keywords 558, 570, 582, 592, 596
definition 1644
rule, large data object 1644
sending to IBM
reader comments xlvii
sequence number parameter
encrypted PIN translate callable service 682
encrypted PIN translate enhanced callable service 710
sequences of callable service 72
set attribute value callable service (CSFPSAV)
format 1389
parameters 1389
syntax 1389
SET block compose callable service (CSNDSBC)
overview 100
SET Block Compose callable service (CSNDSBC)
format 823
parameters 824
syntax 823
SET block decompose callable service (CSNDSBD)
overview 100
SET Block Decompose callable service (CSNDSBD)
format 828
paramters 829
syntax 828
SET protocol 100
SET Secure Electronic Transaction 100
short blocks 493
shortcut keys 1727
signature algorithms 1109
signature field length parameter
Digital Signature Verify callable service 1124
signature field parameter
Digital Signature Verify callable service 1124
single-length key
purpose 74
source key identifier length parameter
PKA Key Import callable service 1142
PKA public key extract callable service 1183
source key identifier parameter
key export callable service 208
key import callable service 239
PKA Key Import callable service 1143
PKA public key extract callable service 1183
source key token length parameter
Prohibit Export Extended callable service 387
source text parameter
character/nibble conversion callable service 1268
code conversion callable service 1270
X9.9 data editing callable service 1324
source_key_length parameter
Remote Key Export callable service 398
special secure mode 10
SRB, scheduling 9
SSL support 59
sym_encrypted_key_length parameter
Remote Key Export callable service 399
symmetric algorithm decipher callable service (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)
format 501
parameters 501
syntax 501
symmetric algorithm encipher callable service (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)
format 509
syntax 509
symmetric algorithm encipher callable service CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)
parameters 509
symmetric key decipher callable service (CSNBSYD and CSNBSYD1)
format 521
parameters 521
syntax 521
symmetric key encipher callable service (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)
format 531
parameters 531
syntax 531
symmetric key export callable service (CSNDSYX and CSNFSYX)
overview 42
symmetric key export callable service (CSNDSYX)
overview 40
Symmetric Key Export callable service (CSNDSYX)
format 417
parameters 417
syntax 417
Symmetric Key Export with Data callable service(CSNDSXD)
format 426
syntax 426
symmetric key generate callable service (CSNDSYG and CSNFSYG)
overview 42
symmetric key generate callable service (CSNDSYG)
overview 40
Symmetric Key Generate callable service (CSNDSYG)
format 430
parameters 430
syntax 430
symmetric key import callable service (CSNDSYI and CSNFSYI)
overview 42
symmetric key import callable service (CSNDSYI)
overview 41
Symmetric Key Import callable service (CSNDSYI)
format 439
parameters 439
syntax 439
Symmetric MAC
generation callable service 62
verify callable service 62
Symmetric MAC generate callable service (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)
format 590
parameters 590
syntax 590
usage notes 594, 599
Symmetric MAC verify callable service (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)
format 595

Symmetric MAC verify callable service (CSNBSMV, CSNBSMV1, CSNBSMV2, and CSNBSMV3) 595
parameters 595
syntax 595
syntax for callable service 3

T

target key identifier length parameter 1143
target key identifier parameter
key export callable service 209
key import callable service 239
Symmetric Key Import callable service 442
target key token parameter
encrypted PIN generate callable service 126
target public key token length parameter 1183
target public key token parameter 1183
target text parameter
character/nibble conversion callable service 1268, 1275, 1277, 1315, 1320
code conversion callable service 1270
ICSF query facility callable service 1281, 1317
X9.9 data editing callable service 1324
text id in parameter
MAC Generate callable service 560
MAC verify callable service 572
MDC Generate callable service 583
One-Way Hash Generate callable service 588
symmetric MAC verify callable service 598
text length parameter
character/nibble conversion callable service 1268
code conversion callable service 1270
Cryptographic Variable Encipher callable service 129
decipher callable service 486
encipher callable service 495
MAC Generate callable service 558
MDC Generate callable service 581
One-Way Hash Generate callable service 587
Symmetric MAC generate callable service 591
Symmetric MAC verify callable service 596
X9.9 data editing callable service 1323
text parameter
MAC Generate callable service 558
MDC Generate callable service 581
One-Way Hash Generate callable service 588
Symmetric MAC generate callable service 591
symmetric MAC verify callable service 596
text, translating 471
TKE
overview 71
Token Record Create callable service (CSFPTRC)
format 1407
parameters 1407
syntax 1407
Token Record Delete callable service (CSFPTRD)
format 1411
parameters 1411
syntax 1411
Token Record List callable service (CSFPTRL)
format 1413
parameters 1413
syntax 1413
token validation value (TVV) 1502
TR-31
CCA API changes 938
TR-31 import callable service (CSNBT31I and CSNET31I) 959
TR-31 Optional Data Build callable services (CSNBT31O and CSNET31O) 998
TR-31 Optional Data Read callable services (CSNBT31R and CSNET31R) 1001
TR-31 Parse callable service (CSNBT31P and CSNET31P) 1004
TR-31 symmetric keys
management 937
TR-31 Translate callable service (CSNBT31X and CSNET31X) 1007
TR-34
protocol 1059
user flow setup 1060
user flows 1059
TR-34 Key Distribution (CSNDT34D and CSNFT34D) 1086
TR-34 Key Receive (CSNDT34R and CSNFT34R) 1098
TR-34 symmetric keys
management 1059
trademarks 1732
trailing short blocks 493
Transaction Validation callable service (CSNBSKY)
format 834
syntax 834
Transaction Validation callable service (CSNBTRV) 834
transport_key_identifier parameter
Remote Key Export callable service 397
trusted block create callable service 452
transport_key_length parameter
Remote Key Export callable service 397
trusted block 45
trusted block create
callable service 450
trusted block create callable service (CSNDTBC)
format 450
overview 41
parameters 450
syntax 450
trusted block sections
format of 1590
trusted blocks
CCA API changes 49
creating 50
exporting keys 50
using 50
Trusted Key Entry
overview 71
trusted_block_identifier parameter
trusted block create callable service 395, 453
trusted_block_length parameter
Remote Key Export callable service 395
trusted block create callable service 453
types of keys 21, 23, 26, 27

U

UKPT
format 613
UNBIND 1064
user interface
ISPF 1727

user interface *(continued)*
TSO/E 1727
utilities
character/nibble conversion 1267
code conversion 1269
ICSF Query Algorithm 1273
ICSF Query Facility 1278
ICSF Query Facility2 1316
Key Token Build 271
PKA key token build 1147
SAF ACEE Selection 1321
X9.9 data editing 1322

V

V2R3 changed information FMID HCR77C1 lxv
V2R3 changed information FMID HCR77D0 lxii
V2R3 deleted information FMID HCR77C1 lxviii
V2R3 deleted information FMID HCR77D0 lxiii
V2R3 new information FMID HCR77C1 lxiv
V2R3 new information FMID HCR77D0 lx
V2R4 changed information FMID HCR77D1 lvi
V2R4 deleted information FMID HCR77D1 lx
V2R4 new information FMID HCR77D1 lv
V2R5 changed information FMID HCR77D2 l
V2R5 deleted information FMID HCR77D2 lv
V2R5 new information FMID HCR77D2 xlix
verification pattern parameter 255, 270
verification pattern, generating and verifying 252, 267
verifying data integrity and authenticity 543
VISA CVV Service Generate callable service (CSNBCSG)
format 839
parameters 839
syntax 839
VISA CVV Service Verify callable service (CSNBCSV)
format 843
parameters 843
syntax 843
VISA PVV
generating 634
VISA-1 683
VISA-2 PIN block format 610, 1632
VISA-3 PIN block format 610, 1633
VISA-4 PIN block format 610
VISA-PVV algorithm 638, 716
VISAPVV4 algorithm 716

W

Weak PIN table 849
where to find information xlv

X

X.509 certificates 101
X9.143 (TR-31)
key block header 1538
key blocks 28
optional block data 1538
X9.143 (TR-31) symmetric keys
management 937
X9.9 data editing callable service (CSNB9ED)
format 1322
X9.9 data editing callable service (CSNB9ED) *(continued)*
overview 72
parameters 1322
syntax 1322
X9.9-1 keyword 558, 570

SC14-7508-10