z/OS

Cryptographic Services Integrated Cryptographic Service Facility Application Programmer's Guide

Version 2 Release 1

SC14-7508-03

Note

Before using this information and the product it supports, read the information in "Notices" on page 1143.

This edition applies to ICSF FMID HCR77B0 and Version 2 Release 1 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 14. Trusted Key Entry Workstation Interfaces 851

Part 3. PKCS #11 Callable Services 859

Chapter 15. Using PKCS #11 tokens and objects 861

Figures

Tables

About this information

This information describes how to use the callable services provided by the Integrated Cryptographic Service Facility (ICSF). The z/OS Cryptographic Services includes 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 featires 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 includes Advanced Encryption Standard (AES), Data Encryption Standard (DES) and public key cryptography. These are very different cryptographic systems.

These topics focus on IBM CCA programming and include:

- Chapter 1, "Introducing Programming for the IBM CCA," on page 3 describes the programming considerations for using the ICSF callable services. It also explains the syntax and parameter definitions used in callable services.
- Chapter 2, "Introducing symmetric key cryptography and using symmetric key callable services," on page 15 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 called installation-defined callable services and provides suggestions on what to do if there is a problem.
- Chapter 3, "Introducing PKA Cryptography and Using PKA Callable Services," on page 81 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.
- Chapter 4, "Introducing PKCS #11 and using PKCS #11 callable services," on page 95 gives an overview of PKCS #11 support and management services.

These topics focus on CCA callable services and include:

- Chapter 5, "Managing Symmetric Cryptographic Keys," on page 101 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.

- Chapter 6, "Protecting Data," on page 371 describes the callable services for deciphering ciphertext from one key and enciphering it under another key. It also describes enciphering and deciphering data with encrypted keys and encoding and decoding data with clear keys.
- Chapter 7, "Verifying Data Integrity and Authenticating Messages," on page 439 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).
- Chapter 8, "Financial Services," on page 491 describes the callable services for generating, verifying, and translating personal identification numbers (PINs). It also describes the callable services that support the Secure Electronic Transaction (SET) protocol and those that and generate and verify VISA card verification values and American Express card security codes.
- Chapter 9, "Financial Services for DK PIN Methods," on page 635 describes the 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.
- Chapter 10, "Using Digital Signatures," on page 707 describes the PKA callable services that support using digital signatures to authenticate messages.
- Chapter 11, "Managing PKA Cryptographic Keys," on page 719 describes the PKA callable services that generate and manage PKA keys.
- Chapter 12, "Key data set management," on page 757 describes the callable services that manage key tokens in the Cryptographic Key Data Set (CKDS) and the PKA Key Data Set (PKDS).
- Chapter 13, "Utilities," on page 811 describes callable services that convert data between EBCDIC and ASCII format, convert between binary strings and character strings, and query ICSF services and algorithms.
- Chapter 14, "Trusted Key Entry Workstation Interfaces," on page 851 describes the service that supports Trusted Key Entry (TKE) workstation, an optional feature available with ICSF.
- Chapter 15, "Using PKCS #11 tokens and objects," on page 861 describes the callable services for managing the PKCS #11 tokens and objects in the TKDS.

The appendixes include this information:

- Appendix A, "ICSF and cryptographic coprocessor return and reason codes," on page 943 explains the return and reason codes returned by the callable services.
- Appendix B, "Key Token Formats," on page 995 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.
- Appendix C, "Control Vectors and Changing Control Vectors with the CVT Callable Service," on page 1065 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.
- Appendix D, "Coding Examples," on page 1081 provides examples for COBOL, assembler, C, and PL/1.

- Appendix E, "Cryptographic Algorithms and Processes," on page 1093 describes the PIN formats and algorithms, cipher processing and segmenting rules, multiple encipherment and decipherment and their equations, and the PKA92 encryption process.
- Appendix F, "EBCDIC and ASCII Default Conversion Tables," on page 1119 presents EBCDIC to ASCII and ASCII to EBCDIC conversion tables.
- Appendix G, "Access control points and callable services," on page 1121 lists which access control points correspond to which callable services.
- Appendix H, "Accessibility," on page 1139 contains information on accessibility features in z/OS.
- "Notices" on page 1143 contains notices, programming interface information, and trademarks.

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*

  This publication can be obtained in PDF format from the Library page at http://www.ibm.com/security/cryptocards.

IBM Crypto Education

The IBM Crypto Education community provides detailed explanations and samples pertaining to IBM cryptographic technology at https://www-304.ibm.com/connections/communities/community/crypto.

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  z/OS ICSF Application Programmer's Guide
  SC14-7508-03
- The topic and page number that is related to your comment.
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right 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

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS support page (http://www.ibm.com/systems/z/support/).

Summary of Changes

ICSF is an element of z/OS, but provides independent ICSF releases as web deliverables. These web deliverables are identified by their FMID. Each release of z/OS includes a particular ICSF FMID level as part of its base.

This document contains terminology, maintenance, and editorial changes to improve consistency and retrievability. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Changes made in Cryptographic Support for z/OS V1R13 - z/OS V2R1 (FMID HCR77B0)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SC14-7508-01.

This document is for ICSF FMID HCR77B0. This release of ICSF runs on z/OS V1R13 and z/OS V2R1 and only on zSeries hardware.

New

- New information has been added to "Callable services for managing the CKDS" on page 51.
- New information has been added to "Callable services to manage the Public Key Data Set (PKDS)" on page 86.
- The following callable services have been added:
  - "Field level decipher (CSNBFLD and CSNEFLD)" on page 546.
  - "Field level encipher (CSNBFLE and CSNEFLE)" on page 553.
  - "FPE decipher (CSNBFPED and CSNEFPED)" on page 563.
  - "FPE encipher (CSNBFPEE and CSNEFPEE)" on page 571.
  - "FPE translate (CSNBFPET and CSNEFPET)" on page 580.
  - "ICSF Multi-Purpose Service (CSFMPS and CSFMPS6)" on page 775.
  - "Key Data Set List (CSFKDSL and CSFKDSL6)" on page 778.
  - "Key Data Set Metadata Read (CSFKDMR and CSFKDMR6)" on page 788.
  - "Key Data Set Metadata Write (CSFKDMW and CSFKDMW6)" on page 795.
- "Metadata for key data set records" on page 757 is new for key data set management.
- The following new reason codes have been added:
  - "Reason codes for return code 0 (0)" on page 944:
    - D25 (3365)
  - "Reason codes for return code 4 (4)" on page 945:
    - CE4 (3300)
    - CE5 (3301)
    - CE6 (3302)
    - CE7 (3303)
    - CEB (3307)
    - CF4 (3316)

- D10 (3344)
- D11 (3345)
- D20 (3360)
- F9E (3998)
– "Reason codes for return code C (12)" on page 982:
  - 839 (2105)
  - D21 (3361)
  - D22 (3362)
  - D23 (3363)
  - D24 (3364)
  - 8CF4 (36084)
  - 8D48 (36168)
- "SHA-256 algorithm" on page 1118 is new.

Changed

- "Trusted Key Entry (TKE) Support" on page 63 has been updated.
- "PKA Master Keys" on page 81 has been updated.
- "Operational private keys" on page 82 has been updated.
- "PKA Callable Services" on page 84 has been updated.
- "Callable Services Supporting Digital Signatures" on page 84 has been updated.
- "ANSI X9.8 PIN Restrictions" on page 495 has been updated.
- "PKA Key Token Change (CSNDKTC and CSNFKTC)" on page 740 has been updated.
- The usage notes have been updated in "ICSF Query Algorithm (CSFIQA and CSFIQA6)" on page 815.
- The *returned_data* parameter has been updated in "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 819.
- "PCI Interface Callable Service (CSFPCI and CSFPCI6)" on page 851 has been updated.
- Chapter 14, "Trusted Key Entry Workstation Interfaces," on page 851 has been updated.
- The following reason codes have been updated:
  – "Reason codes for return code 8 (8)" on page 948
    - 2CB (715)
    - 834 (2100)
    - 3E94 (16020)
  – "Reason codes for return code C (12)" on page 982
    - C0A (3082)
    - 2B34 (11060)
    - 8CA2 (36002)
- Appendix G, "Access control points and callable services," on page 1121 has been updated.

Deleted

No content was removed from this information.

Changes made in Cryptographic Support for z/OS V1R13 - z/OS V2R1 (FMID HCR77A1) as updated June 2014

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SC14-7508-00.

This document is for ICSF FMID HCR77A1. This release of ICSF runs on z/OS V1R13 and z/OS V2R1 and only on zSeries hardware.

New

- Support for the German Banking Industry Committee (Deutsche Kreditwirtschaft (DK)) PIN methods with the application of the PTFs for APARs OA42246, OA43906, and OA44444:
  - "AES key types" on page 25 is new.
  - "DK PIN methods support" on page 29 is new.
  - "Diversified Key Generate2 Callable Service (CSNBDKG2 and CSNEDKG2)" on page 34 is new.
  - The following services are updated:
    - "Key Generate2 (CSNBKGN2 and CSNEKGN2)" on page 156
    - "Key Part Import2 (CSNBKPI2 and CSNEKPI2)" on page 176
    - "Key Test2 (CSNBKYT2 and CSNEKYT2)" on page 185
    - "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 203
    - "Key Translate2 (CSNBKTR2 and CSNEKTR2)" on page 235
    - "Restrict Key Attribute (CSNBRKA and CSNERKA)" on page 279
    - "Secure Key Import2 (CSNBSKI2 and CSNESKI2)" on page 287
    - "Symmetric Key Export (CSNDSYX and CSNFSYX)" on page 292
    - "Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)" on page 314
    - "Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)" on page 417
    - "Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)" on page 428
  - "MAC Generate2 (CSNBMGN2, CSNBMGN3, CSNEMGN2, and CSNEMGN3)" on page 456 and "MAC Verify2 (CSNBMVR2, CSNBMVR3, CSNEMVR2, and CSNEMVR3)" on page 466 in Chapter 7, "Verifying Data Integrity and Authenticating Messages," on page 439 are new.
  - Chapter 9, "Financial Services for DK PIN Methods," on page 635 is new:
    - "DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG)" on page 636
    - "DK Migrate PIN (CSNBDMP and CSNEDMP)" on page 643
    - "DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)" on page 650
    - "DK PAN Translate (CSNBDPT and CSNEDPT)" on page 657
    - "DK PIN Change (CSNBDPC and CSNEDPC)" on page 664
    - "DK PIN Verify (CSNBDPV and CSNEDPV)" on page 676
    - "DK PRW Card Number Update (CSNBDPNU and CSNEDPNU)" on page 680
    - "DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG)" on page 687
    - "DK Random PIN Generate (CSNBDRPG and CSNEDRPG)" on page 692

- "DK Regenerate PRW (CSNBDRP and CSNEDRP)" on page 698
  - "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 819 is updated.
  - "Variable-length symmetric key token" on page 1001 is updated.
  - Appendix G, "Access control points and callable services," on page 1121 is updated.
- Support for the integration of an existing User Defined Extension (UDX) into the CCA base with the application of the PTF for APAR OA43816:
  - "PKA Key Translate (CSNDPKT and CSNFPKT)" on page 743 is updated.
  - Appendix G, "Access control points and callable services," on page 1121 is updated.

Changed

- "SAF ACEE Selection (CSFACEE and CSFACEE6)" on page 846 is updated.
- Reason codes are updated in "Reason codes for return code 8 (8)" on page 948.

Deleted

No content was removed from this information.

Changes made in Cryptographic Support for z/OS V1R13-V2R1 (FMID HCR77A1)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SA22-7522-16.

This document is for ICSF FMID HCR77A1. This release of ICSF runs on z/OS V1R13, and z/OS V2R1 and only on zSeries hardware.

New information

- A new callable service, SAF ACEE Selection (CSFACEE and CSFACEE6), has been added allowing the caller to provide the ENVR to use for SAF checks.
- A new callable service, Symmetric Key Export with Data (CSNDSXD and CSNFSXD), has been added to export a symmetric key, along with some application supplied data, encrypted using an RSA key.
- A new callable service, Athentication Parameter Generate (CSNBAPG and CSNEAPG), has been added to generate an authentication parameter (AP) and return it encrypted using the key supplied in the with the AP_encrypting_key_identifier parameter.
- A new callable service, Recover PIN from Offset (CSNBPFO and CSNEPFO), has been added to calculate the encrypted customer-entered PIN from a PIN generating key, account information, and an IBM-PIN0 Offset.
- A new callable service, ICSF Query Facility2 (CSFIQF2 and CSFIQF26), has been added to retrieve status information on the cryptographic environment as currently known to ICSF.
- The "ISO Format 3" on page 1094 was added to "PIN Formats and Algorithms" on page 1093 in Appendix E, "Cryptographic Algorithms and Processes," on page 1093.
- A new "Payload format" on page 20 section has been added to "Functions of symmetric cryptographic keys" on page 15 in Chapter 2, "Introducing symmetric key cryptography and using symmetric key callable services," on page 15

Changed information

- The PKCS #11 Unwrap key (CSFPUWK and CSFPUWK6)and PKCS #11 Wrap key (CSFPWPK and CSFPWPK6)services have been updated to indicate that PCKS 8 formatting allows wrapping/unwrapping of symmetric keys with symmetric keys.
- The Coordinated KDS Administration (CSFCRC and CSFCRC6) callable service has been updated to perform a coordinated KDS conversion.
- The Remote Key Export (CSNDRKX and CSNFRKX) callable service has been updated with information regarding wrapping mode of output keys.
- The Unique Key Derive (CSNBUKD and CSNEUKD) callable service has been updated with new information regarding the Initial Pin Encryption Key (IPEK).
- The Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL) cxallable service has been updated with a new usage note regarding the CSF.CSFSERV.AUTH.CSFRNG.DISABLE SAF resource profile.
- The PKCS #11 Pseudo-random function (CSFPPRF and CSFPPRF6) callable service has been updated with a new usage note regarding the CSF.CSFSERV.AUTH.CSFRNG.DISABLE SAF resource profile.
- The One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1) callable service has been updated with a new usage note regarding the CSF.CSFSERV.AUTH.CSFRNG.DISABLE SAF resource profile.
- The PKCS #11 One-way hash, sign, or verify (CSFPOWH and CSFPOWH6) callable service has been updated with a new usage note regarding the CSF.CSFSERV.AUTH.CSFRNG.DISABLE SAF resource profile.
- The ICSF Query Facility (CSFIQF and CSFIQF6), Output for option ICSFSTAT table has been updated with a new IVSF status field.
- The PKCS #11 One-way hash, sign, or verify (CSFPOWH and CSFPOWH6) callable service has been updated for RSA-PKCS PSS hashing method.
- The PKCS #11 Derive key (CSFPDVK and CSFPDVK6) callable service rule-array parameter and usage notes have been updated.
- The PCI Interface Callable Service (CSFPCI and CSFPCI6) callable service rule-array parameter and usage notes have been updated.
- The Symmetric MAC Generate (CSNBSMG or CSNBSMG1 and CSNESMG or CSNESMG1) callable service has been updated for the KEY-DRV keyword.
- The Symmetric MAC Verify (CSNBSMV or CSNBSMV1 and CSNESMV or CSNESMV1) callable service has been updated for the KEY-DRV keyword.
-

Deleted information

- In z/OS V2R1, support for z900/z800 is removed. HCR77A0 ships in the base of z/OS V2R1 and will continue to support these older hardware environments. HCR77A1 removes support for the hardware present on z900/z800 (CCF and PCICC). z990/z890 (PCIXCC and PCICA) will continue to be supported because HCR77A1 will still run on z/OS V1R13.

  Additionally, a number of software functions that rely on the presence of CCF and PCICC features or do not serve a purpose without these features is removed, including the Managing Keys According to the ANSI X9.17 Standard chapter and the Using ICSF with BSAFE appendix.
- The Operational Key Load, and the Operational Key Load - Variable-Length Tokens, access control points for callable services CSNBOKL / CSNEOKL have been moved to the CCA coprocessor Access Control Points and Callable Services appendix of the ICSF Administrator’s Guide.

- Support for the following services has been removed in HCR77A1:
  - ANSI X9.17 EDC Generate (CSNAEGN and CSNGEGN)
  - ANSI X9.17 Key Export (CSNAKEX and CSNGKEX)
  - ANSI X9.17 Key Import (CSNAKIM and CSNGKIM)
  - ANSI X9.17 Key Translate (CSNAKTR and CSNGKTR)
  - ANSI X9.17 Transport Key Partial Notarize (CSNATKN and CSNGTKN)
  - Ciphertext Translate (CSNBCTT or CSNBCTT1 and CSNECTT or CSNECTT1)
  - Transform CDMF Key (CSNBTCK and CSNETCK)
  - User Derived Key (CSFUDK and CSFUDK6)
  - PKSC Interface Callable Service (CSFPKSC)

Changes made in Cryptographic Support for z/OS V1R12-R13 (FMID HCR77A0)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SA22-7522-15.

This document is for ICSF FMID HCR77A0. This release of ICSF runs on z/OS V1R11, z/OS V1R12, and z/OS V1R13 and only on zSeries hardware.

New information

- Added information for the Crypto Express4 feature, which can be configured as a CCA coprocessor (CEX4C), a Enterprise PKCS #11 coprocessor (CEX4P), or an accelerator (CEX4A).
- A new callable service, Unique Key Derive (CSFBUKD and CSFEUKD) has been added to perform the key derivation process using a base derivation key and derivation data as inputs for increased support for the Derived Unique Key Per Transaction (DUKPT) key-management scheme as described in ANSI X9.24 Part 1.
- A new callable service, Cipher Text Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3), has been added to translate cipher text. This new service supports both AES and DES algorithms and different encryption modes.
- Added 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 ensure that a key is not wrapped with a key weaker than itself.

Changed information

- Clear Key Import (CSNBCKI and CSNECKI)
- Control Vector Generate (CSNBCVG and CSNECVG)
- Coordinated KDS Administration Callable Service (CSFCRC and CSFCRC6)
- Data Key Export (CSNBDKX and CSNEDKX)
- Data Key Import (CSNBDKM and CSNEDKM)
- Diversified Key Generate (CSNBDKG and CSNEDKG)
- ECC Diffie-Hellman (CSNDEDH and CSNFEDH)
- Key Export (CSNBKEX and CSNEKEX)
- Key Generate (CSNBKGN and CSNEKGN)
- Key Generate2 (CSNBKGN2 and CSNEKGN2)
- Key Import (CSNBKIM and CSNEKIM)
- Key Token Build (CSNBKTB and CSNEKTB)

- Key Token Build2 (CSNBKTB2 and CSNEKTB2)
- Key Translate (CSNBKTR and CSNEKTR)
- Key Translate2 (CSNBKTR2 and CSNEKTR2)
- Multiple Clear Key Import (CSNBCKM and CSNECKM) v Multiple Secure Key Import (CSNBSKM and CSNESKM)
- PIN Change/Unblock (CSNBPCU and CSNEPCU)
- PKA Decrypt (CSNDPKD and CSNFPKD)
- Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)
- Remote Key Export (CSNDRKX and CSNFRKX)
- Restrict Key Attribute (CSNBRKA and CSNERKA)
- Secure Key Import (CSNBSKI and CSNESKI)
- Secure Key Import2 (CSNBSKI2 and CSNESKI2)
- Symmetric Key Export (CSNDSYX and CSNFSYX)
- Symmetric Key Generate (CSNDSYG and CSNFSYG)
- Symmetric Key Import (CSNDSYI and CSNFSYI)
- Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)
- Transaction Validation (CSNBTRV and CSNETRV)
- TR-31 Export (CSNBT31X and CSNET31X)
- TR-31 Import (CSNBT31I and CSNET31I)
- Digital Signature Generate (CSNDDSG and CSNFDSG)
- Digital Signature Verify (CSNDDSV and CSNFDSV)
- PKA Key Generate (CSNDPKG and CSNFPKG)
- PKA Key Import (CSNDPKI and CSNFPKI)
- PKA Key Token Build (CSNDPKB and CSNFPKB)
- PKA Key Token Change (CSNDKTC and CSNFKTC)
- PKA Key Translate (CSNDPKT and CSNFPKT)
- Coordinated KDS Administration (CSFCRC and CSFCRC6)
- ICSF Query Facility (CSFIQF and CSFIQF6)
- PCI Interface Callable Service (CSFPCI and CSFPCI6)
- PKCS #11 Generate secret key (CSFPGSK and CSFPGSK6)
- PKCS #11 Verify HMAC (CSFPHMV and CSFPHMV6)
- PKCS #11 Private key sign (CSFPPKS and CSFPPKS6)
- PKCS #11 Public key verify (CSFPPKV and CSFPPKV6)
- PKCS #11 Pseudo-random function (CSFPPRF and CSFPPRF6)
- PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6)
- PKCS #11 Secret key decrypt (CSFPSKD and CSFPSKD6)
- PKCS #11 Token record create (CSFPTRC and CSFPTRC6)
- PKCS #11 Unwrap key (CSFPUWK and CSFPUWK6)
- PKCS #11 Wrap key (CSFPWPK and CSFPWPK6)

Changes made in Cryptographic Support for z/OS V1R11-R13 (FMID HCR7790)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SA22-7522-14.

This document is for ICSF FMID HCR7790. This release of ICSF runs on z/OS V1R11, z/OS V1R12, and z/OS V1R13 and only on zSeries hardware.

New information

- Added support for the TR-31 key block format defined by the American National Standards Institute (ANSI). 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. The following callable services have been added to provide this support:
  - TR-31 Export (CSNBT31X and CSNET31X)
  - TR-31 Import (CSNBT31I and CSNET31I) TR-31
  - TR-31 Parse (CSNBT31P and CSNET31P) TR-31
  - TR-31 Optional Data Read (CSNBT31R and CSNET31R)
  - TR-31 Optional Data Build (CSNBT31O and CSNET31O)
- Added new callable service, CVV key combine (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 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. See "CVV Key Combine (CSNBCKC and CSNECKC)" for more information.
- Added support for coordinated and dynamic update of a CKDS. The new callable service Coordinated KDS Administration (CSFCRC and CSFCRC6) which performs a CKDS refresh or reencipher operation while allowing applications to update the CKDS. In a sysplex environment, this callable service enables an application to perform a coordinated sysplex-wide refresh or reencipher operation from a single ICSF instance.
- Added new callable service ECC Diffie-Hellman (CSNDEDH and CSNFEDH), which applications can use 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.
- A new health check, ICSFMIG_DEPRECATED_SERV_WARNINGS, has been added to the Health Checker to detect the use of services that will not be supported in subsequent releases: The deprecated services checked in this release are listed below. These are not supported on post zSeries 900 hardware, and will not be supported in subsequent releases of ICSF.
  - ANSI X9.17 EDC Generate
  - ANSI X9.17 Key Export
  - ANSI X9.17 Key Import
  - ANSI X9.17 Key Translate
  - ANSI X9.17 Transport Key Partial Notarize
  - Ciphertext Translate
  - Ciphertext Translate with ALET
  - Transform CDMF Key
  - User Derived Key
  - PKSC Interface Callable Service

  You should use the ICSFMIG_DEPRECATED_SERV_WARNINGS check to determine if these services are being used. For more information on this health check, refer to z/OS Cryptographic Services ICSF Administrator's Guide.

Changed information

- CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)
- Clear PIN Generate (CSNBPGN and CSNEPGN)
- Clear PIN Generate Alternate (CSNBCPA and CSNECPA)
- Control Vector Generate (CSNBCVG and CSNECVG)
- Digital Signature Verify (CSNDDSV and CSNFDSV)
- Encrypted PIN Generate (CSNBEPG and CSNEEPG)
- Encrypted PIN Verify (CSNBPVR and CSNEPVR)
- ICSF Query Facility (CSFIQF and CSFIQF6)
- Key Generate2 (CSNBKGN2 and CSNEKGN2)
- Key Part Import2 (CSNBKPI2 and CSNEKPI2)
- Key Test2 (CSNBKYT2 and CSNEKYT2)
- Key Token Build (CSNBKTB and CSNEKTB)
- Key Token Build2 (CSNBKTB2 and CSNEKTB2)
- Key Translate2 (CSNBKTR2 and CSNEKTR2)
- PKDS Key Record Create (CSNDKRC and CSNFKRC)
- PKDS Key Record Delete (CSNDKRD and CSNFKRD)
- PKDS Key Record Read (CSNDKRR and CSNFKRR)
- PKDS Key Record Write (CSNDKRW and CSNFKRW)
- PKA Decrypt (CSNDPKD and CSNFPKD)
- PKA Encrypt (CSNDPKE and CSNFPKE)
- PKA Key Generate (CSNDPKG and CSNFPKG)
- PKA Key Import (CSNDPKI and CSNFPKI)
- PKA Key Token Change (CSNDKTC and CSNFKTC)
- Restrict Key Attribute (CSNBRKA and CSNERKA)
- Secure Key Import2 (CSNBSKI2 and CSNESKI2)
- Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)
- Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)
- Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)
- Symmetric Key Generate (CSNDSYG and CSNFSYG)
- Symmetric Key Import (CSNDSYI and CSNFSYI)
- Symmetric Key Export (CSNDSYX and CSNFSYX)
- VISA CVV Service Generate (CSNBCSG and CSNECSG)
- VISA CVV Service Verify (CSNBCSV and CSNECSV)

For clarity:

- CSNBKRC and CSNEKRC, which had been referred to as the "Key Record Create" service, are now referred to as the "CKDS Key Record Create" service
- CSNBKRC2 and CSNEKRC2, which had been referred to as the "Key Record Create2" service, are now referred to as the "CKDS Key Record Create2" service
- CSNBKRD and CSNEKRD, which had been referred to as the "Key Record Delete" service, are now referred to as the "CKDS Key Record Delete" service
- CSNBKRR and CSNEKRR, which had been referred to as the "Key Record Read" service, are now referred to as the "CKDS Key Record Read" service

- CSNBKRR2 and CSNEKRR2, which had been referred to as the "Key Record Read2" service, are now referred to as the "CKDS Key Record Read2" service
- CSNBKRW and CSNEKRW, which had been referred to as the "Key Record Write" service, are now referred to as the "CKDS Key Record Write" service
- CSNBKRW2 and CSNEKRW2, which had been referred to as the "Key Record Write2" service, are now referred to as the "CKDS Key Record Write2" service
- CSNDKRC and CSNFKRC, which had been referred to as the "PKDS Record Create" service, are now referred to as the "PKDS Key Record Create" service
- CSNDKRD and CSNFKRD, which had been referred to as the "PKDS Record Delete" service, are now referred to as the "PKDS Key Record Delete" service
- CSNDKRR and CSNFKRR, which had been referred to as the "PKDS Record Read" service, are now referred to as the "PKDS Key Record Read" service
- CSNDKRW and CSNFKRW, which had been referred to as the "PKDS Record Write" service, are now referred to as the "PKDS Key Record Write" service

References to the IBM Eserver zSeries 800 (z800) do not appear in this information. Be aware that the documented notes and restrictions for the IBM Eserver zSeries 900 (z900) also apply to the z800.

Part 1. IBM CCA Programming

IBM CCA Programming introduces programming for the IBM CCA, including AES, DES, RSA, and ECC cryptography. It explains how to use these callable services.

Chapter 1. Introducing Programming for the IBM CCA

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 7 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 1081.

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*.

Verb	Callable Service without ALET	Callable Service with ALET
Ciphertext Translate2	CSNBCTT2	CSNBCTT3
Decipher	CSNBDEC	CSNBDEC1
Encipher	CSNBENC	CSNBENC1
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 passed to the callable services, such as CLEAR, CBC, and FIRST can be in lower, upper, 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 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 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 and reason codes," on page 943.

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 and reason codes," on page 943.

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 these:

- **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 may contain a key encrypted under an importer key-encrypting key but does not contain the other information present in an external key token.

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

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 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:

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 upper case (A-Z). All labels in the key data sets are created with upper case characters.
- Labels must be unique for all key types except EXPORTER, IMPORTER, PINGEN, PINVER, OPINENC and IPINENC.
- Transport and PIN keys can have duplicate labels for different key types. Keys that use the dynamic CKDS update services to create or update, however, must have unique key labels.
- Labels must be unique for any key record, including transport and PIN keys, created or updated 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 16Mb but must be 31-bit addressable.

  Data can be located above 2Gb 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

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.

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 Use 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.

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.

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 that 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 and reason codes," on page 943, 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 and reason codes,” on page 943.

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/1, see Appendix D, “Coding Examples,” on page 1081.

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, Assembler H, or PL/1)

In the SYSLIB concatenation, include the CSF.SCSFMOD0 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'
```

```
//SYSUT1   DD   UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSN=CSF.SCSFMOD0,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 symmetric key cryptography and using symmetric key callable services

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
- Encoding and decoding data with clear keys
- 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

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 25 for more information about the AES key types added for the DK PIN methods.

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.

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.

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.

Note: In PCF, key separation applies only to keys enciphered under the master key (keys in operational form). In ICSF, key separation also applies to keys enciphered under transport keys (keys in importable or exportable form). This allows the creator of a key to transmit the key to another system and to enforce its use at the other system.

Master key variant for fixed-length tokens

Whenever the master key is used to encipher a key, the cryptographic coprocessor produces a variation of the master key according to the type of key the master key will encipher. These variations are called *master key variants*. The cryptographic coprocessor creates a master key variant by exclusive ORing a fixed pattern, called a *control vector*, onto the master key. A unique control vector is associated with each type of key. For example, all the different types of data-encrypting, PIN, MAC, and transport keys each use a unique control vector which is exclusive ORed with the master key in order to produce the variant. The different key types are described in “Types of keys” on page 20.

Each master key variant protects a different type of key. It is similar to having a unique master key protect all the keys of a certain type.

The master key, in the form of master key variants, protects keys operating on the system. A key can be used in a cryptographic function only when it is enciphered under a master key. When systems want to share keys, transport keys are used to protect keys sent outside of systems. When a key is enciphered under a transport key, the key cannot be used in a cryptographic function. It must first be brought on to a system and enciphered under the system's master key, or exported to another system where it will then be enciphered under that system's master key.

Transport key variant for fixed-length tokens

Like the master key, ICSF creates variations of a transport key to encrypt a key according to its type. This allows for key separation when a key is transported off the system. A *transport key variant*, also called *key-encrypting key variant*, is created the same way a master key variant is created. The transport key's clear value is exclusive ORed with a control vector associated with the key type of the key it protects.

Note: To exchange keys with systems that do not recognize transport key variants, ICSF allows you to encrypt selected keys under a transport key itself, not under the transport key variant. For more information, see the "Transport keys (or key-encrypting keys)" list item in "Types of keys" on page 20.

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. The Secure Key Import and 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 65 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 two formats of 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 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

If the first byte of the key identifier is X'01', the key identifier is interpreted as an **internal key token**. 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 101.

- 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
- Secure Key Import, 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*.

Attention: 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 995 for the format of an internal key token.

If the first byte of the key identifier is X'02', the key identifier is interpreted as an **external key token**. 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 these listed. They are described in detail in Chapter 5, "Managing Symmetric Cryptographic Keys," on page 101.
- 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 995 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- 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 995 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 Export callable service is used to export CCA operational tokens to TR-31 key blocks. The TR-31 Import callable service is used to import TR-31 key blocks to CCA operational tokens.

Key wrapping

ICSF supports two methods of wrapping the key value in a fixed-length token: ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant.

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

The key value in DES tokens may be wrapped in two ways:
- The original (ECB) 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.
- The enhanced (CBC) method which was added later. For this method, the key value is bundled with other token data and encrypted using triple DES encryption and cipher block chaining mode. This enhanced method requires a IBM zEnterprise 196, zEnterprise 114, or later with a CCA cryptographic coprocessor that is a CEX3C or later.

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

WRAP-ENH (use enhanced method) and WRAP-ECB (use original ECB key-wrapping method) rule array keywords.

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 HCR77A1, these tokens used a variable-length payload which consisted of the encrypted key and padding. 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.

DATA keys can be either encrypted under the master key or in the clear (See "Clear keys" on page 26 for details on using clear keys). CIPHER keys are encrypted under the master key.

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 59 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 or double length keys with standard CVs for key types DATA, DATAC, DATAM, DATAMV, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER.

- NOCV key-encrypting keys can be used with triple length DATA keys. Since DATA keys have 0 CVs, processing will be the same as if the key-encrypting keys are standard key-encrypting keys (not the NOCV key-encrypting key).

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. Only DATA keys can be triple-length.

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 3. 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.	
DATA	Authentication Parameter Generate, Cipher Text Translate2, CVV Key Combine, Decipher, 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

Table 3. Descriptions of DES key types and service usage (continued)

DES key type	Usable with services
Cipher class (data operation keys) These key are used to encrypt and decrypt data. The keys can be single-length or double-length.	
CIPHER	Cipher Text Translate2, Decipher, Encipher
DECIPHER	Cipher Text Translate2, Decipher
ENCIPHER	Cipher Text Translate2, Encipher
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 or double-length keys.	
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 are double-length keys.	
PINGEN	Clear PIN Generate, Clear PIN Generate Alternate, Encrypted PIN Generate, Recover PIN from Offset
PINVER	Encrypted PIN Verify
These keys are used wrap and unwrap PIN blocks:	
IPINENC	Authentication Parameter Generate, Clear PIN Generate Alternate, Encrypted PIN Translate, Encrypted PIN Verify, PIN Change/Unblock, Secure Messaging for PINs
OPINENC	Clear PIN Encrypt, Clear PIN Generate Alternate, Encrypted PIN Generate, Encrypted PIN Translate, PIN Change/Unblock, Recover PIN from Offset
Key-encrypting key class These keys are used to wrap other keys. The keys are double-length keys.	
EXPORTER	Control Vector Translate, Data Key Export, ECC Diffie-Hellman, 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 Export, TR-31 Import, Unique Key Derive
IMPORTER	Control Vector Translate, Data Key Import, ECC Diffie-Hellman, 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 Export, TR-31 Import

Table 3. Descriptions of DES key types and service usage (continued)

DES key type	Usable with services
IMP-PKA	PKA Key Import, Remote Key Export, Trusted Block Create
IKEYXLAT, OKEYXLAT	Control Vector Translate, Key Translate, Key Translate2, TR-31 Export,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 Verify, Unique Key Derive
DKYGENKY	Diversified Key Generate, PIN Change/Unblock
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 4. Descriptions of AES key types and service usage

AES key type	Usable with services
Fixed-length AES key-token, version X'04'	
DATA	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	Symmetric Algorithm Decipher, Symmetric Algorithm Encipher, Ciphertext Translate2
Key-encrypting key class These keys are used to cipher other keys.	
EXPORTER	Key Generate2, Key Translate2, PKA Key Generate, Symmetric Key Export
IMPORTER	Key Generate2, PKA Key Generate, Key Test2, Key Translate2, Restrict Key Attribute, Secure Key Import2, Symmetric Key Import2
MAC class These keys are used to generate and verify a message authentication code (MAC).	

Table 4. 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 CMAC Generate, DK Random PIN Generate, 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	DK Deterministic PIN Generate, DK Migrate PIN, DK PAN Translate, DK PIN Change, DK PRW Card Number Update, DK Random PIN Generate, DK Regenerate PRW
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 Random PIN Generate, DK Regenerate PRW
Key generating class These keys are used to derive operational keys.	
DKYGENKY	Diversified Key Generate2

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 5. 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 either the Secure Key Import callable service or the Multiple Secure Key Import callable service.

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

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

Table 6. 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

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 discreet 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 7. 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 8. 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.

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 **RKX/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. CSNBTBC 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 private key** access control points will override this restriction.

DES master key

Since ICSF only allows a 16-byte DES master key to be loaded, ICSF cannot be compliant for key strength for 24-byte operational keys wrapped by the DES master key. Starting with ICSF release HCR77A0, a 24-byte master key can be loaded. Only cryptographic coprocessors with at least the October, 2012 licensed internal code support this key length. The **DES master key – 24-byte key** access control point must be enabled in the domain role. See the *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:
- "DK Deterministic PIN Generate (CSNBDDPG and CSNEDDPG)"
- "DK Migrate PIN (CSNBDMP and CSNEDMP)"
- "DK PAN Modify in Transaction (CSNBDPMT and CSNEDPMT)"
- "DK PAN Translate (CSNBDPT and CSNEDPT)"
- "DK PIN Change (CSNBDPC and CSNEDPC)"
- "DK PIN Verify (CSNBDPV and CSNEDPV)" on page 30
- "DK PRW Card Number Update (CSNBDPNU and CSNEDPNU)" on page 30
- "DK PRW CMAC Generate (CSNBDPCG and CSNEDPCG)" on page 30
- "DK Random PIN Generate (CSNBDRPG and CSNEDRPG)" on page 30
- "DK Regenerate PRW (CSNBDRP and CSNEDRP)" on page 30

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 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.

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)

The DK PRW Card Number Update (CSNBDPNU and CSNEDPNU) service is 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.

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)

The DK Random PIN Generate service generates 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.

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.

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 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.

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. Use of this service requires the optional PCI Cryptographic Coprocessor.

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 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 18.

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 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.

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. Triple-length keys can only be imported as DATA keys. 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.

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 PKA Cryptography and Using PKA Callable Services,” on page 81 for additional information.

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

This service transfers an application-supplied symmetric key (a DATA key) from encryption under the DES host master key to encryption under an application-supplied RSA public key. The application-supplied DATA key must be an ICSF DES 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 symmetric key (that is, a DATA key) and returns it encrypted using DES and encrypted under an RSA public key token.

The DES-encrypted key can be an internal token encrypted under a host DES master key, or an external form encrypted under a KEK. (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 (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 ICSF supports. You can also use this service to build CCA key tokens for all key types 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 form 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 as a container 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 as a container 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 the above, one may 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.

New remote key loading methods: New remote key loading methods have been developed to overcome some of the shortcomings of the old manual key loading methods. These new methods define acceptable techniques using public key cryptography to load keys remotely. Using these new 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 new cryptographic features added provide new 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 new data structure called a *trusted block* is defined to facilitate the remote key loading methods. The trusted block is the primary vehicle supporting these new 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 40 shows the contents of a trusted block. The elements shown in the table give an overview of the structure and do not provide all of the details of a trusted block.

Structure version information	
Public key	Modulus
	Exponent
	Attributes
Trusted block protection information	MAC key
	MAC
	Flags
	MKVP
	Activation/Expiration dates
Public key name (optional)	Label
Rules	Rule 1
	Rule 2
	Rule 3
	...
	Rule N
Application defined data	Data defined and understood only by the application using the 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 44. 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.

Field name	Required field	Description
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.

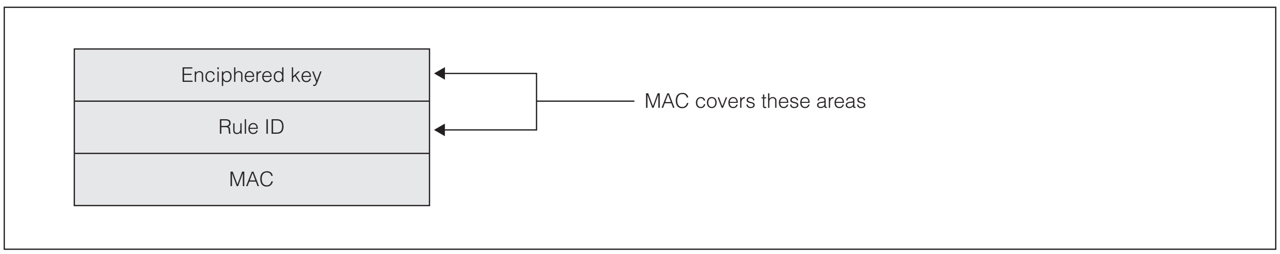

Figure 2. Simplified RKX key-token structure

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.

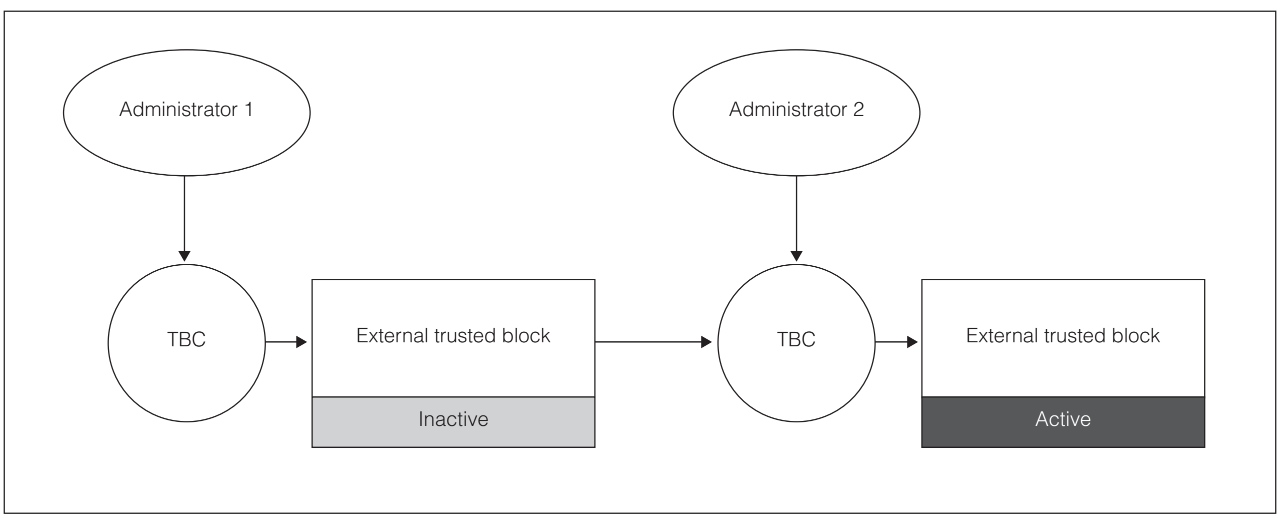

Figure 3. Trusted block creation

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.

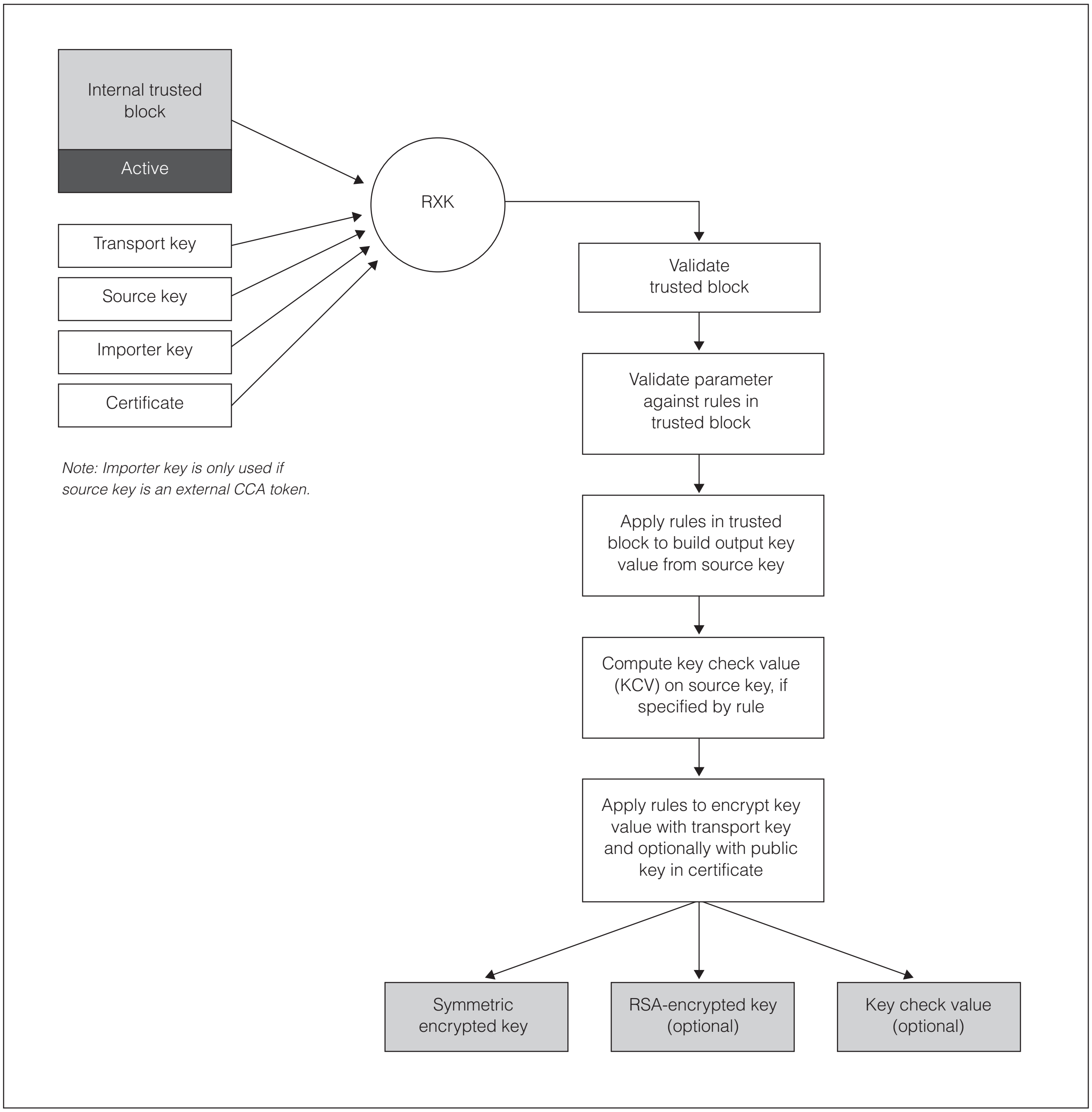

Figure 4. Exporting keys using a trusted block

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 resulting 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.

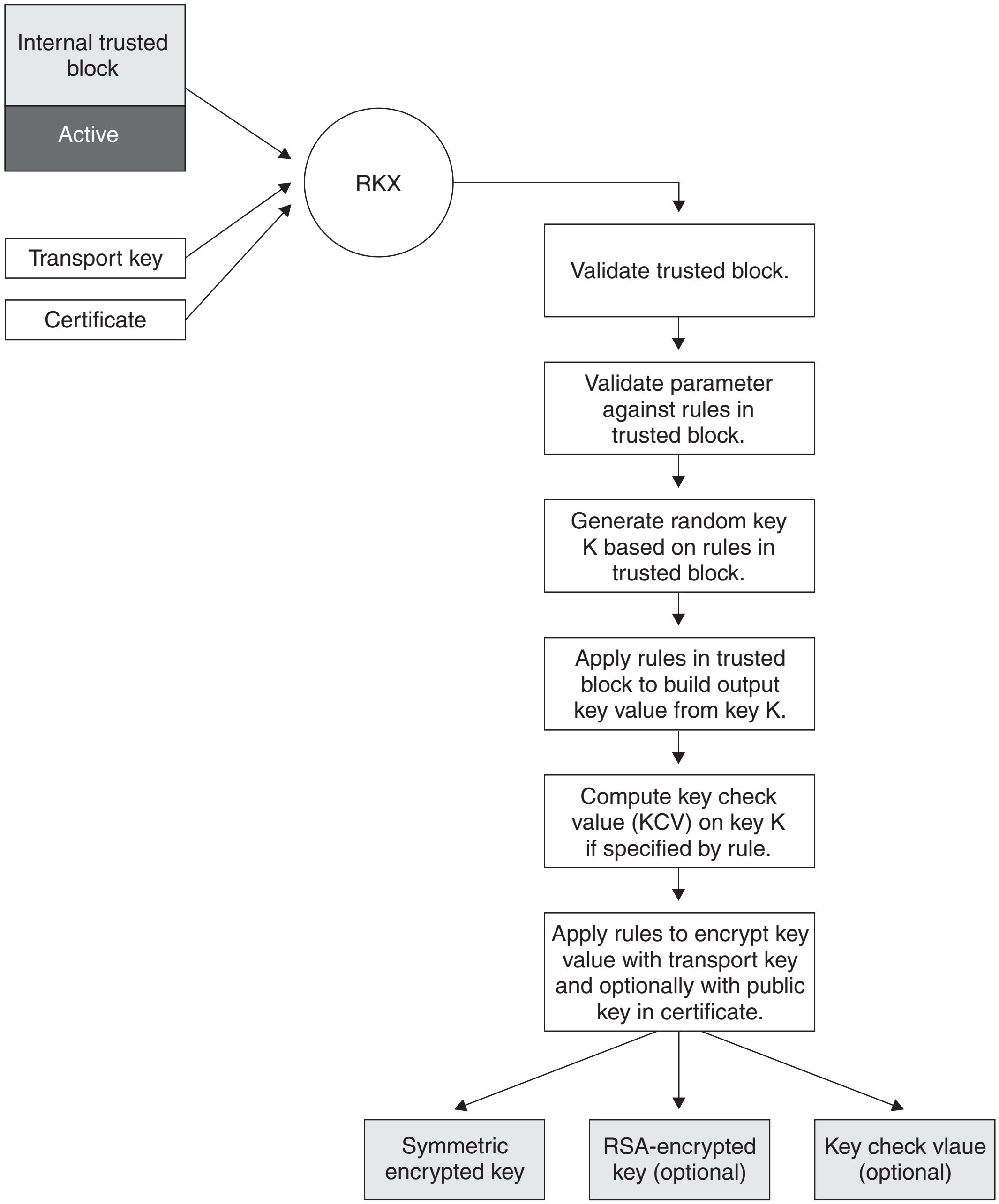

Figure 5. Generating keys using a trusted block

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:

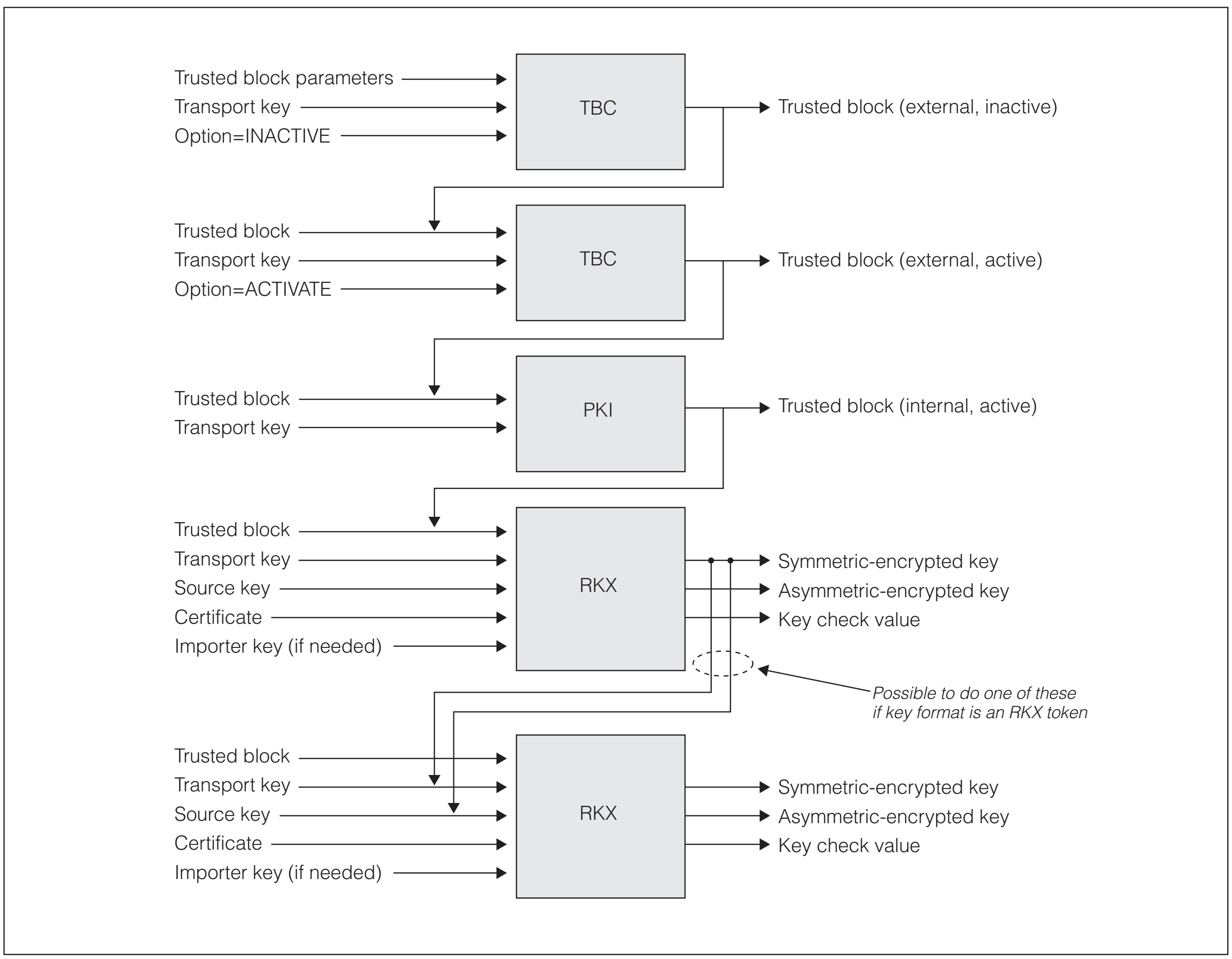

Figure 6. Typical flow of callable services for remote key export

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, and (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 under KEK2, 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 solves one new problem, and one long-standing problem. This support 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 far easier and far 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 for customers to develop more secure systems.

Diversifying keys

CCA supports several methods for diversifying a key using the diversified key generate callable service. 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 lest the interactions with the whole population of cards be placed in jeopardy.

In the current implementation, several methods of diversifying a key are supported: **CLR8-ENC**, **TDES-CBC**, **TDES-ENC**, **TDES-DEC**, **SESS-XOR**, **TDES-XOR**, **TDESEMV2** and **TDESEMV4**. The first three methods triple-encrypt data using the generating_key to form the diversified key. The diversified key is then multiply-enciphered by the 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 **SESS-XOR** method provides a means for modifying an existing DATA, DATAC, MAC, DATAM, or MACVER, DATAMV single- or double-length key. The provided data is exclusive-ORed into the clear value of the key. This form of key diversification is specified by several of the credit card associations.

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 – 22 associated with the key-generating key specify the permissible type of final key. (See DKYGENKY in Figure 20 on page 1070.) Control-vector bits 12 – 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 – 14 specify a counter that is decreased by one each time the diversified key generate service is used to produce another key-generating key. For example, if the key-generating key that you specify has this counter set to B'010', then you must specify the control vector for the generated_key with a DKYGENKY key type having the counter bits set to B'001' and specifying the same final key type in bits 19 – 22. Use of a generating_key with bits 12 – 14 set to B'000' results in the creation of the final key. Thus 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 **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 at "Visa and EMV-related smart card formats and processes" on page 1114. Refer to "Working with Europay–MasterCard–Visa smart cards" on page 492 to understand the various verbs you can use to operate with EMV smart cards.

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 syntax of the CKDS Key Record Create, CKDS Key Record Delete, CKDS Key Record Read, and CKDS 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.

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 algorithm is documented in the Federal Information Processing Standard #46. The algorithm is documented in the Federal Information Processing Standard #192.

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 Ciphertext Translate2 Callable Service" on page 68.

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, RIPEMD-160 and MD5.

(See Chapter 10, "Using Digital Signatures," on page 707 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 verification callable service. This support permits ICSF MAC services to interface more smoothly with non-CCA key distribution system.

HMAC Generation 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 generation callable service. The callable service computes the message authentication code using FIPS-198 Keyed-Hash Message Authentication Code method.

HMAC Verification Callable Service (CSNBHMV or CSNBHMV1 and CSNEHMV or CSNEHMV1)

When the receiver gets the message, an application program calls the HMAC verification 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 verification callable service can use FIPS-198 Keyed-Hash Message Authentication Code method.

MAC Generation 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 generation 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 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 MAC or a double-length DATA key and the message text.

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 Generation2 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 generation2 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 Verification Callable Service (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)

When the receiver gets the message, an application program calls the MAC verification 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 verification 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 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 MAC or a double-length DATA key and the message text.

The method used to verify the MAC should correspond with the method used to generate the MAC.

MAC Verification2 Callable Service (CSNBMVR2 or CSNBMVR3 and CSNEMVR2 or CSNEMVR3)

When the receiver gets the message, an application program calls the MAC verification2 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 verification 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
- RIPEMD-160
- MD5

MDC Generation 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 generation callable service are:

- MDC-2
- MDC-4
- PADMDC-2

1. The Secure Hash Algorithm (SHA) is also called the Secure Hash Standard (SHS), which Federal Information Processing Standard (FIPS) Publication 180 defines.

- 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 generation 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.

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
- IBM German Bank Pool PIN Offset (GBP-PINO)
- 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
- 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.

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 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 509 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 518 for more information.

Note: The PIN block must be encrypted under either an input PIN-encrypting key (IPINENC) or output PIN-encrypting key (OPINENC). Using an IPINENC key requires NOCV keys to be enabled in the CKDS.

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 generation callable service. Using a PIN generation algorithm, data used in the algorithm, and

2. The VISA CVV and the MasterCard CVC refer to the same value. CVV is used here to mean both CVV and CVC.

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 529 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 534 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 491 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 540 for more information.

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.

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 589 for more information.

Transaction Validation Callable Service (CSNBTRV and CSNETRV)

To support generation and validation of American Express card security codes, call the transaction validation callable service.

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.

Authentication Parameter Generate (CSNBAPG and CSNEAPG)

This callable service will calculate an authentication parameter (AP) and optionally return it encrypted under an encrypting key.

ANSI TR-31 key block support

A 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.

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 Export 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.

| The TR-31 key block support requires a z196 or IBM zEnterprise EC12 or later with
| a CCA Cryptographic coprocessor that is a CEX3C or later with Licensed Internal
| Code (LIC) of Sept. 2011 or later. Only DES/TDES keys can be transported in
| TR-31 key blocks. There is no support for transporting AES keys.

TR-31 Export Callable Service (CSNBT31X and CSNET31X)

The TR-31 Export callable service converts a CCA token to TR-31 format for export to another party. 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 exported key through the rule array.

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.

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 1121 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.

On z890, z990 z9 EC, z9 BC, z10 EC and z10 BC systems running with May 2004 or higher version of Licensed Internal Code or an z9 EC, z9 BC, z10 EC and z10 BC with MCL 029 Stream J12220 or higher of Licensed Internal Code, you must enable TKE commands for each PCIXCC or 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.

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.

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.

Typical Sequences of ICSF Callable Services

Sample sequences in which the ICSF callable services might be called are shown in Table 9.

Table 9. 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 144.

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 38 on page 152 and Table 39 on page 152 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” 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 Ciphertext Translate2 Callable Service

Restriction: The Ciphertext Translate2 callable service is only available on the IBM zEnterprise EC12 and later servers.

This topic describes a scenario using the encipher, Ciphertext 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 Ciphertext 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 CIPHERX0 CIPHERXI.
4. Send the exportable CIPHERX0 key to node B.
5. Send the exportable CIPHERXI key to node C.
6. Node C and node D need to share a CIPHERX0 key and a CIPHER key. Node D can generate one key in two forms: OPEX CIPHERX0 CIPHERXI.
7. Node D sends the exportable CIPHERX0 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  CIPHERX0    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 10 lists the callable services described in this publication, and their corresponding verbs. The figure also references the topic that describes the callable service.

Table 10. Summary of ICSF Callable Services

Verb	Service Name	Function
Chapter 5, "Managing Symmetric Cryptographic Keys," on page 101		
CSNBCKI CSNECKI	Clear key import	Imports an 8-byte clear 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 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.
CSNBDKX CSNEDKX	Data key export	Converts a DATA key from operational form into exportable form.
CSNBDKM CSNEDKM	Data key import	Imports an encrypted source DES single- or double-length DATA key and creates or updates a target internal key token with the master key enciphered source key.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBDKG CSNEDKG	Diversified key generate	Generates a 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.
CSNBEDH CSNEEDH	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).
CSNBKEX CSNEKEX	Key export	Converts any 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 key, or a pair of keys; and returns them in encrypted forms (operational, exportable, or importable). CSNBKGN does not produce keys in plaintext.
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 key from importable form into operational form.
CSNBKPI CSNEKPI	Key part import	Combines the clear key parts of any 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.
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 key types ICSF supports.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBKTB2 CSNEKTB2	Key token build2	Builds an internal clear key or skeleton token from the supplied parameters. 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 an 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 an input key and then enciphers this key using another key-encrypting key within the secure environment.
CSNBCKM CSNECKM	Multiple clear key import	Imports a single-, double-, or triple-length clear 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.
CSNBSKM CSNESKM	Multiple secure key import	Enciphers a single-, double-, or triple-length clear 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. Triple-length keys can only be imported as DATA keys. 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 key so that it cannot be exported.
CSNBPEXX CSNEPEXX	Prohibit export extended	Changes the external token of a 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 operational variable-length key so that it cannot be exported.
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.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
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 key type. This service executes only in special secure mode.
CSNBSKI2 CSNESKI2	Secure key import2	Enciphers a variable-length clear HMAC or AES 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 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 DATA key and returns the key in two forms: enciphered under the DES master key or KEK 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.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBT31X CSNET31X	TR-31 Export	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.
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 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 371		
CSNBCTT2 CSNBCTT3 CSNECTT2 CSNECTT3	Ciphertext 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.
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.)

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
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.
CSNBSYD CSNBSYD1 CSNESYD CSNESYD1	Symmetric key decipher	Deciphers data using the AES or DES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. Only clear 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.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBSYE CSNBSYE1 CSNESYE CSNESYE1	Symmetric key encipher	Enciphers data using the AES or DES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. Only clear 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 439		
CSNBHMG CSNEHMG CSNBHMG1 CSNEHMG1	HMAC generation	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 verification	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.
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 the ANSI X9.9-1 algorithm, ANSI X9.19 optional double key algorithm the EMV padding rules or the ISO 16609 TDES algorithm. 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.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
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 the ANSI X9.9-1 algorithm, ANSI X9.19 optional double key algorithmthe EMV padding rules or the ISO 16609 TDES algorithm. 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.
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 491		
CSNBAPG CSNEAPG	Authentication Parameter Generate	Generate an authentication parameter (AP) and optionally return it encrypted under a supplied encrypting key.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
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 or GBP-PINO) 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.
CSNBEPG CSNEEPG	Encrypted PIN generate	Generates and formats a PIN and encrypts the PIN block.
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.
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.
CSNBPTR CSNEPTR	Encrypted PIN translate	Reenciphers a PIN block from one PIN-encrypting key to another and, optionally, changes the PIN block format. UKPT keywords are supported.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
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 or GBP-PINO) VISA PIN validation value (VISA-PVV) Interbank PIN (INBK-PIN) UKPT keywords are supported.
CSNBPCU CSNEPCU	PIN Change/Unblock	Supports the PIN change algorithms specified in the VISA Integrated Circuit Card Specification; only available on a z890 or Requires May 2004 or later version of Licensed Internal Code (LIC).
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.
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).
Financial Services for DK PIN Methods		
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.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBDPV CSNEDPV	DK PIN Verify	Verifies an ISO-1 format PIN.
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.
CSNBDRPG CSNEDRPG	DK Random PIN Generate	Generates a PIN and a PIN reference value using the random process.
CSNBDRP CSNEDRP	DK Regenerate PRW	Generates a new PIN reference value for a changed account number.
Chapter 12, "Key data set management," on page 757		
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.

Table 10. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
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.
Chapter 13, "Utilities," on page 811		
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.
CSNB9ED	X9.9 data editing	Edits an ASCII text string according to the editing rules of ANSI X9.9–4.
Chapter 14, "Trusted Key Entry Workstation Interfaces," on page 851		
CSFPCI	PCI interface	Puts a request to a specific coprocessor queue and removes the corresponding response when complete. Only the Trusted Key Entry (TKE) workstation uses this service.

Chapter 3. Introducing 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:

  $\text{ciphertext} = \text{cleartext}^{PU} \pmod{M}$

  Similarly, this operation recovers cleartext from ciphertext:

  $\text{cleartext} = \text{ciphertext}^{PR} \pmod{M}$

  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 Digital Signature Algorithm (ECDSA)**

  The ECDSA algorithm uses elliptic curve cryptography (an encryption system based on the properties of elliptic curves) to provide a variant of the Digital Signature Algorithm.

PKA Master Keys

PKA master keys protect private keys.

- RSA keys are protected by the RSA Master Key (RSA-MK). The RSA-MK is a triple-length DES key used to protect RSA private keys. On the IBM zEnterprise 196 or later with a CCA cryptographic coprocessor that is a CEX3C or later, there is an additional master key: ECC. The ECC master key is a 256-bit AES key used to protect ECC private keys and new format RSA private keys.
- 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.

On systems with a PCIXCC or CCA Crypto Express coprocessor, changing the RSA master key requires that the PKA callable services control be disabled. The new master key value is loaded, the PKDS is reenciphered and the Change Asymmetric Master Key utility makes the reenciphered PKDS the active PKDS. The PKA callable services control will be enabled automatically.

On systems with a CCA Cryptographic coprocessor that is a CEX3C or later, the ECC master key is changed in the same manner as the DES and AES master keys. On systems with the CEX3C (with the September 2011 licensed internal code) or by any CCA cryptographic coprocessors that are a CEX4C or later, the RSA master key is changed in the same manner as the DES, AES, and ECC master 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 discreet 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 11. 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 12. 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.

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 **RKX/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. CSNBTBC 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 private key** 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.

Restrictions:

- ECDSA is only supported through a CCA cryptographic coprocessor that is a CEX3C or later hardware.

Digital Signature Generate Callable Service (CSNDDSG and CSNFDSG)

This service generates a digital signature using an RSA or ECC private key. It supports these methods of signature generation:

- ANSI X9.30 (ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

The input text must have been previously hashed using the one-way hash generate callable service or the MDC generation service.

Digital Signature Verify Callable Service (CSNDDSV and CSNFDSG)

This service verifies a digital signature using an RSA or ECC public key. This service supports these methods of signature generation:

- ANSI X9.30 (ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

The text that is input to this service must be previously hashed using the one-way hash generate callable service or the MDC generation 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 751 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 CSNFKRL)

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 SET Secure Electronic Transaction

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 or ECC private or public keys. PKA tokens are variable length because they contain either RSA or ECC key values, which are variable in length. Consequently, length parameters precede all PKA token parameters. The maximum allowed size is 3500 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 or ECC 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 13 summarizes the sections in each type of token.

Table 13. Summary of PKA Key Token Sections

Section	Public External Key Token	Private External Key Token	Private Internal Key Token
Header	X	X	X
RSA or ECC private key information		X	X
RSA or ECC public key information	X	X	X
Key name (optional)		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 or ECC key tokens begin in "RSA public key token" on page 1018.

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.

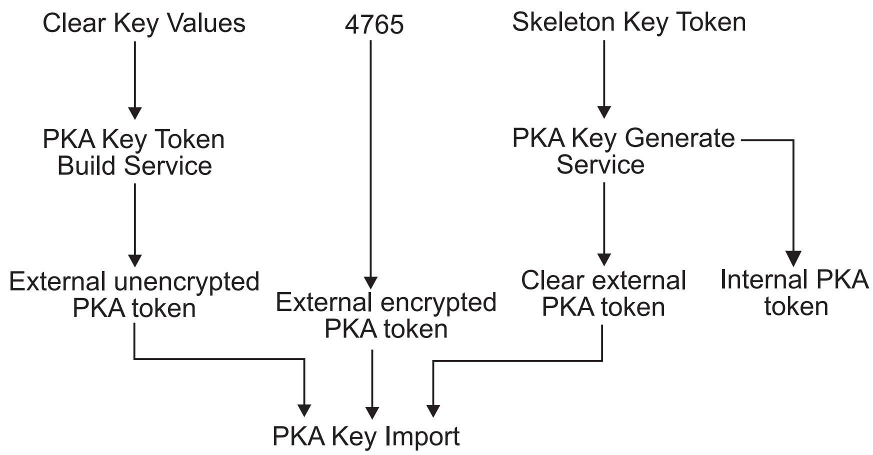

Figure 7. PKA Key Management

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 729).

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.

Application RSA or ECC public and private keys can be stored in the public key data set (PKDS), a VSAM data set.

Security and Integrity of the Token

PKA private key tokens may optionally have a 64-byte *private_key_name* field. If *private_key_name* exists, ICSF uses RACROUTE REQUEST=AUTH to verify it prior to using the token in a callable service. For additional security, the processor also validates the entire private key token.

Key Identifier for PKA Key Token

A *key identifier* for a PKA key token is a variable length (maximum allowed size is 3500 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:

Offset	Length	Data
00-63	64	Key label name

Key Token

A key token is a variable length (maximum allowed size is 3500 bytes) field composed of key value and control information. PKA keys can be either public or private RSA or ECC 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 995.

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 those listed. The callable services are described in detail in Chapter 11, "Managing PKA Cryptographic Keys," on page 719.

- 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 995 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 those listed. They are described in detail in Chapter 11, "Managing PKA Cryptographic Keys," on page 719.

- PKA public key extract
- PKA key token build
- PKA key generate

For debugging information, see Appendix B, "Key Token Formats," on page 995 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 995 for the format of a null key token.

Summary of the PKA callable services

Table 14 lists the PKA callable services, described in this publication, and their corresponding verbs. (The PKA services start with CSNDxxx and have corresponding CSFxxx names.) This table also references the topic that describes the callable service.

Table 14. Summary of PKA callable services

Verb	Service Name	Function
Chapter 8, "Financial Services," on page 491		
CSNDSBC CSNFSBC	SET block compose	Composes the RSA-OAEP block and the DES-encrypted block in support of the SET protocol.
CSNDSBD	SET block decompose	Decomposes the RSA-OAEP block and the DES-encrypted block to provide unencrypted data back to the caller.
Chapter 10, "Using Digital Signatures," on page 707		

Table 14. Summary of PKA callable services (continued)

Verb	Service Name	Function
CSNDDSG CSNFDSG	Digital signature generate	Generates a digital signature using a PKA private key supporting RSA and ECDSA algorithms.
CSNDDSV CSNFDSV	Digital signature verify	Verifies a digital signature using a PKA public key supporting RSA and ECDSA algorithms.
Chapter 11, "Managing PKA Cryptographic Keys," on page 719		
CSNDPKG CSNFPKG	PKA key generate	Generate RSA and ECC 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.
CSNDPKB CSNFPKB	PKA key token build	Creates an external PKA key token containing a clear private RSA or ECC 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 or ECC 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 ECC or RSA key token.
CSNDKTC CSNFKTC	PKA key token change	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.
CSNDPKT CSNFPKT	PKA key translate	Translates a CCA RSA key token to a smart card 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 12, "Key data set management," on page 757		
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.

Table 14. Summary of PKA callable services (continued)

Verb	Service Name	Function
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.
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.

Many services have hardware requirements. 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 see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

PKCS #11 Services

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 Part 3, “PKCS #11 Callable Services,” on page 859.

Table 15. 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 HMAC	Generate a hashed message authentication code (MAC).
CSFPGKP	PKCS #11 Generate key pair	Generate an RSA, DSA, Elliptic Curve, or Diffie-Hellman key pair.

Table 15. Summary of PKCS #11 callable services (continued)

Verb	Service Name	Function
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.
CSFPPRF	PKCS #11 Pseudo-random function	Generate pseudo-random output of arbitrary length.
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.
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.
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 HMAC	Verify a hash message authentication code (MAC).
CSFPWPK	PKCS #11 Wrap key	Wrap a key with another key.

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 a secure session object

Y the handle belongs to a secure token object

Z the handle belongs to a 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 publication 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 and maintaining the cryptographic key data set (CKDS).

This topic describes these callable services:

- "Clear Key Import (CSNBCKI and CSNECKI)" on page 102
- "Control Vector Generate (CSNBCVG and CSNECVG)" on page 104
- "Control Vector Translate (CSNBCVT and CSNECVT)" on page 108
- "Cryptographic Variable Encipher (CSNBCVE and CSNECVE)" on page 112
- "Data Key Export (CSNBDKX and CSNEDKX)" on page 115
- "Data Key Import (CSNBDKM and CSNEDKM)" on page 118
- "Diversified Key Generate (CSNBDKG and CSNEDKG)" on page 120
- "Diversified Key Generate2 Callable Service (CSNBDKG2 and CSNEDKG2)" on page 127
- "ECC Diffie-Hellman (CSNDEDH and CSNFEDH)" on page 132
- "Key Export (CSNBKEX and CSNEKEX)" on page 140
- "Key Generate (CSNBKGN and CSNEKGN)" on page 144
- "Key Generate2 (CSNBKGN2 and CSNEKGN2)" on page 156
- "Key Import (CSNBKIM and CSNEKIM)" on page 168
- "Key Part Import (CSNBKPI and CSNEKPI)" on page 172
- "Key Part Import2 (CSNBKPI2 and CSNEKPI2)" on page 176
- "Key Test (CSNBKYT and CSNEKYT)" on page 180
- "Key Test2 (CSNBKYT2 and CSNEKYT2)" on page 185
- "Key Test Extended (CSNBKYTX and CSNEKTX)" on page 190
- "Key Token Build (CSNBKTB and CSNEKTB)" on page 194
- "Key Token Build2 (CSNBKTB2 and CSNEKTB2)" on page 203
- "Key Translate (CSNBKTR and CSNEKTR)" on page 232
- "Key Translate2 (CSNBKTR2 and CSNEKTR2)" on page 235
- "Multiple Clear Key Import (CSNBCKM and CSNECKM)" on page 242
- "Multiple Secure Key Import (CSNBSKM and CSNESKM)" on page 246
- "PKA Decrypt (CSNDPKD and CSNFPKD)" on page 251
- "PKA Encrypt (CSNDPKE and CSNFPKE)" on page 256
- "Prohibit Export (CSNBPEX and CSNEPEX)" on page 261
- "Prohibit Export Extended (CSNBPEXX and CSNEPEXX)" on page 263
- "Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)" on page 265
- "Remote Key Export (CSNDRKX and CSNFRKX)" on page 269
- "Restrict Key Attribute (CSNBRKA and CSNERKA)" on page 279
- "Secure Key Import (CSNBSKI and CSNESKI)" on page 283

- “Secure Key Import2 (CSNBSKI2 and CSNESKI2)” on page 287
- “Symmetric Key Export (CSNDSYX and CSNFSYX)” on page 292
- “Symmetric Key Export with Data (CSNDSXD and CSNFSXD)” on page 298
- “Symmetric Key Generate (CSNDSYG and CSNFSYG)” on page 302
- “Symmetric Key Import (CSNDSYI and CSNFSYI)” on page 309
- “Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)” on page 314
- “Trusted Block Create (CSNDTBC and CSNFTBC)” on page 319
- “TR-31 Export (CSNBT31X and CSNET31X)” on page 323
- “TR-31 Import (CSNBT31I and CSNET31I)” on page 337
- “TR-31 Optional Data Build (CSNBT31O and CSNET31O)” on page 351
- “TR-31 Optional Data Read (CSNBT31R and CSNET31R)” on page 354
- “TR-31 Parse (CSNBT31P and CSNET31P)” on page 358
- “Unique Key Derive (CSNBUKD and CSNEUKD)” on page 361

Clear Key Import (CSNBCKI and CSNECKI)

Use the clear key import callable service to import a clear 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 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: To import 16-byte or 24-byte DATA keys, use the multiple clear key import callable service that is described in “Multiple Clear Key Import (CSNBCKM and CSNECKM)” on page 242. The multiple clear key import service supports 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 and reason codes,” on page 943 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 and reason codes," on page 943 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
Input/Output	String

A 64-byte string that is to receive the internal key token. "Key Identifier for Key Token" on page 8 describes the internal key token.

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.

Clear Key Import

Table 16. Clear key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor/Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	
IBM z13	Crypto Express5 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 and reason codes," on page 943 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 and reason codes,” on page 943 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
- 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 194 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:

- AMEX-CSC
- ANSIX9.9
- ANY
- ANY-MAC
- CLR8-ENC
- CPINENC
- CPINGEN
- CPINGENA
- CVVKEY-A
- CVVKEY-B
- DALL
- DATA
- DDATA
- DEXP
- DIMP
- DKYL0
- DKYL1
- DKYL2
- DKYL3
- DKYL4
- DKYL5
- DKYL6
- DKYL7

- DMAC
- DMKEY
- DMPIN
- DMV
- DOUBLE
- DOUBLE-O
- DPVR
- ENH-ONLY
- EPINGEN
- EPINGENA
- EPINVER
- EXEX
- EXPORT
- GBP-PIN
- GBP-PINO
- IBM-PIN
- IBM-PINO
- IMEX
- IMIM
- IMPORT
- INBK-PIN
- KEY-PART
- KEYLN8
- KEYLN16
- LMTD-KEK
- MIXED
- NO-SPEC
- NO-XPORT
- NON-KEK
- NOOFFSET
- NOT31XPT
- OPEX
- OPIM
- REFORMAT
- SINGLE
- SMKEY
- SMPIN
- T31XPTOK
- TRANSLAT
- UKPT
- VISA-PVV
- XLATE
- XPORT-OK

Note: CLR8-ENC or UKPT must be coded in *rule_array* when the KEYGENKY key type is coded. When the SECMSG *key_type* is coded, either SMKEY or SMPIN must be specified in the *rule_array*. ENH-ONLY is not supported with key type DATA.

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 69 on page 201 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 1075 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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.

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 17. Keywords for Control Vector Translate

Keyword	Meaning
Parity Adjustment Rule (optional)	

Table 17. Keywords for Control Vector Translate (continued)

Keyword	Meaning
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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *target_key_token* will be wrapped in the same manner as the input *source_key_token*.

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.

Table 18. Control vector translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ENH-ONLY, USECONFG, WRAP-ENC and WRAP-ECB not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ENH-ONLY, USECONFG, WRAP-ENC and WRAP-ECB not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ENH-ONLY, USECONFG, WRAP-ENC and WRAP-ECB not supported.
	Crypto Express3 Coprocessor	Enhanced key token wrapping not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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 and reason codes," on page 943 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 and reason codes," on page 943 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.

Table 19. Cryptographic variable encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	

Table 19. Cryptographic variable encipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM z13	Crypto Express5 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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

A 64-byte string for an internal key token or key label that contains the exporter *key_encrypting* key. The data-encrypting key previously discussed will be encrypted under this exporter *key_encrypting* 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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *target_key_identifier* will be wrapped in the same manner as the *source_key_identifier*.

Restrictions

For existing TKE users, you may have to explicitly enable new access control points. 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.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 20. 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 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.

Table 21. Data key export required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes," on page 943 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 and reason codes," on page 943 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

A 64-byte string variable containing the (IMPORTER) transport key or key label of the transport key used to decipher the source 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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. 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.

Restrictions

For existing TKE users, you may have to explicitly enable new access control points. 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.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 22. 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.

Table 23. Data key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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,
            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 and reason codes," on page 943 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 and reason codes,” on page 943 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 24. Rule Array Keywords for Diversified Key Generate

Keyword	Meaning
Processing Method for generating or updating diversified keys (required)	
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.

Table 24. Rule Array Keywords for Diversified Key Generate (continued)

Keyword	Meaning
SESS-XOR	Modifies an existing DATA, DATAC, MAC, DATAM, or MACVER, DATAMV single- 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, then 8-bytes of data is TDES decrypted under the *generating_key_identifier*. If the *generated_key_identifier* specifies a double length key, then 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.
TDES-ENC	Data supplied may be 8 or 16 bytes of clear data. If the *generated_key_identifier* specifies a single length key, then 8-bytes of data is TDES encrypted under the *generating_key_identifier*. If the *generated_key_identifier* specifies a double length key, then 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, then 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.

Table 24. Rule Array Keywords for Diversified Key Generate (continued)

Keyword	Meaning
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*.
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
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 *generated_key_identifier* token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.

generating_key_identifier

Direction	Type
Input/Output	String

The label or internal 64 byte token of a key-generating key. The type of key depends on the processing method.

data_length

Direction	Type
Input	Integer

The length of the *data* parameter that follows. Length depends on the processing method and the generated key. The data length for TDESEMV4 or TDESEMV2 is either 18 or 34.

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) or 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

key_identifier

Direction	Type
Input/Output	String

This parameter is currently not used. It must be a 64-byte null token.

generated_key_identifier

Direction	Type
Input/Output	String

The internal token of an operational key, a skeleton token containing the control vector of the key to be generated, or a null token. A null token can be supplied if the *generated_key_identifier* will be a DKYGENKY with a CV derived from the *generating_key_identifier*. A skeleton token or internal token is required when *generated_key_identifier* will not be a DKYGENKY key type or the processing method is not SESS-XOR. For SESS-XOR, this must be a null token. On output, this parameter contains the generated key.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *generated_key_token* will use the default method unless a rule array keyword overriding the default is specified.

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, "Control Vectors and Changing Control Vectors with the CVT Callable Service," on page 1065 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, etc.

  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.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 25. Required access control points for Diversified Key Generate

Rule array keyword	Access control point
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
WRAP-ECB or WRAP-ENH and default key-wrapping method setting does not match the keyword	Diversified Key Generate - Allow wrapping override keywords

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.

Table 26. Diversified key generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Keywords ENH-ONLY, USECONFG, WRAP-ENH, WRAP-ECB and TDES-CBC not supported. Enhanced key token wrapping not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Keywords ENH-ONLY, USECONFG, WRAP-ENH, WRAP-ECB and TDES-CBC not supported. Enhanced key token wrapping not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	Keywords ENH-ONLY, USECONFG, WRAP-ENH, WRAP-ECB and TDES-CBC not supported. Enhanced key token wrapping not supported.
	Crypto Express3 Coprocessor	Keyword TDES-CBC is not supported. Enhanced key token wrapping not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Keyword TDES-CBC is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	TDES-CBC support requires the Sep. 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

Diversified Key Generate2 Callable Service (CSNBDKG2 and CSNEDKG2)

The diversified key generate2 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.
  - Key usage field 1 determines the type of key that is generated and restricts the use of this key to the key-diversification process.
  - Key usage field 2 contains a flag to determine how key usage fields 3 through 6 control the key usage fields of the generated key.
    - When the flag is on, the key usage fields of the DKYGENKY must be equal (KUF-MBE or 'KUF – must be equal') to the key usage fields of the generated key.
    - When the flag is off, the key usage fields of the DKYGENKY limit the values of the key usage fields of the generated key (KUF-MBP, or 'KUF must be permitted').

  For the service to be valid, the generated key cannot have usage that is not enabled in the DKYGENKY key. The UDX-ONLY bit is always treated as 'must be equal'.

– Key usage fields 3 through 6 in the key generating key indicate the key usage attributes for the key to be generated.

Note: The only exception to this rule is when the type of key to diversify is D-ALL.

- The data and length of data used in the diversification process.
- The AES key token with a suitable key usage field for receiving the diversified key.

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,
            reserved1_length,
            reserved1,
            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 and reason codes," on page 943 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 and reason codes," on page 943 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

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 27. Rule array keywords for diversified key generate2

Keyword	Meaning
Diversification Process (required)	
SESS-ENC	A session key is created by enciphering a 16-byte diversification value with the *k*-bit AES key-generating key to produce a *k*-bit AES session key using the AES algorithm in ECB mode, where *k* is 128, 192 or 256 bits.

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 725.

generating_key_identifier

Direction	Type
Input/Output	String

The identifier of the key-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 and the key type must be DKYGENKY. The key usage field indicates the key type 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.

If the token supplied was encrypted under the old master key, the token 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 is specified, the value must be 16.

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.

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.

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 725 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 generated key token.

On input, the buffer contains a valid internal skeleton token 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 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. D-ALL permits the derivation of several different keys. 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.

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.

Access control points

The **Diversified Key Generate2 – AES EMV1 SESS** access control point in the domain role controls the function of this service.

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.

Table 28. Diversified key generate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Requires the November 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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.

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.
- “Z” – The “secret” material output from D-H process.

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,
chaining_vector_length,
chaining_vector,
party_identifier_length,
party_identifier,
key_bit_length,
reserved_length,
reserved,
reserved2_length,
reserved2,
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 and reason codes," on page 943 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 and reason codes," on page 943 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 between 1 and 5.

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 29. Keywords for ECC Diffie-Hellman

Keyword	Meaning
Key agreement (one required)	
DERIV01	Use the static unified model key agreement scheme.
PASSTHRU	Skip Key derivation step and return raw “Z" material.
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.
Key Wrapping Method (one optional, only supported when the output type is DES)	
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.
WRAP-ECB	Specifies that the original wrapping method is to be used.
Translation Control (one optional, only supported when the output type is DES)	
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 (ENH ONLY) in the control vector. This keyword is not valid if processing a zero CV data key.

private_key_identifier_length

Direction	Type
Input	Integer

The length of the *private_key_identifier* parameter.

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. Clear keys will be accepted.

private_KEK_key_identifier_length

Direction	Type
Input	Integer

The length of the *private_KEK_key_identifier* in bytes. The maximum value is 900. 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 *private_KEK_key_identifier* must contain a KEK key token, the label of a KEK key token, or a null token. The KEK key token must be present if the *private_key_identifier* contains an external ECC token.

public_key_identifier_length

Direction	Type
Input	Integer

The length of the *public_key_identifier*.

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.

chaining_vector_length

Direction	Type
Input/Output	Integer

The *chaining_vector_length* parameter must be zero.

chaining_vector

Direction	Type
Input/Output	String

The *chaining_vector* parameter is ignored.

party_identifier_length

Direction	Type
Input/Output	Integer

The length of the *party_identifier* parameter. Valid values are 0, or between 8 and 64. The *party_identifier_length* must be 0 when the PASSTHRU rule array keyword is specified.

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.

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.

reserved_length

Direction	Type
Input/Output	Integer

The *reserved_length* parameter must be zero.

reserved

Direction	Type
Input/Output	String

This parameter is ignored.

reserved2_length

Direction	Type
Input/Output	Integer

The *reserved2_length* parameter must be zero.

reserved2

Direction	Type
Input/Output	String

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*. The maximum value is 900. The *output_KEK_key_identifier_length* must be zero if *output_key_identifier* will contain an internal token or if the PASSTHRU rule array keyword was specified.

output_KEK_key_identifier

Direction	Type
Input/Output	String

The *output_KEK_key_identifier* contains a KEK key token or the label of a KEK key if the *output_key_identifier* will contain an external ECC token. Otherwise this field is ignored.

If the output KEK key identifier identifies a DES KEK, then it must be an IMPORTER or an EXPORTER key type, and have the export bit set. The XLATE bit is not checked. If the output KEK key identifier identifies an AES KEK, then it must be either an IMPORTER or an EXPORTER key type and have the export/import bit set in key usage field 1 and the derivation bit set in key usage field 4.

output_key_identifier_length

Direction	Type
Input/Output	Integer

The length of the *output_key_identifier*. 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 900 bytes.

output_key_identifier

Direction	Type
Input/Output	String

On input, the *output_key_identifier* must contain a skeleton token or a null token.

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 KEK 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 KEK 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 30. 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
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.

Access control points

The ECC Diffie-Hellman callable service requires the **ECC Diffie-Hellman Callable Service** access control point to be enabled in the active role.

Specifying the PASSTHRU rule array keyword requires that the **ECC Diffie-Hellman – Allow PASSTHRU** access control point be enabled in the active role.

If the *output_key_identifier* parameter references a DES key token and the wrapping method specified in not the default method, then the **ECC Diffie-Hellman – Allow key wrap override** access control point must be enabled in the active 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

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.

Table 31. ECC Diffie-Hellman required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	ECC Clear Key and Internal tokens support requires the Sep. 2010 licensed internal code (LIC). ECC External and Diffie-Hellman support requires Sep. 2011 licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

Key Export (CSNBKEX and CSNEKEX)

Use the key export callable service to reencipher any type of key (except an IMP-PKA) from encryption under a master key variant to encryption under the same variant of an exporter key-encrypting key. The reenciphered key can be exported to another system.

If the key to be exported is a DATA key, the key export service generates a key token with the same key length as the input token's key.

This service supports the no-export bit that the prohibit export service sets in the internal token.

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 and reason codes," on page 943 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 and reason codes," on page 943 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

A 64-byte string of the internal key token or key label that contains the exporter key-encrypting key. 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 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 389 on page 997.

target_key_identifier

Direction	Type
Output	String

The 64-byte field external key token that contains the reenciphered key. The reenciphered key can be exchanged with another cryptographic system.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *target_key_identifier* will be wrapped in the same manner as the *source_key_identifier*.

Restrictions

For existing TKE users, you may have to explicitly enable new access control points. 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.

If running with a PCIXCC, CEX2C, or CEX3C, existing internal tokens created with key type MACD must be exported with either a TOKEN or DATAM key type. The external CV will be DATAM CV.

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.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 32. 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.

Table 33. Key export required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Key types CIPHERXI, CIPHERXL, CIPHERXO and MACD are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

Key Generate (CSNBKGN and CSNEKGN)

Use the key generate callable service to generate either one or two odd parity DES keys of *any* type. The keys can be single-length (8 bytes), double-length (16 bytes), or, in the case of DATA keys, 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 an 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 and reason codes," on page 943 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 and reason codes," on page 943 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.

The first two characters refer to *key_type_1*. The next two characters refer to *key_type_2*.

These keywords are allowed: OP, IM, EX, OPIM, OPEX, IMEX, EXEX, OPOP, and IMIM. See Table 34 for their meanings.

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 34. 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 39 on page 152 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 35. Key Length values for the Key Generate callable service

Value	Description	Algorithm
SINGLE or KEYLN8	The key should be a single length (8-byte) key.	DES
SINGLE-R	The key should be a 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	The key should be a double length (16-byte or 128-bit) key	AES or DES
DOUBLE-O	The key should be a double length (16-byte) key. Each of the two key halves will be unique (not the same value).	DES
KEYLN24	The key should be a 24-byte (192-bit) key.	AES or DES
KEYLN32	The key should be a 32-byte (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. This key length is valid for DATA keys only. To generate a triple-length DATA key with three different values to be the basis of each key part, specify *key_length* as KEYLN24.

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 **Y** indicates that the key generated will be a double-length key with replicated key values. It is preferred that SINGLE-R be used for this result.

Table 36. Key lengths for DES keys

Key Type	Single - KEYLN8	Single-R	Double - KEYLN16	DOUBLE-O	KEYLN24
MAC MACVER	X X	X X	X X	X X	
DATA	X		X		X
DATAC* DATAM DATAMV		X X X	X X X	X X X	
EXPORTER IMPORTER	Y Y	X X	X X	X X	
IKEYXLAT OKEYXLAT	Y Y	X X	X X	X X	
CIPHER DECIPHER ENCIPHER	X X X	X X X	X X X	X X X	
IPINENC OPINENC PINGEN PINVER	Y Y Y Y	X X X X	X X X X	X X X X	
CVARDEC* CVARENC* CVARPINE* CVARXCVL* CVARXCVR*	X X X X X	X X X X X	X X X X X		
DKYGENKY* KEYGENKY*		X X	X X	X X	
CIPHERXI CIPHERXL CIPHERXO			X 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 37. Key lengths for AES keys

Key Type	128-byte	192-byte	256-byte
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 38 on page 152 and Table 39 on page 152 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 39 on page 152. 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 38 on page 152 and Table 39 on page 152 for valid key type and key form combinations.

KEK_key_identifier_1

Direction	Type
Input/Output	String

A 64-byte string of a DES internal key token containing the importer or exporter key-encrypting key, or a key label. If you supply a key label that is less than 64-bytes, it must be left-justified and padded with blanks. *KEK_key_identifier_1* is required for a *key_form* of IM, EX, IMEX, EXEX, or IMIM.

When key_form OP is used, parameters *KEK_key_identifier_1* and *KEK_key_identifier_2* are ignored. In this case, it is recommended that the parameters are initialized to 64-bytes of X'00'.

If 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 389 on page 997.

KEK_key_identifier_1 cannot be an AES key token or label.

KEK_key_identifier_2

Direction	Type
Input/Output	String

A 64-byte string of a DES internal key token containing the importer or exporter key-encrypting key, or a key label of an internal token. If you supply a key label that is less than 64-bytes, it must be left-justified and padded with blanks. *KEK_key_identifier_2* is required for a *key_form* of OPIM, OPEX, IMEX, IMIM, or EXEX. This field is ignored for *key_form* keywords OP, IM and EX. When key_form OP is used, parameter *KEK_key_identifier_2* is ignored. In this case, it is recommended that the parameter is initialized to 64-bytes of X'00'.

If 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 389 on page 997.

KEK_key_identifier_2 cannot be an AES key token or label.

generated_key_identifier_1

Direction	Type
Input/Output	String

This parameter specifies either a generated:

- 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.

If you specify a *key_type_1* of TOKEN, then this field contains a valid DES token of the key type you want to generate. Otherwise, on input, this parameter must be binary zeros. See *key_type_1* for a list of valid key types.

If you specify a *key_type_1* of IMPORTER or EXPORTER and a *key_form* of OPEX, and if the *generated_key_identifier_1* parameter contains a valid DES internal token of the SAME type, the NOCV bit, if on, is propagated to the generated key token.

When *key_type_1* parameter is AESDATA, then *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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. 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.

generated_key_identifier_2

Direction	Type
Input/Output	String

This parameter specifies either a generated:

- internal DES key token or
- external DES key token enciphered under *KEK_key_identifier_2*.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. 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.

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.

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.

Usage notes - Key type and key form combinations

Table 38 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 38. 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		
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 39 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 40 on page 154 for an explanation of the differences between **E** as compared to **X**.

Table 39. 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

Table 39. 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
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
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

Table 39. 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
PINGEN	PINVER	X	X		X

If you need to use NOCV key-encrypting keys, you need to enable NOCV IMPORTER and NOCV EXPORTER access control points

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 40. 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 38 on page 152.	Key Generate – OP
The key-form and key-type combinations shown with an 'X' in the Key_Form IM column in Table 38 on page 152.	Key Generate – Key set
The key-form and key-type combinations shown with an 'X 'in the Key_Form EX column in Table 38 on page 152.	Key Generate - Key set
The key-form and key-type combinations shown with an 'X' in Table 39 on page 152	Key Generate - Key set
The key-form and key-type combinations shown with an 'E' in Table 39 on page 152	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

Table 41 lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 41. Key generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported. Key length DOUBLE-O is not supported Secure AES keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported. Key length DOUBLE-O is not supported Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported. Key length DOUBLE-O is not supported Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported. Key length DOUBLE-O is not supported
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

Key Generate2 (CSNBKGN2 and CSNEKGN2)

Use the Key Generate2 callable service to generate either one or two keys 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 and reason codes," on page 943 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 and reason codes," on page 943 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 42. 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.

Table 42. Keywords for Key Generate2 Control Information (continued)

Keyword	Meaning
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.
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 45 on page 163 and Table 46 on page 164.

The 8-byte keyword for the *key_type_1* parameter can be one of the following:

Table 43. 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 46 on page 164. 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 44. Keywords and associated algorithms for key_type_2 parameter

Keyword	Algorithm
CIPHER	AES
EXPORTER	AES
IMPORTER	AES

Table 44. Keywords and associated algorithms for key_type_2 parameter (continued)

Keyword	Algorithm
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 between the actual length of the token and 725 bytes when *key_encrypting_key_identifier_1* is a token.

The value must be 64 bytes when *key_encrypting_key_identifier_1* is a label.

key_encrypting_key_identifier_1

Direction	Type
Input/Output	String

When *key_encrypting_key_identifier_1_length* is zero, this parameter is ignored. Otherwise, *key_encrypting_key_identifier_1* contains an internal key token containing the AES importer or exporter key-encrypting key, or a key label.

If the token supplied was encrypted under the old master key, the token 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 between the actual length of the token and 725 when *key_encrypting_key_identifier_2* is a token. The value must be 64 when *key_encrypting_key_identifier_2* is a label.

When only one key is being generated, this parameter is ignored.

key_encrypting_key_identifier_2

Direction	Type
Input/Output	String

When *key_encrypting_key_identifier_2_length* is zero, this parameter is ignored. Otherwise, *key_encrypting_key_identifier_2* contains an internal key token containing the AES importer or exporter key-encrypting key, or a key label.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

When only one key is being generated, this parameter is ignored.

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.

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.

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 45 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 45. 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.

Table 45. Key Generate2 valid key type and key form for one AES or HMAC key (continued)

key_type_1 (key usage)	Key Form OP, IM, EX	Notes
*DKYGENKY (D-MAC)	X	If you supply a skeleton token, the key usage for the derived key may not specify ONLY generate or ONLY verify.
*DKYGENKY (D-PCALC)	X	
HMAC MAC (GENERATE)	X	If you supply a skeleton token, the key usage may not specify ONLY generate or ONLY verify.
AES MAC (GENERATE)	X	The key usage in the skeleton may not specify ONLY generate or ONLY verify.

Table 46 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 46. 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 (DECRYPT)	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

Table 46. 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
*DKYGENKY	*DKYGENKY	X	X	X	X
EXPORTER	IMPORTER		X	X	X
IMPORTER	EXPORTER		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
MAC (VERIFY)	MAC (GENONLY)	X	X	X	X

See Table 50 on page 167 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 47 shows the valid key pairs. Access control points are required to be enabled for the generation of these keys.

Table 47. 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	D2

Table 48. 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			

Table 48. Key type and key form keywords for AES keys - DK PIN methods (continued)

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, 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 49. 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

Table 49. AES KEK strength required for generating an HMAC key under an AES KEK (continued)

Key-usage field 2 in the HMAC key contains	Minimum strength of AES KEK to adequately protect the HMAC key
SHA-1	128 bits

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 50. Required access control points for Key Generate2

Access Control Point	Function control
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 46 on page 164 and Table 48 on page 165.
Key Generate2 - Key set extended	The key-form and key-type combinations shown with an E in Table 48 on page 165.
Key Generate2 - DK PIN key set	The key-form and key-type combinations shown with an D in Table 48 on page 165.
Key Generate2 - DK PIN Admin1 Set PINPROT	The key-form and key-type combinations shown with an D1P in Table 48 on page 165.
Key Generate2 - DK PIN Admin1 Set MAC	The key-form and key-type combinations shown with an D1M in Table 48 on page 165.
Key Generate2 - DK PIN Print Set	The key-form and key-type combinations shown with an DP in Table 48 on page 165.
Key Generate2 - DK PIN Admin2 Set MAC	The key-form and key-type combinations shown with an D2 in Table 48 on page 165.
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.

Table 51. Key Generate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.

Table 51. Key Generate2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN support requires the November 2013 or later licensed internal code (LIC). V0PYLDK1, V1PYLDK1, V0PYLDK2, and V1PYLDK2 keywords require the November 2013 or later licensed internal code (LIC). AES key support requires the September 2011 or later licensed internal code (LIC). HMAC key support requires the November 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN support requires the September 2013 or later licensed internal code (LIC). V0PYLDK1, V1PYLDK1, V0PYLDK2, and V1PYLDK2 keywords require the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

Key Import (CSNBKIM and CSNEKIM)

Use the key import callable service to reencipher a 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 and reason codes," on page 943 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 and reason codes," on page 943 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, the token is all binary zeros, except for a key in bytes 16-23 or 16-31, or in bytes 16-31 and 48-55 for triple-length DATA keys. Refer to Table 392 on page 1000.

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 importer key-encrypting key that the key is currently encrypted under. The parameter is a 64-byte area containing either the key label of the key in the cryptographic key data set or the internal key token for the key. If you supply a key label that is less than 64-bytes, it must be left-justified and padded with blanks.

Note: If you specify a NOCV importer in the *importer_key_identifier* parameter, the key to be imported must be enciphered under the importer key itself.

target_key_identifier

Direction	Type
Input/Output	String

This parameter is the generated reenciphered key. The parameter is a 64-byte area that receives the internal key token for the imported key.

If the imported key TYPE is IMPORTER or EXPORTER and the token key TYPE is the same, the *target_key_identifier* parameter changes direction to both input and output. If 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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. 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.

Restrictions

For existing TKE users, you may have to explicitly enable new access control points. 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 a importer or exporter with the default control vector.
- If the target token is valid and the NOCV flag is on and the source token is valid 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 source token is valid and the control vector of the target token is NOT exactly the same as the source token, a 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.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 52. 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.

Table 53. Key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported. IMP-PKA keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported. IMP-PKA keys are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported. IMP-PKA keys are not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL and CIPHERXO are not supported. IMP-PKA keys are not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	IMP-PKA keys are not supported.
IBM z13	Crypto Express5 CCA Coprocessor	IMP-PKA keys are not supported.

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 or 2.

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 54. Keywords for Key Part Import Control Information

Keyword	Meaning
Key Part (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.
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 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*.
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.

key_part

Direction	Type
Input	String

A 16-byte field containing the clear key part to be entered. If the key is a single-length key, the key part must be left-justified and padded on the right with zeros. This field is ignored if 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 of a single- or double-length key 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.

ICSF supports two methods of wrapping the key value in a DES key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *key_identifier* will use the default method unless a rule array keyword overriding the default for the FIRST key part is specified. When the *key_identifier* is an existing token, the same wrapping method as the existing token will be used.

Restrictions

If a label is specified on *key_identifier*, the label must be unique. If more than one record is found, the service fails.

For existing TKE users, you may have to explicitly enable new access control points. 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.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 55. 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
WRAP-ECB or WRAP-ENH and default key-wrapping method setting does not match keyword	Key Part Import - Allow wrapping override keywords

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 documented above 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.

Table 56. Key part import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
	Crypto Express3 Coprocessor	Enhanced key token wrapping not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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 and reason codes," on page 943 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 and reason codes," on page 943 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 57. Keywords for Key Part Import2 Control Information

Keyword	Meaning
Token Algorithm (Required)	
HMAC	Specifies to import an HMAC key token.
AES	Specifies to import an AES 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 80 to 2048 for HMAC keys or 128, 192, or 256 for AES keys. 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 725 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 token or a 64-byte label of an existing CKDS record. If the Key Part rule is FIRST, the key is a skeleton token. If the Key Part rule is ADD-PART, 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.

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.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 58. Required access control points for Key Part Import2

Rule array keywords	Access control point
ADD-PART	Key Part Import2 - Add second of three 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

Table 58. Required access control points for Key Part Import2 (continued)

Rule array keywords	Access control point
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

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 59. Key Part Import2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN support requires the November 2013 or later licensed internal code (LIC). AES key support requires the September 2011 or later licensed internal code (LIC). HMAC key support requires the November 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

Key Test (CSNBKYT and CSNEKYT)

Use the key test callable service to generate or verify a secure, cryptographic verification pattern for DES and AES clear keys or encrypted DES and AES keys in an internal, fixed-length key token. 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- and double-length keys)" on page 1117 as the default algorithm (except for triple-length DATA 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. The random number parameter is not used.

CSNBKYT is consistent with the IBM 4765 PCIe Cryptographic Coprocessor verb of the same name. If you generate a key on the IBM 4765 PCIe Cryptographic Coprocessor, you can verify it on ICSF and vice versa.

See "Key Test Extended (CSNBKYTX and CSNEKTX)" on page 190 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 and reason codes," on page 943 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 and reason codes," on page 943 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 60 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 60. 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-ENC	Specifies the DES key supplied in *key_identifier* is a single-length encrypted key in a fixed-length key token.
KEY-ENCD	Specifies the DES key supplied in *key_identifier* is a double-length encrypted key in a fixed-length key token.
TOKEN	Process an AES clear or encrypted key token.
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.

Table 60. Keywords for Key Test Control Information (continued)

Keyword	Meaning
Verification Process Rule (optional)	
ENC-ZERO	ENC-ZERO can be used with any of the rules.
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

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.

There is support for the generation and verification of single, double and triple-length keys for the ENC-ZERO verification process. For triple-length keys, use KEY-ENC or KEY-ENCD with ENC-ZERO. Clear triple-length keys are not supported.

| For the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors, KEY-ENC and KEY-ENCD both support enciphered single-length and double-length keys. They use the key-form bits in byte 5 of CV to determine the length of the key. To be consistent, in ICSF, both KEY-ENC and KEY-ENCD handle single- and double-length keys. Both products effectively ignore the keywords, which are supplied only for compatibility reasons.

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 - Warn when keyword inconsistent with 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.

Table 61. Key test required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only. AES keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only. Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only. Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.

Table 61. Key test required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.
IBM z13	Crypto Express5 CCA Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.

Key Test2 (CSNBKYT2 and CSNEKYT2)

Use this callable service to generate or verify a secure, cryptographic verification pattern 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, key tokens may be either internal, fixed-length (version 04) tokens, or external and internal variable-length (version 05) tokens.

For DES, key tokens may be external and internal, fixed-length (versions 00 or 01) tokens, external TR-31 key blocks, or external variable-length tokens with a DESUSECV key.

For HMAC, key tokens are external and internal variable-length (version 05) key tokens.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 62. Keywords for Key Test2 Control Information

Keyword	Meaning
Token algorithm (Required)	
AES	Specifies the key token is an AES key token.
DES	Specifies the key token is a DES token. CCA internal, CCA external, and TR-31 token types are supported. Clear keys are not supported for this rule.

Table 62. Keywords for Key Test2 Control Information (continued)

Keyword	Meaning
HMAC	Specifies the key token is an HMAC key token.
Process rule (One required)	
GENERATE	Generate a verification pattern for the specified key.
VERIFY	Verify that a verification pattern matches the specified key.
Verification pattern calculation algorithm (Optional)	
ENC-ZERO	Verification pattern for AES and DES keys calculated by encrypting a data block filled with 0x00 bytes. This is the default and only method available for DES. This method is only available for AES if Access Control Point "Key Test2 - AES, ENC-ZERO" is enabled.
SHA-256	Verification pattern will be calculated for an AES token using the same method as the Key Test service with the SHA-256 rule. 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 pairs.
SHA2VP1	Specifies to use the SHA-256 based verification pattern calculation algorithm. For more information, see "SHAVP1 Algorithm" on page 1117. This is the default and only method available for HMAC.
Token type rule (One required if the DES token algorithm is specified and a TR-31 token or DESUSECV token is passed; not valid otherwise)	
TR-31	Specifies that *key_identifier* contains a TR-31 key block.
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.

key_identifier_length

Direction	Type
Input	Integer

The length of the key_identifier in bytes. The maximum value is 9992.

key_identifier

Direction	Type
Input/Output	String

The key for which to generate or verify the verification pattern. This is an internal or external token or the 64-byte label of a key in the CKDS. This token may be a DES internal or external token, AES internal version ‘04’X token, internal or external variable-length symmetric token, or a TR-31 key block.

Clear DES tokens are not supported.

If an internal token was supplied and was encrypted under the old master key, the token 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. If *key_encrypting_key_identifier* is an RSA KEK, the maximum length is 3500.

key_encrypting_key_identifier

Direction	Type
Input/Output	String

When *key_encrypting_key_identifier_length* is non-zero, *key_encrypting_key_identifier* contains an internal key token containing the key-encrypting key, or a key label.

If the key identifier supplied was an AES or DES token encrypted under the old master key, the token 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.

On input: For GENERATE, the length must be at least 8 bytes; For VERIFY, the length must be 8 bytes.

On output for GENERATE, 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.

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.

| The **Key Test2** - AES, ENC-ZERO access control point must be enabled to use the
| ENC-ZERO keyword for AES keys.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 63. Key Test2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.

Table 63. Key Test2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN support requires the November 2013 or later licensed internal code (LIC). DES/AES key support requires the September 2011 or later licensed internal code (LIC). HMAC key support requires the November 2010 or later licensed internal code (LIC). The AESKWCV keyword is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN support requires the September 2013 or later licensed internal code (LIC). The AESKWCV keyword requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

Key Test Extended (CSNBKYTX and CSNEKTX)

Use the key test extended service to generate or verify a secure, cryptographic verification pattern for DES keys. The key to test can be in the clear or encrypted under the master key. The callable service also supports keys 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.

This algorithm is supported for encrypted single and double length keys. Single, double and triple length keys are also 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 CSNEKTX.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 64 on page 192 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 64. 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.
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)	
ENC-ZERO	Specifies use of the "encrypted zeros" method.

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

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 and double-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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 65. Key test extended required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.

Table 65. Key test extended required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.
IBM z13	Crypto Express5 CCA Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.

Key Token Build (CSNBKTB and CSNEKTB)

Use the key token build callable service to build an external or internal key token from information which you supply. 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 and reason codes," on page 943 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 and reason codes," on page 943 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

This field will contain the key token built.

key_type

Direction	Type
Input	String

An 8-byte field that specifies the type of key you want to build. The key types are:

Table 66. Key type keywords for key token build

Key type	Description	Algorithm
CIPHER CIPHERXI CIPHERXL CIPHERXO	See "Other considerations" on page 23.	DES
CLRAES	The *key_token* parameter is a clear AES 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.	AES
CLRDES	The *key_token* parameter is a clear DES 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.	DES
CVARDEC CVARENC CVARPINE CVARXCVL CVARXCVR	See "Other considerations" on page 23.	DES
DATA	Valid for AES and DES keys and must be specified with the *rule_array* keyword AES to build an encrypted AES key token.	AES and DES
DATAC DATAM DATAMV DECIPHER DKYGENKY ENCIPHER	See "Other considerations" on page 23.	DES
EXPORTER	If the *key_type* parameter is TOKEN, then this is a 64-byte internal token that is updated as specified in the *rule_array*.	DES
IKEYXLAT	See "Other considerations" on page 23.	DES
IMPORTER	If the *key_type* parameter is TOKEN, then this is a 64-byte internal token that is updated as specified in the *rule_array*.	DES
KEYGENKY	CLR8-ENC or UKPT must be coded in *rule_array* parameter	DES
IPINENC MAC MACVER OKEYXLAT OPINENC PINGEN PINVER	See "Other considerations" on page 23.	DES
SECMSG	SMKEY or SMPIN must be specified in the *rule_array* parameter.	DES
USE-CV	A user-supplied control vector is specified. The key type should be obtained from the control vector specified in the control_vector parameter. The CV rule array keyword should be specified if USE-CV is specified.	DES

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 67 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. For any key type, there are no more than four valid *rule_array* values.

Table 67. 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 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)		
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 or 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. **Note:** See Table 69 on page 201 for valid key types for these key length values.	DES

Table 67. Keywords for Key Token Build Control Information (continued)

Keyword	Meaning	Algorithm
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
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
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.	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 69 on page 201 for the key-usage keywords that can be specified for a given key type.		DES
Master Key Verification Pattern (optional) — not valid for CLRDES or CLRAES keywords		
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 key contained in the *key_value* field must be enciphered under the current master key.	AES and DES
Key Wrapping Method (optional)		
WRAP-ENH	Use enhanced key wrapping method, 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

Table 67. Keywords for Key Token Build Control Information (continued)

Keyword	Meaning	Algorithm
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. Can only be specified with WRAP-ENH.	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 DATA key, this parameter is a 24-byte string containing the encrypted key value. 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 68. 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

A pointer to a 16 byte string variable. When the CV rule array keyword is used, this parameter must point to a control vector which is copied into the key token. 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).

NOCV keyword is only supported for the standard IMPORTERs and EXPORTERs with the default CVs.

This 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.

Table 69. Control Vector Generate and Key Token Build Control Vector Keyword Combinations

Key Type	Key Usage				
DATA		**SINGLE** KEYLN8 MIXED DOUBLE DOUBLE-O KEYLN16 KEYLN24	**XPORT-OK** NO-XPORT	KEY-PART	
CIPHER ENCIPHER DECIPHER		**SINGLE** KEYLN8 MIXED DOUBLE DOUBLE-O KEYLN16	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
CIPHERXI CIPHERXL CIPHERXO		**DOUBLE** DOUBLE-O KEYLN16	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
MAC MACVER	ANY-MAC ANSIX9.9 CVVKEY-A CVVKEY-B AMEX-CSC	SINGLE KEYLN8 MIXED DOUBLE DOUBLE-O KEYLN16	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT

Table 69. Control Vector Generate and Key Token Build Control Vector Keyword Combinations (continued)

Key Type	Key Usage							
CVARPINE CVARENC CVARDEC CVARXCVL CVARXCVR					**SINGLE** KEYLN8	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
DATAC DATAM DATAMV					**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
KEYGENKY	CLR8-ENC UKPT				**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
DKYGENKY	DDATA DMAC DMV DIMP DEXP DPVR DMKEY DMPIN DALL	DKYL0 DKYL1 DKYL2 DKYL3 DKYL4 DKYL5 DKYL6 DKYL7			**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
SECMSG	SMKEY SMPIN				**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
IKEYXLAT OKEYXLAT				**ANY** NOT-KEK DATA PIN LMTD-KEK	**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
IMPORTER	OPIM* IMEX* IMIM* IMPORT*	XLATE		**ANY** NOT-KEK DATA PIN LMTD-KEK	**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
EXPORTER	OPEX* IMEX* EXEX* EXPORT*	XLATE		**ANY** NOT-KEK DATA PIN LMTD-KEK	**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
PINVER			**NO-SPEC**** IBM-PIN** GBP-PIN** IBM-PINO GBP-PINO VISA-PVV INBK-PIN	NOOFFSET	**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT

Table 69. Control Vector Generate and Key Token Build Control Vector Keyword Combinations (continued)

Key Type	Key Usage						
PINGEN	CPINGEN* CPINGENA* EPINGENA* EPINGEN* EPINVER*	**NO-SPEC**** IBM-PIN** GBP-PIN** IBM-PINO GBP-PINO VISA-PVV INBK-PIN	NOOFFSET	**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
IPINENC	CPINGENA* EPINVER* REFORMAT* TRANSLAT*			**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
OPINENC	CPINENC* EPINGEN* REFORMAT* TRANSLAT*			**DOUBLE** DOUBLE-O KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART ENH-ONLY	**T31XPTOK** NOT31XPT
Notes:	Default keys are indicated in bold. * All keywords in the list are defaults unless one or more keywords in the list are specified ** The NOOFFSET keyword is only valid if NO-SPEC, IBM-PIN, GBP-PIN, or the default (NO-SPEC) is specified. A key usage keyword is required for the KEYGENKY and SECMSG key types. • CLR8-ENC and/or UKPT must be specified for the KEYGENKY key type • SMKEY or SMPIN must be specified for the SECMSG key type						

Required hardware

No cryptographic hardware is required by this callable service.

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 70. 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, and DKYGENKY 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 8 on page 209 and Table 72 on page 227 for key-usage and key-management keywords.
DKYGENKY	Specifies that this key is for key-generation. Only valid for AES algorithm. See Figure 13 on page 219 and Table 72 on page 227 for key-usage and key-management keywords.
EXPORTER	Specifies that this key is an EXPORTER key-encrypting key. Only valid for AES algorithm. See Figure 11 on page 215 and Table 72 on page 227 for key-usage and key-management keywords.
IMPORTER	Specifies that this key is an IMPORTER key-encrypting key. Only valid for AES algorithm. See Figure 12 on page 217 and Table 72 on page 227 for key-usage and key-management keywords.
MAC	Specifies that this key is for message authentication code operations. Valid for HMAC and AES algorithms. See Figure 9 on page 211, Figure 10 on page 213, and Table 72 on page 227 for key-usage and key-management keywords.
PINCALC	Specifies that this key is for calculating PINs. Only valid for AES algorithm. See Figure 14 on page 222 and Table 72 on page 227 for key-usage and key-management keywords.

Table 70. Keywords for Key Token Build2 Control Information (continued)

Keyword	Meaning
PINPROT	Specifies that this key is for wrapping and unwrapping PIN blocks. Only valid for AES algorithm. See Figure 15 on page 224 and Table 72 on page 227 for key-usage and key-management keywords.
PINPRW	Specifies that this key is for generating and verifying PIN reference values. Only valid for AES algorithm. See Figure 16 on page 226 and Table 72 on page 227 for key-usage and key-management keywords.

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, CIPHER/EXPORTER/IMPORTER key types, 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. For rule array keyword DKYUSAGE, the value must be a multiple of 8. Otherwise, the value must be 0. The maximum value is 280.

service_data

Direction	Type
Input	String

Data to be processed by this service when building the skeleton token. If the DKYUSAGE keyword is specified in the rule array, this parameter contains an array of key usage keywords for the type of key to be derived. The keywords are 8 bytes in length and must be left-aligned and padded on the right with blanks.

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 8 on page 209 shows all the valid keyword combinations and their defaults for AES key type CIPHER. For a description of these keywords, see Table 72 on page 227.

```
| Key token hdr | WI section       | Associated data section                                        |
|---------------|------------------|----------------------------------------------------------------|
| Token         | Key mtrl state   | Key type & type  |                    | Key management        |
| identifier    | & pyld fmt ver   | of algorithm     | Key usage fields   | fields                |

  Note: One        Note:                                                 Note:
  required.        NO-KEY                              See Note 2        XPRT-SYM
                   is default.                                           is default.
>>─┬─EXTERNAL─┬─┬─KEY-CLR─┬─CIPHER─AES─┬─V─C-XLATE─┬─┬─NOEX-SYM─┬─
   └─INTERNAL─┘ └─NO-KEY──┘            │   DECRYPT─┤ └─XPRT-SYM─┘
                                       │   ENCRYPT─┘
                   See Note 1          │                          Note:
                ┬─V0PYLD─┬             │ See Note 3               XPRTUASY
                └─V1PYLD─┘             │                          is default.
                                       ├─V─UDX-ONLY─┐           ┬─NOEXUASY─┬
                                       │   UDX-001──┤           └─XPRTUASY─┘
                                       │   UDX-010──┤
                                       │   UDX-100──┘             Note:
                                       │                          XPRTAASY
                                       │ Note: CBC                is default.
                                       │ is default.            ┬─NOEXAASY─┬
                                       ├─CBC─┐                  └─XPRTAASY─┘
                                       │ CFB─┤
                                       │ ECB─┤                    Note:
                                       │ GCM─┤                    XPRTDES
                                       │ OFB─┤                    is default.
                                       └─XTS─┘                  ┬─NOEX-DES─┬
                                                                └─XPRT-DES─┘

                                                                  Note:
                                                                  XPRTAES
                                                                  is default.
                                                                ┬─NOEX-AES─┬
                                                                └─XPRT-AES─┘

                                                                  Note:
                                                                  XPRT-RSA
                                                                  is default.
                                                                ┬─NOEX-RSA─┬
                                                                └─XPRT-RSA─┘

                                                                  See Note 4
                                                                ┬─NOEX-RAW─┬
                                                                └─XPRT-RAW─┴──>>
```

Figure 8. Key Token Build2 keyword combinations for AES CIPHER keys

Note:

1. Keyword V0PYLD is the default for compatibility reasons. V1PYLD is recommended.
2. Keywords DECRYPT and ENCRYPT are defaults unless one or more keywords in the group are specified.

3. Choose any number of keywords in this group. No keywords in the group are defaults.
4. NOEX-RAW is default. These keywords are for future use and their meanings are currently undefined.

Figure 9 on page 211 shows all the valid keyword combinations and their defaults for AES key type MAC. For a description of these keywords, see Table 72 on page 227.

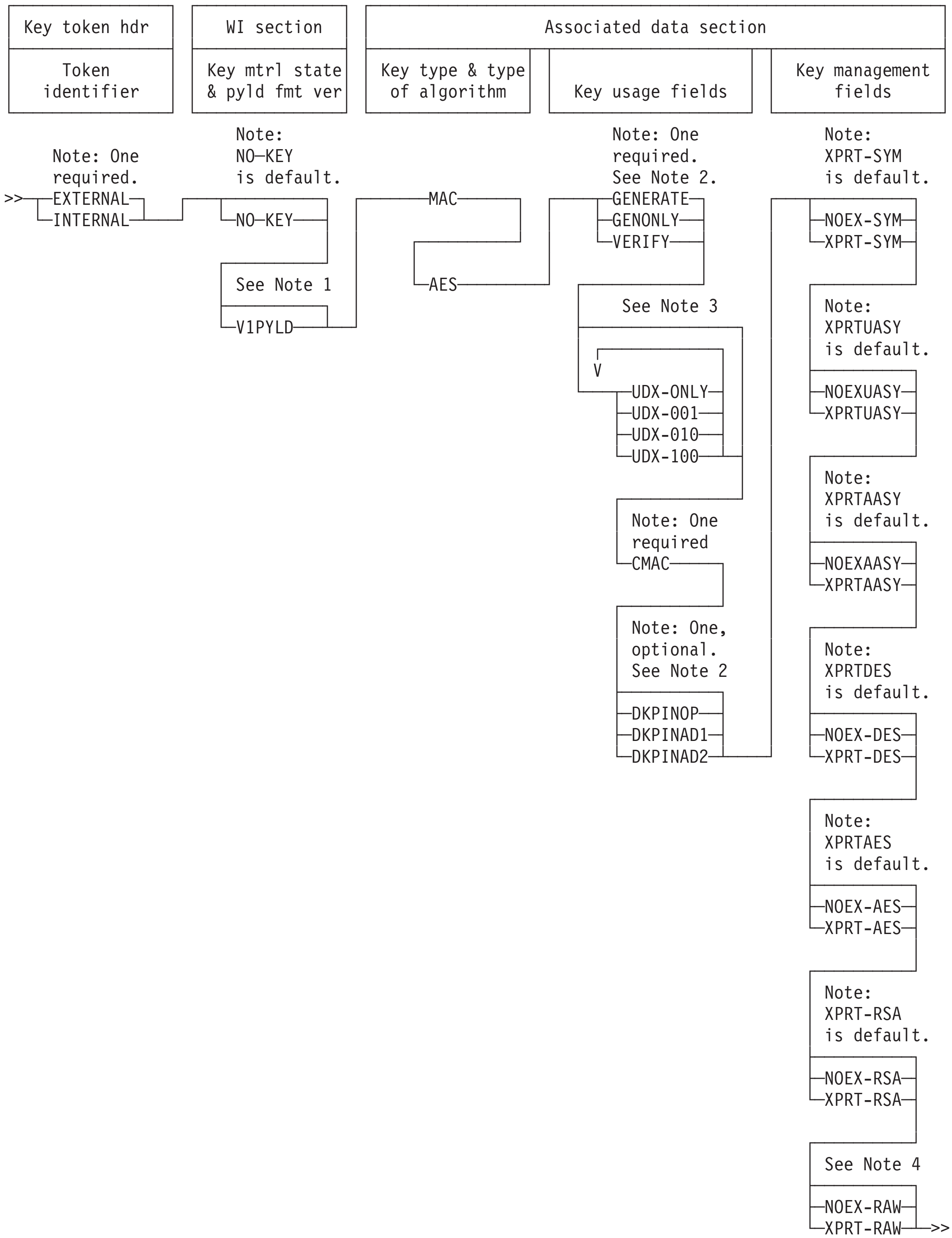

Figure 9. Key Token Build2 keyword combinations for AES MAC keys

Note:

1. Keyword V1PYLD is the default. V1PYLD is the only payload format version allowed for this key type.
2. Keyword GENERATE is not valid with keywords DKPINOP, DKPINAD1, and DKPINAD2.

3. Choose any number of keywords in this group. No keywords in the group are defaults.
4. NOEX-RAW is default. These keywords are for future use and their meanings are currently undefined.

Figure 10 on page 213 shows all the valid keyword combinations and their defaults for HMAC key type MAC. For a description of these keywords, see Table 72 on page 227.

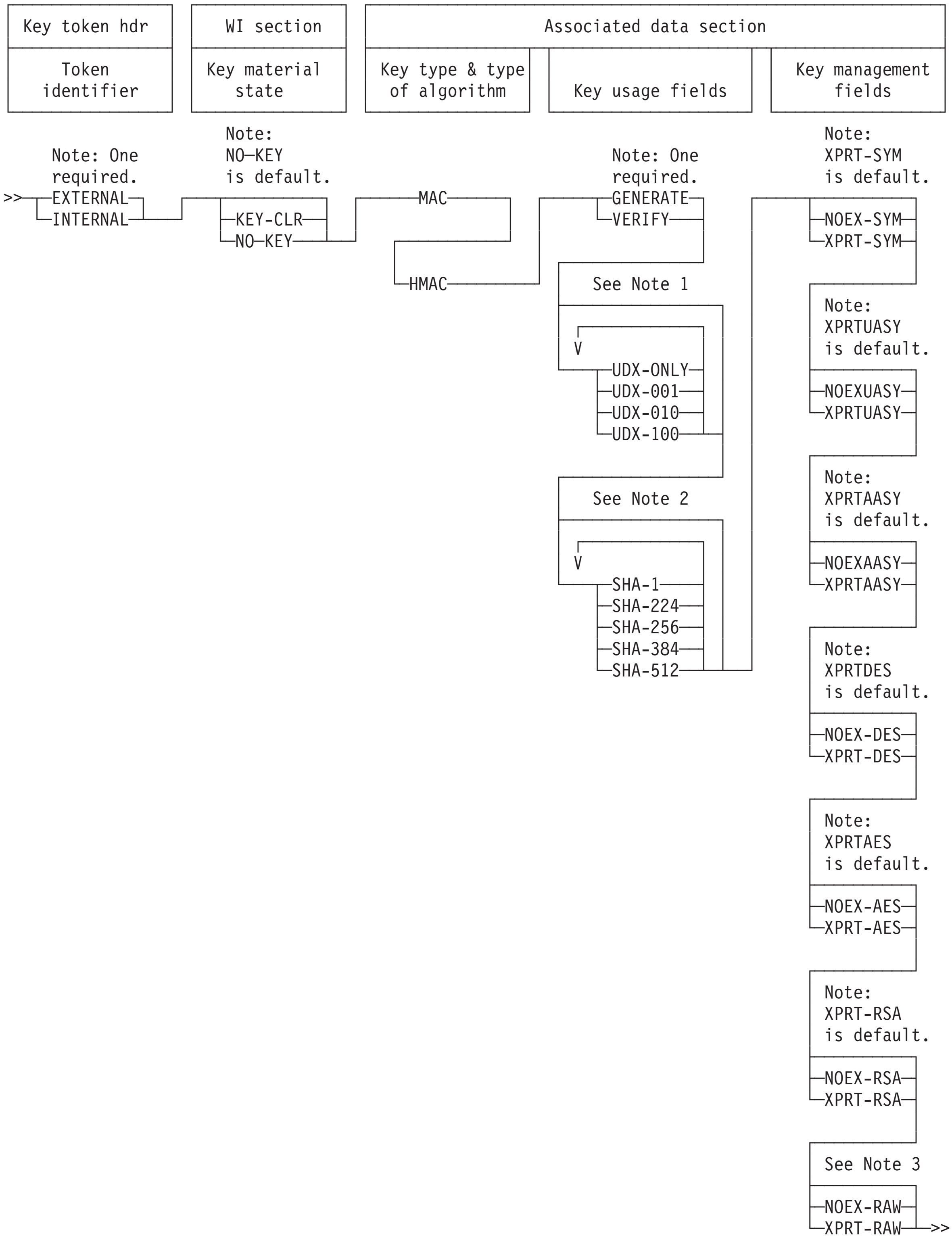

Figure 10. Key_Token_Build2 keyword combinations for HMAC MAC keys

Note:

1. Choose any number of keywords in this group. No keywords in the group are defaults.
2. All keywords in the group are defaults unless one or more keywords in the group are specified.

3. NOEX-RAW is default. These keywords are for future use and their meanings are currently undefined.

Figure 11 on page 215 shows all the valid keyword combinations and their defaults for AES key type EXPORTER. For a description of these keywords, see Table 72 on page 227.

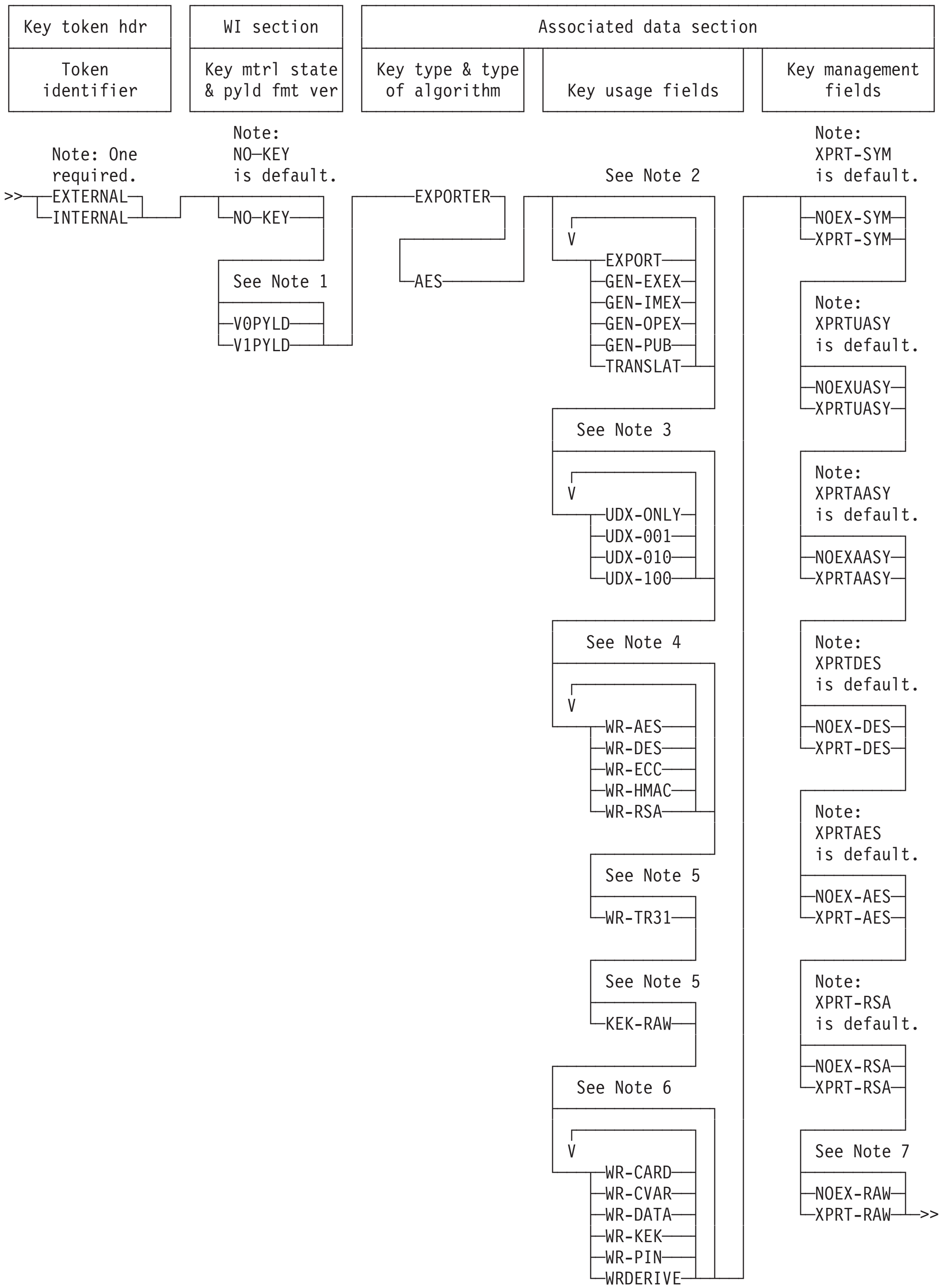

Figure 11. Key Token Build2 keyword combinations for AES EXPORTER keys

Note:

1. Keyword V0PYLD is the default for compatibility reasons. V1PYLD is recommended.
2. All keywords in the group are defaults unless one or more keywords in the group are specified.
3. Choose any number of keywords in this group. No keywords in the group are defaults.
4. Keywords WR-AES, WR-DES, and WR-HMAC are defaults unless one or more keywords in the group are specified.
5. This keyword is for future use and its meaning is currently undefined.
6. 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. 2.
7. NOEX-RAW is default. These keywords are for future use and their meanings are currently undefined.

Figure 12 on page 217 shows all the valid keyword combinations and their defaults for AES key type IMPORTER. For a description of these keywords, see Table 72 on page 227.

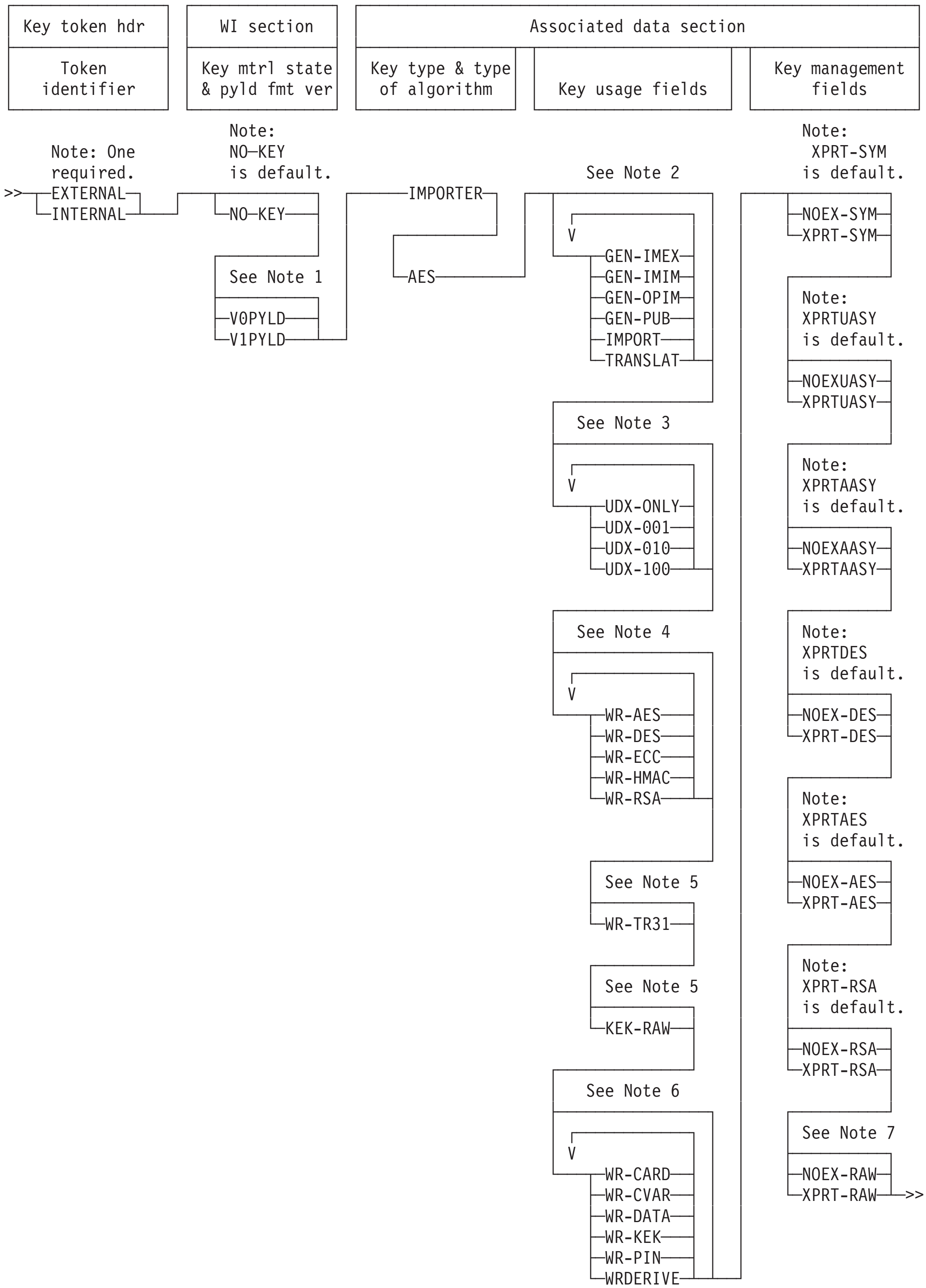

Figure 12. Key Token Build2 keyword combinations for AES IMPORTER keys

Note:

1. Keyword V0PYLD is the default for compatibility reasons. V1PYLD is recommended.
2. All keywords in the group are defaults unless one or more keywords in the group are specified.
3. Choose any number of keywords in this group. No keywords in the group are defaults. 4.
4. Keywords WR-AES, WR-DES, and WR-HMAC are defaults unless one or more keywords in the group are specified.
5. This keyword is for future use and its meaning is currently undefined.
6. 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.
7. NOEX-RAW is default. These keywords are for future use and their meanings are currently undefined.

Figure 13 on page 219 shows all the valid keyword combinations and their defaults for AES key type DKYGENKY. For a description of these keywords, see Table 72 on page 227.

```
┌────────────┐ ┌────────────┐ ┌────────────────────────────────────────────────┐
│Key token hdr│ │ WI section │ │             Associated data section            │
├────────────┤ ├────────────┤ ├────────────┬──────────────┬───────────────────┤
│   Token    │ │Key mtrl state│ │Key type & type│              │  Key management   │
│ identifier │ │& pyld fmt ver│ │of algorithm│ Key usage fields │      fields       │
└────────────┘ └────────────┘ └────────────┴──────────────┴───────────────────┘

                Note:                                          Note:
   Note: One    NO─KEY                     Note: One           XPRT-SYM
   required.    is default.                required.           is default.
>>─┬─EXTERNAL─┐ ┌─┬───────────┐ ┌──────DKYGENKY─┐ ┌─┬─D-ALL─────┐ ┌─┬───────────┐
   └─INTERNAL─┴─┘ └─NO─KEY────┤ │ ┌─────────────┘ │ ├─D-CIPHER──┤ │ ├─NOEX-SYM─┤
                              │ │ │               │ ├─D-EXP─────┤ │ └─XPRT-SYM─┤
                  ┌───────────┘ │ │               │ ├─D-IMP─────┤ │            │
                  │ See Note 1  │ └─AES───────────┘ ├─D-MAC─────┤ │ ┌──────────┘
                  │             │                   ├─D-PCALC───┤ │ │ Note:
                  └─V1PYLD──────┘                   ├─D-PPROT───┤ │ │ XPRTUASY
                                                    └─D-PPRW────┤ │ │ is default.
                                                                │ │ │
                                                  ┌─────────────┘ │ ├─NOEXUASY─┐
                                                  │ See Note 2    │ └─XPRTUASY─┤
                                                  │ ┌───────────┐ │            │
                                                  │ V           │ │ ┌──────────┘
                                                  └─┬───────────┘ │ │ Note:
                                                    ├─UDX-ONLY──┤ │ │ XPRTAASY
                                                    ├─UDX-001───┤ │ │ is default.
                                                    ├─UDX-010───┤ │ │
                                                    └─UDX-100───┴─┤ ├─NOEXAASY─┐
                                                                  │ └─XPRTAASY─┤
                                                                  │            │
                                                                  │ ┌──────────┘
                                                                  │ │ Note:
                                                     See Note 3   │ │ XPRTDES
                                                    ┌─────────────┘ │ is default.
                                                    ├─KUF-MBE───┐   │
                                                    └─KUF-MBP───┤   ├─NOEX-DES─┐
                                                                │   └─XPRT-DES─┤
                                                    ┌───────────┘              │
                                                    │                ┌─────────┘
                                                    └─DKYL0───────┐  │ Note:
                                                                  │  │ XPRTAES
                                                                  │  │ is default.
                                                    ┌─────────────┘  │
                                                    │ See Note 4     ├─NOEX-AES─┐
                                                    └─DKYUSAGE────┘  └─XPRT-AES─┤
                                                                                │
                                                                     ┌──────────┘
                                                                     │ Note:
                                                                     │ XPRT-RSA
                                                                     │ is default.
                                                                     │
                                                                     ├─NOEX-RSA─┐
                                                                     └─XPRT-RSA─┤
                                                                                │
                                                                     ┌──────────┘
                                                                     │ See Note 5
                                                                     │
                                                                     ├─NOEX-RAW─┐
                                                                     └─XPRT-RAW─┴──>>
```

Figure 13. Key Token Build2 keyword combinations for AES DKYGENKY keys

Note:

1. Keyword V1PYLD is the default. V1PYLD is the only payload format version allowed for this key type.
2. Choose any number of keywords in this group.

3. If keyword D-ALL is specified, keywords KUF-MBE and KUF-MBP are not valid and there is no default; otherwise the default is KUF-MBE. If keyword DKPINOP, DKPINOPP, DKPINAD1, or DKPINAD2 is specified in the *service_data* parameter (in other words, the type of key to diversify is DK enabled), keyword KUF-MBP is not valid.
4. There is no default. DKYUSAGE specifies that the *service_data* parameter contains key-usage field keywords related to the type of key to diversify. These related attributes become part of the key usage fields of the DKYGENKY diversifying key. They are related because they are used to control which key usage attributes are permissible in the generated diversified key. To generate a diversified key, use the Diversified_Key_Generate2 verb. The type of key to diversify affects the use of the DKYUSAGE keyword:
   - For D-ALL, DKYUSAGE is not allowed.
   - For D-CIPHER, D-EXP, and D-IMP, DKYUSAGE is optional. If not specified, default key usage fields of the type of key to diversify are used.
   - For D-MAC, D-PCALC, D-PPROT, and D-PPRW, DKYUSAGE is required.

   See Table 71 for additional information on the meaning of the verb_data variable when DKYUSAGE is specified.
5. NOEX-RAW is default. These keywords are for future use and their meanings are currently undefined.

Table 71. 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-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.

Table 71. Meaning of service_data parameter when DKYUSAGE specified (continued)

Type of key to diversify	DKYUSAGE keyword	Description of verb_data variable when DKYUSAGE specified
D-PPRW	Required	The service_data parameter must contain key usage fields keywords related to an AES PINPRW key.

Figure 14 on page 222 shows all the valid keyword combinations and their defaults for AES key type PINCALC. For a description of these keywords, see Table 72 on page 227.

```
Key token hdr      | WI section        | Associated data section
Token identifier   | Key mtrl state    | Key type & type  | Key usage fields | Key management
                   | & pyld fmt ver    | of algorithm     |                  | fields

Note: One          Note:                                 Note: One          Note:
required.          NO-KEY                                required.          XPRT-SYM
                   is default.                                              is default.
>>─┬─EXTERNAL─┬─   ──NO-KEY──          ──PINCALC──       ──GENONLY──        ┬─NOEX-SYM─┬
   └─INTERNAL─┘                                                             └─XPRT-SYM─┘
                   See Note 1          ──AES──           See Note 2
                   ──V1PYLD──                                               Note:
                                                         V                  XPRTUASY
                                                         ┬─UDX-ONLY─┬       is default.
                                                         ├─UDX-001──┤       ┬─NOEXUASY─┬
                                                         ├─UDX-010──┤       └─XPRTUASY─┘
                                                         └─UDX-100──┘
                                                                            Note:
                                                         Note: One          XPRTAASY
                                                         required           is default.
                                                         ──CBC──            ┬─NOEXAASY─┬
                                                                            └─XPRTAASY─┘
                                                         Note: One
                                                         required.          Note:
                                                         ──DKPINOP──        XPRTDES
                                                                            is default.
                                                                            ┬─NOEX-DES─┬
                                                                            └─XPRT-DES─┘

                                                                            Note:
                                                                            XPRTAES
                                                                            is default.
                                                                            ┬─NOEX-AES─┬
                                                                            └─XPRT-AES─┘

                                                                            Note:
                                                                            XPRT-RSA
                                                                            is default.
                                                                            ┬─NOEX-RSA─┬
                                                                            └─XPRT-RSA─┘

                                                                            See Note 3
                                                                            ┬─NOEX-RAW─┬
                                                                            └─XPRT-RAW─┴──>>
```

Figure 14. Key Token Build2 keyword combinations for AES PINCALC keys

Note:

1. Keyword V1PYLD is the default. V1PYLD is the only payload format version allowed for this key type.
2. Choose any number of keywords in this group. No keywords in the group are defaults.

3. NOEX-RAW is default. These keywords are for future use and their meanings are currently undefined.

Figure 15 on page 224 shows all the valid keyword combinations and their defaults for AES key type PINPROT. For a description of these keywords, see Table 72 on page 227.

```
Key token hdr      | WI section        | Associated data section
Token identifier   | Key mtrl state    | Key type & type   | Key usage fields | Key management
                   | & pyld fmt ver    | of algorithm      |                  | fields

   Note: One         Note:                                  Note: One           Note:
   required.         NO—KEY                                 required.           XPRT-SYM
                     is default.                                                is default.
>>─┬─EXTERNAL─┬──┬─────────────┬─────PINPROT──┐   ┌──┬─DECRYPT─┬──┐          ┌─┬─NOEX-SYM─┬─┐
   └─INTERNAL─┘  │             └─NO─KEY──┘    │   │  └─ENCRYPT─┘  │          │ └─XPRT-SYM─┘ │
                 │  See Note 1                └─AES               │          │  Note:
                 │  └─V1PYLD───┘                  See Note 2      │          │  XPRTUASY
                                                  V                          │  is default.
                                                  ├─UDX-ONLY─┤               ├─NOEXUASY─┤
                                                  ├─UDX-001──┤               └─XPRTUASY─┘
                                                  ├─UDX-010──┤                  Note:
                                                  └─UDX-100──┘                  XPRTAASY
                                                    Note: One                   is default.
                                                    required                 ├─NOEXAASY─┤
                                                    └─CBC──┘                 └─XPRTAASY─┘
                                                    Note: One                   Note:
                                                    required.                   XPRTDES
                                                  ├─DKPINOP──┤                  is default.
                                                  ├─DKPINOPP─┤               ├─NOEX-DES─┤
                                                  └─DKPINAD1─┘───────────┘   └─XPRT-DES─┘
                                                                                Note:
                                                                                XPRTAES
                                                                                is default.
                                                                             ├─NOEX-AES─┤
                                                                             └─XPRT-AES─┘
                                                                                Note:
                                                                                XPRT-RSA
                                                                                is default.
                                                                             ├─NOEX-RSA─┤
                                                                             └─XPRT-RSA─┘
                                                                                See Note 3
                                                                             ├─NOEX-RAW─┤
                                                                             └─XPRT-RAW─┴──>>
```

Figure 15. Key Token Build2 keyword combinations for AES PINPROT keys

Note:

1. Keyword V1PYLD is the default. V1PYLD is the only payload format version allowed for this key type.
2. Choose any number of keywords in this group. No keywords in the group are defaults.

3. NOEX-RAW is default. These keywords are for future use and their meanings are currently undefined.

Figure 16 on page 226 shows all the valid keyword combinations and their defaults for AES key type PINPRW. For a description of these keywords, see Table 72 on page 227.

Key Token Build2

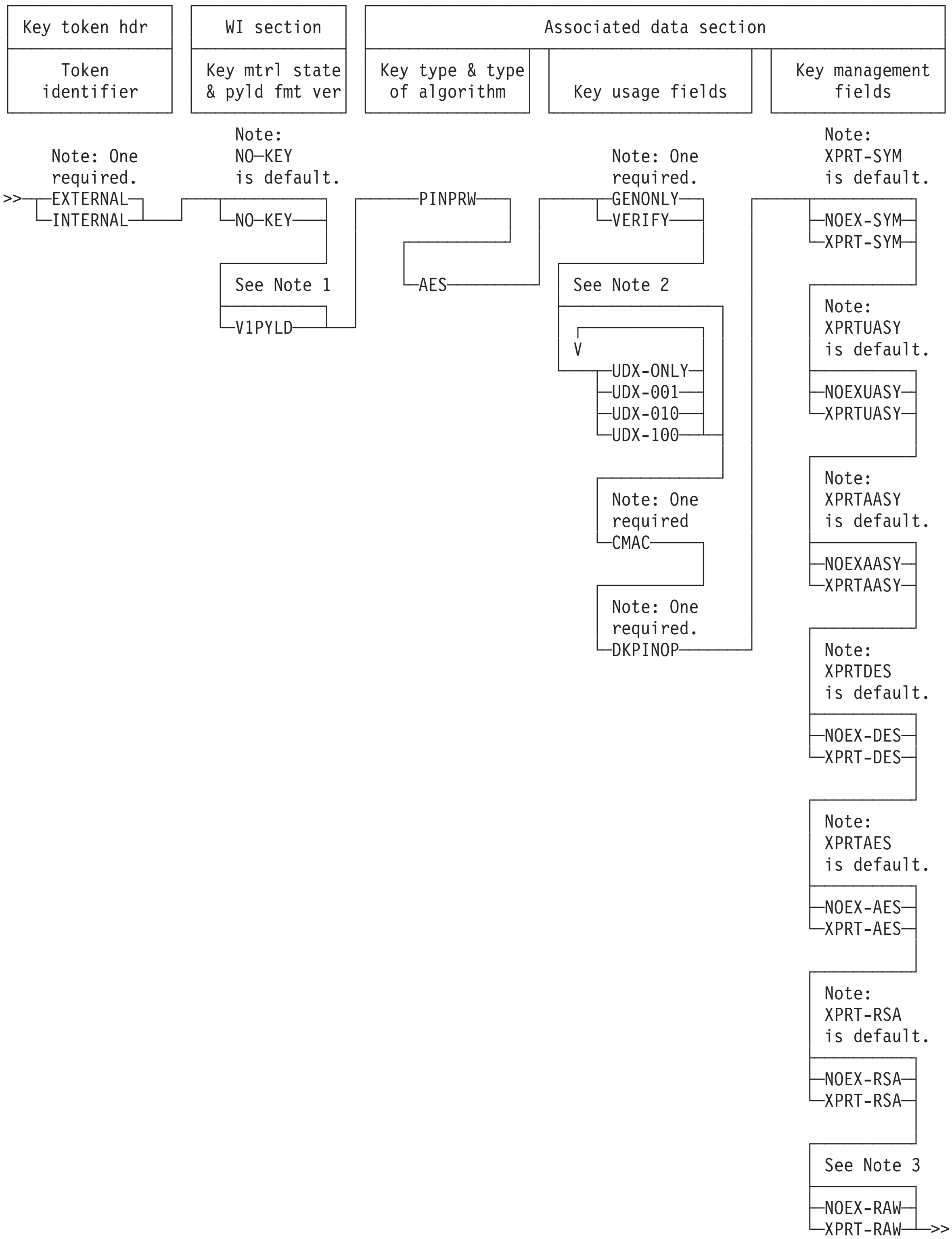

Figure 16. Key Token Build2 keyword combinations for AES PINPRW keys

Note:

1. Keyword V1PYLD is the default. V1PYLD is the only payload format version allowed for this key type.
2. Choose any number of keywords in this group. No keywords in the group are defaults.

3. NOEX-RAW is default. These keywords are for future use and their meanings are currently undefined.

Table 72. Key Token Build2 key-usage keywords

Keyword	Meaning
Key-usage field 1, high-order byte	
Generate control (one required). ***For MAC key type. For AES PINCALC and PINPRW key types.***	
GENERATE	Specifies that this key can be used to generate a MAC. A key that can generate a MAC can also verify a MAC.
GENONLY	Specifies that this key can only be used to generate a MAC. It cannot be used to verify a MAC or PRW. Valid only for AES MAC, PINCALC, and PINPRW key types.
VERIFY	Specifies that this key cannot be used to generate a MAC. It can only be used to verify a MAC.
Encrypt control. ***For AES algorithm, CIPHER key types (optional, any combination).*** **Note:** All keywords in the list below are defaults unless one or more keywords in the list are specified. ***For AES algorithm, PINPROT key types (one, required).***	
ENCRYPT	Specifies that this key can be used to encipher data.
DECRYPT	Specifies that this key can be used to decipher data.
Ciphertext Translate Control (optional). ***For AES algorithm, CIPHER key type.***	
C-XLATE	Specifies that this key can only be used for cipher text translation.
Exporter control (any combination, optional). ***For AES algorithm, EXPORTER key type.*** **Note:** All keywords in the list below are defaults 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.
Importer control (any combination, optional). ***For AES algorithm, IMPORTER key type.*** **Note:** All keywords in the list below are defaults 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.
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.
Key derivation control (one, required). ***For AES algorithm, DKYGENKY key type.***	
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.

Table 72. Key Token Build2 key-usage keywords (continued)

Keyword	Meaning
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
Key-usage field 1, low-order byte	
User-defined extension control (any combination, optional). ***For all algorithms and key types.*** **Note:** The default is such that the key can be used in both UDXs and CCA and none of the user-defined UDX bits are set.	
UDX-ONLY	Specifies that this 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.
Key-usage field 2, high-order byte	
Hash method control (any combination, optional). ***For HMAC algorithm, MAC key type.*** **Note:** All keywords in the list below 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.
Mode control (one, optional). ***For AES algorithm, CIPHER key type.***	
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.
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.
Mode control (one, required). ***For AES algorithm, MAC or PINPRW key types.***	
CMAC	MAC calculation mode is block cipher-based MAC algorithm.
Mode control (one, required). ***For AES algorithm, PINPROT or PINCALC key type.***	
CBC	Specifies that this key can be used for cipher block chaining.
Key-usage field level of control (one, optional). ***For AES algorithm, DKYGENKY key type.*** **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.

Table 72. Key Token Build2 key-usage keywords (continued)

Keyword	Meaning
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-usage field 2, low-order byte	
Key-encrypting key control (any combination, optional). ***For AES algorithm, EXPORTER or IMPORTER key type.*** **Note:** The default is such that the key cannot export a RAW key nor wrap or unwrap a TR-31 key block.	
KEK-RAW	Specifies that this key-encrypting key can export a RAW key. A RAW key is a key that is encrypted, but does not have any associated data.
WR-TR31	Specifies that this key-encrypting key can wrap or unwrap a TR-31 key block.
Key-derivation sequence level (one, required). ***For AES algorithm, DKYGENKY key type.***	
DKYL0	Specifies that this key-generating key can be used to derive the key specified by the Key derivation and Derived key usage controls.
Key-usage field 3, high-order byte	
Key-usage wrap algorithm control (any combination, optional). ***For AES algorithm, EXPORTER or IMPORTER key type.*** **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.
German Banking Industry Committee PIN method Command key-usage. ***For AES algorithm, MAC key type (One, optional).*** ***For AES algorithm, PINCALC, PINPROT, or PINPRW key types (One, required).*** **Note:** • One keyword is required for keys to be used with the DK PIN services. • The command keyword required depends on the service being called. See the description of the key identifiers of the service for the required command key-usage flag.	
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. Valid only with AES PINPROT or MAC key types.
DKPINAD2	Specifies that this key may be used to create or verify an account change string to allow changing the account number associated with a PIN. Valid only with AES MAC.
DKPINOP	Specifies that this key may be used as a general-purpose key. It may not be used as a special-purpose key. Valid for AES MAC, PINCALC, PINPROT, or PINPRW key types.
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. Valid only with AES PINPROT key type with the ENCRYPT keyword specified.

Table 72. Key Token Build2 key-usage keywords (continued)

Keyword	Meaning
Verb data content (required for D-MAC, D-PPROT, D-PCALC and D-DPRW; otherwise optional). ***For AES algorithm, DKYGENKY key type.*** **Note:** Not valid when D-ALL key derivation control is specified.	
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.
Key-usage field 3, low-order byte	
Byte is reserved (except for DK-enabled keys AES MAC, PINCALC, PINPROC, PINPRW).	
Key-usage field 4, high-order byte	
Key-usage wrap class control (any combination, optional). ***For AES algorithm, EXPORTER or IMPORTER key type.*** **Note:** Keywords WR-DATA, WR-KEK, WR-PIN, WRDERIVE and WR-CARD in the list below 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.
Key-usage field 4, low-order byte	
This byte is reserved.	
Key-management field 1, high-order byte	
Symmetric-key export control (one, optional). ***Key-management field 1 for all algorithms and key types.***	
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 for all algorithms and key types.***	
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 for all algorithms and key types.***	
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.
Key-management field 1, low-order byte	
RAW-format export control (one, optional). ***Key-management field 1 for all algorithms and key types.***	
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 for all algorithms, all key types.***	
NOEX-DES	Prohibits the export of the key using DES key.

Table 72. Key Token Build2 key-usage keywords (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 for all algorithms, all key types.	
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 for all algorithms, all key types.	
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.
Key-management field 2, high-order byte	
Byte contains key completeness. There is no user-defined content.	
Key-management field 2, low-order byte	
Byte contains security history. There is no user-defined content.	
Key-management field 3, high-order byte	
Byte contains pedigree original rules. There is no user-defined content.	
Key-management field 3, low-order byte	
Byte contains pedigree current rules. There is no user-defined content.	

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 derive 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.

Note:

– 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.

Required hardware

No cryptographic hardware is required by this callable service.

Key Translate (CSNBKTR and CSNEKTR)

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The *output_key_token* will be wrapped in the same manner as the *input_key_token*.

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 73. Key translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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.

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 zero.

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.

Note: All key labels must be unique.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 (optional)	
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 version1 (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)	

Keyword	Meaning
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.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens.
Translation Control (optional, valid only with WRAP-ENH)	
ENH-ONLY	Restrict rewrapping of the *output_key_token*. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.
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 REFORMAT and AES keywords are specified and *input_key_token* was encrypted under the old master key, the token will be 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 token, the value must be between the actual length of the token and 725. When the *input_KEK_identifier* is a label, the value must be 64.

If the REFORMAT keyword is specified, and *input_key_token* is an AES key token, this parameter must be 0.

input_KEK_identifier

Direction	Type
Input/Output	String

A variable length 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.

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, the *input_KEK_identifier* must be an internal DES token that contains 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.

If the TRANSLAT, V0PYLD, or V1PYLD keyword is specified and the *input_key_token* is an external variable-length key token, the *input_KEK_identifier* must be an internal 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.

If the REFORMAT keyword is specified and *input_key_token* is an external DES key token, this parameter may be an IMPORTER, IKEYXLAT, EXPORTER, or OKEYXLAT key type.

If an internal token was supplied and was encrypted under the old master key, the token 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 token, the value must be between the actual length of the token and 725. When the *output_KEK_identifier* is a label, the value must be 64.

If the REFORMAT, V0PYLD, or V1PYLD keyword is specified, this value must be 0.

output_KEK_identifier

Direction	Type
Input/Output	String

A variable length 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.

If *output_KEK_length* is 0, this parameter is ignored.

If the *output_key_token* is an external DES key, the *output_KEK_identifier* must be an internal DES token that contains 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.

If the *input_key_token* is an external variable-length key token, the *output_KEK_identifier* must be an internal variable-length key token containing an EXPORTER key-encrypting key. The EXPORTER key must have the TRANSLAT bit on in key-usage field 1 of the token.

If an internal token was supplied and was encrypted under the old master key, the token 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*.

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. This 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

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 74. 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 the wrapping method keywords WRAP-ECB or WRAP-ENH to be used when the default key-wrapping method setting does not match the keyword.
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.

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.

Table 75. Key Translate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code. V0PYLD and V1PYLD keywords require the November 2013 or later licensed internal code. AES key support requires the September 2011 or later licensed internal code (LIC). Enhanced key token wrapping and HMAC key support requires the November 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code. V0PYLD and V1PYLD keywords require the September 2013 or later licensed internal code.
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes," on page 943 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 and reason codes," on page 943 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 76 on page 244 for a list of keywords.

Table 76. 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)	
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.
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.

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
Input/Output	String

A 64-byte string that is to receive an internal AES or DES key token.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *key_identifier* will use the default method unless a rule array keyword overriding the default is specified.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 77. 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 WRAP-ECB or WRAP-ENH keywords are specified and default key-wrapping method setting does not match the keyword, 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.

Table 78. Multiple clear key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported. Enhanced key token wrapping not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported. Enhanced key token wrapping not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported. Enhanced key token wrapping not supported.
	Crypto Express3 Coprocessor	Enhanced key token wrapping not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	

Table 78. Multiple clear key import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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 and reason codes," on page 943 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 and reason codes,” on page 943 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, or 4. 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 79 on page 248.

The first keyword is the algorithm. If no algorithm is specified, DES is used. The algorithm keyword applies only when the desired output token is of key form OP and key type IMPORTER, EXPORTER, or DATA. For key form IM or any other key type, specifying DES causes an error.

The second keyword is optional and specifies that the output 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.

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 79. 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.
NOCV Choice (optional)	
NOCV-KEK	The output token is to be marked as an NOCV-KEK. This keyword only applies if key form is OP and key type is IMPORTER, EXPORTER or IMP-PKA. For key form IM or any other key type, specifying NOCV-KEK causes an error.
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*.
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
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 *imported_key_identifier* token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.

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.

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.

Key type values for the Multiple Secure Key Import callable service 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.

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

A 64-byte string internal key token or key label of a DES importer key-encrypting key. This parameter is ignored for AES secure keys.

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

A 64-byte string that is to receive the output key token. If OP is specified in the *key_form* parameter, the service returns an internal key token. If IM is specified in the *key_form* parameter, the service returns an external key token. On input, this parameter is ignored except when the *key_type* is TOKEN. If you specify a *key_type* of TOKEN, then this field contains a valid token of the key type you want to encipher. See *key_type* for a list of valid key types. Appendix B, “Key Token Formats,” on page 995 describes the key tokens.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *imported_key_identifier* will use the default method unless a rule array keyword overriding the default is specified.

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 standard IMPORTERs and EXPORTERs.

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.

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 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

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.

When the WRAP-ECB or WRAP-ENH keywords are specified and default key-wrapping method setting does not match the keyword, the **Multiple Secure Key Import - Allow wrapping override** keywords access control point must be enabled.

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.

Table 81. Multiple secure key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor	Key types CIPHERXI, CIPHERXL, and CIPHERXO are not supported.
	Crypto Express2 Coprocessor	ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Key types CIPHERXI, CIPHERXL, and CIPHERXO are not supported. Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	Key types CIPHERXI, CIPHERXL, and CIPHERXO are not supported. Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
	Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL, and CIPHERXO are not supported. Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL, and CIPHERXO are not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

PKA Decrypt (CSNDPKD and CSNFPKD)

Use this service to decrypt (unwrap) a formatted key value. The service unwraps the key, deformats it, and returns the deformatted value to the application in the clear. PKCS 1.2 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 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 a clear key token is used. ZERO-PAD is only supported for external RSA clear private keys.

This service also supports the use of secure PKCS #11 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 and reason codes," on page 943 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 and reason codes," on page 943 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 82. 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.
ZERO-PAD	The input *PKA_enciphered_keyvalue* is decrypted using the RSA 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 token or the labelname of a external token. This keyword is not valid when using a secure PKCS #11 private key.

PKA_enciphered_keyvalue_length

Direction	Type
Input	Integer

The length of the *PKA_enciphered_keyvalue* parameter in bytes. The maximum size that you can specify is 512 bytes. The length should be the same as the modulus length of the *key_identifier*.

PKA_enciphered_keyvalue

Direction	Type
Input	String

This field contains the key value protected under an RSA 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 3500 bytes.

key_identifier

Direction	Type
Input	String

For PKA keys, an internal RSA private key token, the label of an internal RSA private key token, or an external RSA private key token containing a clear RSA private key 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.

target_keyvalue_length

Direction	Type
Input/Output	Integer

The length of the *target_keyvalue* parameter. The maximum size that you can specify is 512 bytes. On return, this field is updated with the actual length of *target_keyvalue*.

If ZERO-PAD is specified, this length will be the same as the RSA modulus byte length.

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.

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.

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.

Table 83. PKA decrypt required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported.
	PCI Cryptographic Accelerator	Only clear RSA private keys are supported. RSA keys with moduli greater than 2048-bit length are not supported.

Table 83. PKA decrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express2 Accelerator	Only clear RSA private keys are supported. RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express2 Accelerator Crypto Express3 Accelerator	Only clear RSA private keys are supported. RSA keys with moduli greater than 2048-bit length are not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
	Crypto Express3 Accelerator	Only clear RSA private keys are supported. RSA clear key support with moduli within the range 2048-bit to 4096-bit requires the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor (CEX4C)	
	Crypto Express3 Accelerator Crypto Express4 Accelerator (CEX4A)	Only clear RSA private keys are supported.
	Crypto Express4 Enterprise PKCS #11 coprocessor (CEX4P)	Required to use a secure PKCS #11 private key.
IBM z13	Crypto Express5 CCA Coprocessor (CEX5C)	
	Crypto Express5 CCA Accelerator (CEX5A)	Only clear RSA private keys are supported.
	Crypto Express5 Enterprise PKCS #11 coprocessor (CEX5P)	Required to use a secure PKCS #11 private key.

PKA Encrypt (CSNDPKE and CSNFPKE)

This callable service encrypts a supplied clear key value under an RSA 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,
            data_structure_length,
            data_structure,
            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 and reason codes," on page 943 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 and reason codes," on page 943 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 or 2.

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 84. 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.
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 public key must be odd.
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.
Key Rule (Optional)	
KEYIDENT	This indicates that the value in the *keyvalue* field is the label of clear tokens in the CKDS. The *keyvalue_length* must be 64.

keyvalue_length

Direction	Type
Input	Integer

The length of the *keyvalue* parameter. The maximum field size is 512 bytes. The actual maximum size depends on the 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.

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.

data_structure_length

Direction	Type
Input	Integer

This value must be 0.

data_structure

Direction	Type
Input	String

This field is currently ignored.

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 3500 bytes.

PKA_key_identifier

Direction	Type
Input	String

The RSA public or private key token or the label of the RSA public or private key to be used to encrypt the supplied key value.

PKA_enciphered_keyvalue_length

Direction	Type
Input/Output	Integer

The length of the *PKA_enciphered_keyvalue* parameter in bytes. The maximum size that you can specify is 512 bytes. On return, this field is updated with the actual length of *PKA_enciphered_keyvalue*.

This length should be the same as the modulus length of the *PKA_key_identifier*.

PKA_enciphered_keyvalue

Direction	Type
Output	String

This field contains the key value protected under an RSA 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. When the modulus is greater than 2048, the public key exponent must be 3 or 65537.

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 point

The **PKA Encrypt** 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.

Table 85. PKA encrypt required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Routed to a PCICA if one is available (ZERO-PAD and MRP only). RSA keys with moduli greater than 2048-bit length are not supported.
	PCI Cryptographic Accelerator	PKCS-1.2 keyword not supported. RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Routed to a CEX2A if one is available (ZERO-PAD and MRP only). RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express2 Accelerator	PKCS-1.2 keyword not supported. RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Routed to a CEX2A or CEX3A if one is available (ZERO-PAD and MRP only). RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express2 Accelerator Crypto Express3 Accelerator	PKCS-1.2 keyword not supported. RSA keys with moduli greater than 2048-bit length are not supported.

Table 85. PKA encrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Routed to a CEX2A or CEX3A if one is available (ZERO-PAD and MRP only).
	Crypto Express3 Accelerator	PKCS-1.2 keyword not supported. RSA clear key support with moduli within the range 2048-bit to 4096-bit requires the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Routed to a CEX3A or CEX4A if one is available (ZERO-PAD and MRP only).
	Crypto Express3 Accelerator Crypto Express4 CCA Accelerator	PKCS-1.2 keyword not supported.
IBM z13	Crypto Express5 CCA Coprocessor (CEX5C)	Routed to a CEX5A if one is available (ZERO-PAD and MRP only).
	Crypto Express5 CCA Accelerator (CEX5A)	PKCS-1.2 keyword not supported.

Prohibit Export (CSNBPEX and CSNEPEX)

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 and reason codes," on page 943 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 and reason codes," on page 943 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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *key_identifier* will be wrapped in the same manner as the input *key_identifier*.

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.

Table 86. Prohibit export required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.

Table 86. Prohibit export required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.
IBM z13	Crypto Express5 CCA Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.

Prohibit Export Extended (CSNBPEXX and CSNEPEXX)

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 and reason codes,” on page 943 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 and reason codes," on page 943 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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *source_key_token* will be wrapped in the same manner as the input *source_key_token*.

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

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.

Table 87. Prohibit export extended required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	External MACD keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	External MACD keys are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	External MACD keys are not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	External MACD keys are not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	External MACD keys are not supported.
IBM z13	Crypto Express5 CCA Coprocessor	External MACD keys are not supported.

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

The callable service uses a cryptographic feature to generate a random number. The foundation for the random number generator is a time variant input with a very low probability of recycling.

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,
              reserved_length,
              reserved,
              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 and reason codes," on page 943 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 and reason codes," on page 943 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 88.

Table 88. 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 one.

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 89. Keywords for Random Number Generate Control Information

Keyword	Meaning
Parity of the random number bytes (required)	
EVEN	Generate a random number with even parity in each byte. Its length is the *random_number_length*.
ODD	Generate a random number with odd parity in each byte. Its length is the *random_number_length*.
RANDOM	Generate a random number. Its length is the *random_number_length*.

reserved_length

Direction	Type
Input	Integer

This parameter must be zero.

reserved

Direction	Type
Input	String

This parameter is ignored.

random_number_length

Direction	Type
Input	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.

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.

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.

The CSNBRNGL callable service returns a value under the following conditions:

- The server has the cryptographic coprocessor that supports CSNBRNGL and the coprocessor creates the random number with the desired length. This requires a CCA Crypto Express coprocessor with a version of the licensed internal code (LIC) that supports the RNGL verb.
- The server has the cryptographic coprocessor that processes CSNBRNG requests. In this case, the CSNBRNGL callable service calls the processor to create the random number with the desired length, 8 bytes at a time.
- The server has the CP Assist for Cryptographic Functions with a feature level that supports pseudo-random number generation.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 90. Random number generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	

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, enhanced wrapping mode (WRAP-ENH) or original ECB wrapping mode (WRAP-ECB).

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 91. 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.
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.

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 92. 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
24	1	Identifier for the hash algorithm used
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.

These values are defined for the hash algorithm identifier at offset 24 in the structure.

Table 93. Values defined for hash algorithm identifier at offset 24 in the structure for remote key export

Identifier	Algorithm
0X01	SHA-1
0X02	MD5
0X03	RIPEMD-160

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 94. Transport_key_identifer used by RKX

Flag Byte Offset 00	Version Number Offset 04	Description
0X01	0X00	Internal DES key token version 0
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. This parameter is known as the *source_key_identifier* in other callable services.

Table 95. 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, and so 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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The *sym_encrypted_key* will be wrapped in the same manner as the *source_key_identifier*.

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 431 on page 1052.

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

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.

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.

When the **RKX/TBC – Disallow triple-length MAC key** access control point is enable, this service will not import a triple-length MAC wrapped with a double-length KEK.

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 Key-Wrapping Configuration** 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 WRAP-ECB or WRAP-ENH keywords are specified, 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.

Table 96. Remote key export required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express 2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
	Crypto Express3 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	ENH-ONLY, USECONFG, WRAP-ENH and WRAP-ECB require the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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 and reason codes," on page 943 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 and reason codes," on page 943 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 97. Keywords for Restrict Key Attribute Control Information

Keyword	Meaning
Token Type (Required)	
AES	Specifies the key token is an AES key token.
DES	Specifies the key token is a DES key token.
HMAC	Specifies the key token is an HMAC key token.
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. Internal DES key token: Use this keyword to set CV bit 17 = B'0' (NO-XPORT) and CV bit 27 = B'1' (NOT31XPT). This is the default if no DES attribute restriction keywords are used.
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.
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).

Table 97. Keywords for Restrict Key Attribute Control Information (continued)

Keyword	Meaning
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 internal tokens, set bit 17 of the CV to 0 to prohibit any export of the key.
NOT31XPT	For DES internal 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. **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. The maximum value is 900.

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 token or the 64-byte CKDS label of an internal token. 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 an AES or DES token encrypted under the old master key, the token 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.
- If *key_encrypting_key_identifier* is an RSA KEK, the maximum length is 3500.

key_encrypting_key_identifier

Direction	Type
Input/Output	String

When *key_encrypting_key_identifier_length* is non-zero, *key_encrypting_key_identifier* contains an internal key token containing a key-encrypting key, or a key label.

If the key identifier supplied was an AES or DES token encrypted under the old master key, the token 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:

- Restrict Key Attribute - Export Control
- Restrict Key Attribute - Permit setting the TR-31 export bit

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 98. Restrict Key Attribute required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code. AES key support requires the September 2011 or later licensed internal code (LIC). Enhanced key token wrapping and HMAC key support requires the November 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code.
IBM z13	Crypto Express5 CCA Coprocessor	

Secure Key Import (CSNBSKI and CSNESKI)

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.

To import double-length and triple-length DATA keys, or double-length MAC, MACVER, CIPHER, DECIPHER and ENCIPHER keys, use the Multiple Secure Key Import callable service. See "Multiple Secure Key Import (CSNBSKM and CSNESKM)" on page 246.

To import AES DATA keys, use the multiple secure key import service ("Multiple Secure Key Import (CSNBSKM and CSNESKM)" on page 246).

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 and reason codes," on page 943 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 and reason codes," on page 943 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	Character String

The generated encrypted key. The parameter is a 64-byte string. The callable service returns either an internal key token if you encrypted the clear key under the master key (*key_form* was OP); or an external key token if you encrypted the clear key under the importer key-encrypting key (*key_form* was IM).

If the *key_type* parameter is not TOKEN, this parameter must be a 64-byte string of hex zero

If the imported key_type is IMPORTER or EXPORTER and the key_form is OP, the *key_identifier* parameter changes direction to both input and output. If 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.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *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.

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 table shows the access control points in the domain role that control the function of this service.

Table 99. 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.

Table 100. Secure key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	

Table 100. Secure key import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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 and reason codes," on page 943 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 and reason codes," on page 943 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 101. 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.
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 Key Form is IM, the value must be between the actual length of the token and 725 when *key_encrypting_key_identifier* is a token. The value must be 64 when *key_encrypting_key_identifier* is a label.

key_encrypting_key_identifier

Direction	Type
Input/Output	String

When the Key Form rule is OP, *key_encrypting_key_identifier* is ignored. When the Key Form rule is IM, *key_encrypting_key_identifier* contains an internal key token containing the AES importer key-encrypting key or a key label.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

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.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 102. 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 **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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 103. Secure Key Import2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.

Table 103. Secure Key Import2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code. V0PYLD and V1PYLD keywords require the November 2013 or later licensed internal code. AES key support requires the September 2011 or later licensed internal code (LIC). HMAC key support requires the November 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code. V0PYLD and V1PYLD keywords require the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

Symmetric Key Export (CSNDSYX and CSNFSYX)

Use the symmetric key export callable service to transfer an application-supplied AES, DES or variable-length symmetric key token 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, DES, or HMAC 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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 104 on page 294 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 104. 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 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.
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 the method found in RSA DSI PKCS #1 block type 02 to recover the symmetric key.
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.

source_key_identifier_length

Direction	Type
Input	Integer

The length of the *source_key_identifier* parameter. The minimum size is 64 bytes. The maximum size is 725 bytes.

source_key_identifier

Direction	Type
Input/Output	String

The label or internal token of a secure AES DATA (version X'04'), DES DATA, or variable-length symmetric key token to encrypt under the supplied RSA public key or a secure AES or DES key token to encrypt under the supplied AES EXPORTER key. The key in the key identifier must match the algorithm in the *rule_array*. DES is the default algorithm.

transporter_key_identifier_length

Direction	Type
Input	Integer

The length of the *transporter_key_identifier* parameter. The maximum size is 3500 bytes for an RSA key token or 725 for an AES EXPORTER key token. The length must be 64 if *transporter_key_identifier* is a label.

transporter_key_identifier

Direction	Type
Input	String

An RSA public key token, AES EXPORTER token, or label of the key to protect the exported symmetric key.

When the AESKW or AESKWCV Key Formatting method is specified, this parameter must be an AES EXPORTER key token or label with the EXPORT bit on in the key-usage field. Otherwise, this parameter must be an RSA public key token or label.

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 107 on page 297).

enciphered_key

Direction	Type
Output	String

This field contains the exported key, protected by the RSA public or AES EXPORTER key specified in the *transporter_key_identifier* field.

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 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 105. 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 106. 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

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.

Table 107. Symmetric key export required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported. Encrypted AES keys are not supported. The AESKW, AESKWCV, HMAC, and PKOAEP2 keywords are not supported. The SHA-256 keyword is not supported for PKCSOAEP.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the November 2007 or later licensed internal code (LIC). Encrypted AES key support requires the November 2008 or later licensed internal code (LIC). The AESKW, AESKWCV, HMAC, and PKOAEP2 keywords are not supported. The SHA-256 keyword is not supported for PKCSOAEP.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the November 2007 or later licensed internal code (LIC). Encrypted AES key support requires the November 2008 or later licensed internal code (LIC). The AESKW, AESKWCV, HMAC, and PKOAEP2 keywords are not supported. The SHA-256 keyword is not supported for PKCSOAEP.
	Crypto Express3 Coprocessor	The AESKW, AESKWCV, HMAC, and PKOAEP2 keywords are not supported. The SHA-256 keyword is not supported for PKCSOAEP.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code. Variable-length AES Keys, the AESKW method, and PKCSOAEP with the SHA-256 hash method require the September 2011 or later licensed internal code (LIC). HMAC key support requires the November 2010 or later licensed internal code (LIC). The AESKWCV keyword is not supported.

Table 107. Symmetric key export required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code. AESKWCV requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes," on page 943 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 and reason codes," on page 943 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 108. 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. 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. For DES keys, the value must be 64. For AES keys, the maximum value is 725.

source_key_identifier

Direction	Type
Input	String

An internal key token or the label of the CKDS record containing an operational AES or DES key token that is to be exported. If the key is a DES key, bit 17 of the control vector must be equal to '1'b (XPORT-OK). The key must have a control vector of DATAC or DKYGENKY with subtype DKYL0, unless the “Allow Symmetric Key Export with Data Special” access control point is enabled. If the AES key is in a fixed length key token, no control vector checking is needed. If the AES key is in a variable length token, the key type must be CIPHER. If the key type is not CIPHER, an access control point “Allow Symmetric Key Export with Data Special” must be enabled. If they key is an AES key, the key management field in the key must allow export by RSA keys and by unauthenticated asymmetric keys.

If the token supplied was encrypted under the old master key, the token 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

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 109. 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

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 110. Symmetric key export with data required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.

Table 110. Symmetric key export with data required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise 196 IBM zEnterprise 114		This service is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Recover PIN From Offset requires the Sep. 2013 or later LIC.
IBM z13	Crypto Express5 CCA Coprocessor	

Symmetric Key Generate (CSNDSYG and CSNFSYG)

Use the symmetric key generate callable service to generate an AES 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 a RSA public key according to the PKA92 formatting structure. See “PKA92 Key Format and Encryption Process” on page 1111 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 and reason codes,” on page 943 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 and reason codes," on page 943 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 111 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 111. Keywords for Symmetric Key Generate Control Information

Keyword	Description	Algorithm
Algorithm (one keyword, optional)		
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)		

Table 111. Keywords for Symmetric Key Generate Control Information (continued)

Keyword	Description	Algorithm
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
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		
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

Table 111. Keywords for Symmetric Key Generate Control Information (continued)

Keyword	Description	Algorithm
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. See the *z/OS Cryptographic Services ICSF System Programmer's Guide*.	AES and DES
WRAP-ENH	Use enhanced key wrapping method, 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.	AES or 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.	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
SHA-256	Specifies to use the SHA-256 hash method to calculate the OAEP message hash.	AES or DES

key_encrypting_key_identifier

Direction	Type
Input/Output	String

The label or internal token of a key-encrypting key. If the *rule_array* specifies IM, this DES key must be an IMPORTER. If the *rule_array* specifies EX, this DES key must be an EXPORTER. Otherwise, the parameter is ignored.

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 the RSA public key to be used for protecting the generated symmetric key.

local_enciphered_key_token_length (was DES_enciphered_key_token_length)

Direction	Type
Input/Output	Integer

The length in bytes of the local_enciphered_key_token. This field is updated with the actual length of the token that is generated. The minimum length is 64-bytes and the maximum length is 128 bytes.

local_enciphered_key_token (was DES_enciphered_key_token)

Direction	Type
Input/Output	String

This parameter contains the generated DATA key in the form of an internal or external token, depending on rule_array specification. If you specify PKA92, on input specify an internal (operational) key token of an Importer or Exporter Key.

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.

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 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 1111). 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 112. 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 WRAP-ECB or WRAP-ENH keywords are specified and the default key-wrapping method setting does not match the keyword, 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.

Table 113. Symmetric key generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported. Secure AES keys are not supported. ENH-ONLY, USECONFG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported. PKCSOAEP with the SHA-256 hash method is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH, WRAP-ECB, and SHA-256 not supported. PKCSOAEP with the SHA-256 hash method is not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH, WRAP-ECB, and SHA-256 not supported. PKCSOAEP with the SHA-256 hash method is not supported.
	Crypto Express3 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC). The SHA-256 keyword is not supported. PKCSOAEP with the SHA-256 hash method is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	PKCSOAEP with the SHA-256 hash method requires the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 import of a PKA92-formatted DES key-encrypting key 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 and reason codes," on page 943 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 and reason codes," on page 943 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 114 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 114. Keywords for Symmetric Key Import Control Information

Keyword	Meaning
Algorithm (one keyword, optional)	
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. See the *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Table 114. Keywords for Symmetric Key Import Control Information (continued)

Keyword	Meaning
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
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.
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.

Restrictions

The exponent of the RSA public key must be odd.

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 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 115. 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 WRAP-ECB or WRAP-ENH keywords are specified and the default key-wrapping method setting does not match the keyword, the **Symmetric 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.

Table 116. Symmetric key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported. Encrypted AES keys are not supported. ENH-ONLY, USECONFG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported. PKCSOAEP with the SHA-256 hash method is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). Encrypted AES keys are not supported. ENH-ONLY, USECONFG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported. PKCSOAEP with the SHA-256 hash method is not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). Encrypted AES key support requires the Nov. 2008 or later licensed internal code (LIC). ENH-ONLY, USECONFG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported. PKCSOAEP with the SHA-256 hash method is not supported.
	Crypto Express3 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). Encrypted AES key support requires the Nov. 2008 or later licensed internal code (LIC). The SHA-256 keyword is not supported. PKCSOAEP with the SHA-256 hash method is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	PKCSOAEP with the SHA-256 hash method requires the Sep. 2011 or later licensed internal code (LIC).

Table 116. Symmetric key import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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, either enhanced wrapping mode (WRAP-ENH) or original ECB wrapping mode (WRAP-ECB).

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 and reason codes,” on page 943 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 and reason codes," on page 943 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 117. 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.

Table 117. Keywords for Symmetric Key Import2 Control Information (continued)

Keyword	Meaning
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. The Access Control Point Symmetric Key Import2 – Allow wrapping override keywords must be enabled to specify these keywords)*	
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.
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.

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

The key to import, protected under either an RSA public key or an AES KEK. If the Recovery Method is PKOAEP2, 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. If the Recovery Method is AESKW, the encrypted key is an AES key or HMAC key in the external variable length key token. If the Recovery Method is AESKWCV, the encrypted key is a DES key in an external variable length DESUSECV key token.

transport_key_identifier_length

Direction	Type
Input	Integer

The length of the *transport_key_identifier* parameter. When the *transport_key_identifier* parameter is a key label, this field must be 64. The maximum size is 3500 bytes for an RSA private key or 725 bytes for an AES IMPORTER KEK.

transport_key_identifier

Direction	Type
Input	String

An internal RSA private key token, internal AES IMPORTER KEK, or the 64-byte label of a key token whose corresponding key protects the symmetric key.

When the AESKW or AESKWCV Key Formatting method is specified, this parameter must be an AES IMPORTER with the IMPORT bit on in the key-usage field. Otherwise, this parameter must be an RSA private 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.

Restrictions

The exponent of the RSA public key must be odd.

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 is the message layout used to encode the key material exported with the new PKOAEP2 formatting method.

Table 118. 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 119. Symmetric Key Import2 Access Control Points

Key formatting method	Token Algorithm	Access control point
PKOAEP2	HMAC, AES	Symmetric Key Import2 - HMAC/AES, PKOAEP2
AESKW	HMAC, AES	Symmetric Key Import2 - HMAC/AES, 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 wrapping method specified is not the default method, then 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.

Table 120. Symmetric key import2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	HMAC key support requires the November 2010 or later licensed internal code (LIC). AES key support and the AESKW wrapping method require the September 2011 or later licensed internal code (LIC). DK AES PIN key support requires the November 2013 or later licensed internal code. DES, AESKWCV, USECONFG, WRAP-ECB, WRAP-ENH and ENH-ONLY keywords are not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code. DES, AESKWCV, USECONFG, WRAP-ECB, WRAP-ENH and ENH-ONLY keywords require the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

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 TSS Return and Reason Codes 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 and reason codes," on page 943 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 121. 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 121 on page 321 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 122. 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

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 123. Trusted Block Create required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC	Cryptographic Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).

Table 123. Trusted Block Create required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

TR-31 Export (CSNBT31X and CSNET31X)

Use the TR-31 Export callable service to convert a CCA token to TR-31 format for export to another party. 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 exported 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 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 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. See Table 125 on page 330 for valid combinations of Usage and Mode

Table 124. Keywords for TR-31 Export Rule Array Control Information

Keyword	Meaning
TR-31 key block protection method – one required	
VARXOR-A	Use the variant method corresponding to a TR-31 Key Block Version ID of “A” (0x41)

Table 124. Keywords for TR-31 Export Rule Array Control Information (continued)

Keyword	Meaning
VARDRV-B	Use the key derivation method corresponding to a TR-31 Key Block Version ID of “B” (0x42)
VARXOR-C	Use the variant method corresponding to a TR-31 Key Block Version ID of “C” (0x43)
TR-31 key usage values for output key – one required **Note:** If ATTR-CV is specified from the Control Vector Transport group, then usage keyword must not be specified. The proprietary usage ‘10’ will be used.	
BDK	Base Derivation Key (BDK) – (B0)
CVK	Card Verification Key (CVK) – (C0)
ENC	Data encryption key – (D0)
EMVACMK	EMV application cryptogram master key – (E0)
EMVSCMK	EMV secure messaging for confidentiality master key – (E1)
EMVSIMK	EMV secure messaging for integrity master key – (E2)
EMVDAMK	EMV data authentication code key – (E3)
EMVDNMK	EMV dynamic numbers master key – (E4)
EMVCPMK	EMV card personalization master key – (E5)
KEK	Key-encrypting key – (K0)
KEK-WRAP	Key-encrypting key for wrapping TR-31 blocks (for ‘B’ and ‘C’ TR-31 Key Block Version IDs only) – (K1)
ISOMAC0	Key for ISO 16609 MAC algorithm 1 using TDES – (M0)
ISOMAC1	Key for ISO 9797-1 MAC algorithm 1– (M1)
ISOMAC3	Key for ISO 9797-1 MAC algorithm 3– (M3)
PINENC	PIN encryption key – (P0)
PINVO	PIN verification key, “other” algorithm – (V0)
PINV3624	PIN verification key for IBM 3624 algorithm – (V1)
VISAPVV	PIN verification key, VISA PVV algorithm – (V2)
TR-31 modes of key use – one required **Note:** If ATTR-CV is specified from the Control Vector Transport group, then mode keyword must not be specified. The proprietary mode ‘1’ will be used.	
ENCDEC	Encrypt and decrypt – (B)
DEC-ONLY	Decrypt only – (D)
ENC-ONLY	Encrypt only – (E)
GENVER	MAC or PIN generate and verify – (C) • MAC key must have Gen and Ver bits on • PIN key must have any PINGEN bit and EPINVER bit on
GEN-ONLY	MAC or PIN generate only – (G) • MAC key must have only Gen bit on • PIN key must have any PINGEN bit on and EPINVER bit off
VER-ONLY	MAC or PIN verify only– (V) • MAC key must have only Ver bit on • PIN key must have all PINGEN bits off and EPINVER bit on
DERIVE	Key Derivation(for ‘B’ and ‘C’ TR-31 Key Block Version IDs only) – (X)

Table 124. Keywords for TR-31 Export Rule Array Control Information (continued)

Keyword	Meaning
ANY	Any mode allowed – (N)
Export control to set export field in TR-31 key block – optional	
EXP-ANY	Export allowed using any key-encrypting key. This is the default.
EXP-TRST	Export allowed using a trusted key-encrypting key, as defined in TR-31. **Note:** A CCA key wrapped in the X9.24 compliant CCA key block is considered a trusted key.
EXP-NONE	Export prohibited
Control vector transport control – optional **Note:** 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 groups above.	
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 above 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:** If this keyword is specified, then usage and mode keywords from the preceding groups must not be specified. The proprietary values will be used.

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.

key_field_length

Direction	Type
Input	Integer

This parameter specifies the length of the key field which is encrypted in the TR-31 block. The length must be a multiple of 8, the DES cipher block size, and it must be greater than or equal to the length of the cleartext key passed with parameter *source_key_identifier* plus the length of the 2-byte key length that precedes this key in the TR-31 block. For example, if the source key is a double-length TDES key of length 16 bytes, then the key field length must be greater than or equal to (16+2) bytes, and must also be a multiple of 8. This means that the minimum *key_field_length* in this case would be 24. TR-31 allows a variable number of padding bytes to follow the cleartext key, and the caller may choose to pad with more than the minimum number of bytes needed to form a block that is a multiple of 8. This is generally done to hide the length of the cleartext key from those who cannot decipher that key. Most often, all keys – single, double, or triple length – are padded to the same length so that it is not possible to determine which length is carried in the TR-31 block by examining the encrypted block.

Note that this parameter is not expected to allow for ASCII encoding of the encrypted data stored in the key field according to the TR-31 specification. For example when the user passes a value of 24 here, following the minimum example above, the length of the final ASCII-encoded encrypted data in the key field in the output TR-31 key block will be 48 Bytes.

source_key_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *source_key_identifier* parameter, in bytes. The value in this parameter must currently be 64, since only CCA key tokens are supported for the source key parameter.

source_key_identifier

Direction	Type
Input/Output	String

This parameter contains either the label or the key token for the key that is to be exported. The key must be a CCA internal or external token. 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. Only DES/TDES keys are supported. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped 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. If the *source_key_identifier* is an external CCA token, then this parameter must be 64. Otherwise, this parameter must be 0.

unwrap_kek_identifier

Direction	Type
Input/Output	String

When the *source_key_identifier* is an external CCA token, this parameter contains either the label or the key token for the KEK which the *source_key_identifier* is currently wrapped under. It must be a CCA internal DES KEK token of type EXPORTER or OKEYXLAT. If the *source_key_identifier* is not an external CCA token, this parameter is ignored. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped 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. If the *unwrap_kek_identifier* is also to be used to wrap the output TR-31 key block, specify 0 for this parameter. Otherwise, this parameter must be 64.

wrap_kek_identifier

Direction	Type
Input/Output	String

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 TR-31 key block. Otherwise, this parameter contains either the label or the key token for the KEK to use for wrapping the output TR-31 key block. It must be a CCA internal token for a KEK EXPORTER or OKEYXLAT type and must have the same clear key as the *unwrap_kek_identifier*. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped 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.

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.

`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

This callable service only exports DES and TDES keys.

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 1119). 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 export 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.

Access control points

The access control points in the domain role that control the general function of this service are:

- TR31 Export – Permit version A TR-31 key blocks
- TR31 Export – Permit version B TR-31 key blocks
- TR31 Export – Permit version C TR-31 key blocks

If the wrap KEK 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.

The following table 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 from below.

Table 125. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all ‘usage’ + ‘mode’ here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg’m	Required “TR31 Export” ACP
Any Exportable Key							
Permit export of any CCA key (for allowable export scenarios as defined by this table) as long as the TR-31 key block will have the CCA Control Vector (CV) included as an optional block (the INCL-CV keyword was supplied on the callable service). Normally export of CCA keys to TR-31 key blocks is controlled by ACPs specific to the translation or a small set of translations, to give fine control. This ACP allows any allowable export to occur as long as the CV is included, thus overriding the specific ACPs. **Note:** 1. Some target systems, produced by other vendors may not accept TR-31 key blocks with the proprietary optional CV block. 2. The ATTR-CV keyword does not require any ACPs.							
DUKPT Base Derivation Keys							
KEYGENKY	UKPT	BDK + ANY	B0	A	N	T	Permit KEYGENKY:UKPT to B0
KEYGENKY	UKPT	BDK + DERIVE	B0	B,C	X	T	
Note: These are the base keys from which DUKPT initial keys are derived for individual devices such as PIN pads							
Card Verification Keys							
MAC	AMEX-CSC, gen bit(20)=1	CVK + GEN-ONLY	C0	A,B,C	G	D,T	Permit MAC/MACVER:AMEX-CSC to C0:G/C/V
	AMEX-CSC, gen bit(20)=0, ver bit(21)=1	CVK + VER-ONLY		A,B,C	V	D,T	
	AMEX-CSC, gen bit(20)=1, ver bit(21)=1	CVK + GENVER		A,B,C	C	D,T	
	CVVKEY-A, gen bit(20)=1	CVK + GEN-ONLY		A,B,C	G	T	Permit MAC/MACVER:CVV-KEYA to C0:G/C/V
	CVVKEY-A, gen bit(20)=0, ver bit(21)=1	CVK + VER-ONLY		A,B,C	V	T	
	CVVKEY-A, gen bit(20)=1, ver bit(21)=1	CVK + GENVER		A,B,C	C	T	

Table 125. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
	ANY-MAC, gen bit(20)=1	CVK + GEN-ONLY		A,B,C	G	T	Permit MAC/MACVER:ANY-MAC to C0:G/C/V
	ANY-MAC, gen bit(20)=0, ver bit(21)=1	CVK + VER-ONLY		A,B,C	V	T	
	ANY-MAC, gen bit(20)=1, ver bit(21)=1	CVK + GENVER		A,B,C	C	T	
DATA	gen bit(20)=1 or zeroCV	CVK + GEN-ONLY		A,B,C	G	T	Permit DATA to C0:G/C
	gen bit(20)=1, ver bit(21)=1 or zeroCV	CVK + GENVER		A,B,C	C	T	

Note:

1. Keys 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 different APIs.
   - Visa CVV and MasterCard CVC codes are computed with CVV_Generate and verified with CVV_Verify. Keys must be DATA or MAC with sub-type (in bits 0-3) "ANY-MAC" , "CVVKEY-A" or "CVVKEY-B". The GEN bit (20) or VER bit (21) must be set appropriately.
   - American Express CSC codes are generated and verified with the Transaction_Validate verb. The key must be a MAC or MACVER key with sub-type "ANY-MAC" or "AMEX-CSC". The GEN bit (20) or VER bit (21) must be set appropriately.
2. CCA and TR-31 represent CVV keys incompatibly. CCA represents the "A" and "B" keys as two 8 B keys, while TR-31 represents these as one 16 B key. The CVV generate and verify verbs now accept a 16 B CVV key, using left and right parts as A and B. Current Visa standards require this.
3. Import and export of the 8 B CVVKEY-A and CVVKEY-B types will only be allowed using the proprietary TR-31 usage+mode values to indicate encapsulation of the IBM CV in an optional block, since the 8 B CVVKEY-A is meaningless / useless as a TR-31 C0 usage key of any mode.
4. It is possible to convert a CCA CVV key into a CSC key or vice-versa, since the translation from TR 31 usage "C0" is controlled by rule array keywords on the import verb. This can be restricted by using ACPs, but if both of translation types are required they cannot be disabled and control is up to the development, deployment, and execution of the applications themselves

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
Data Encryption Keys							
ENCIPHER	(none)	ENC + ENC-ONLY	D0	A,B,C	E	D, T	Permit ENCIPHER/DECIPHER/ CIPHER to D0:E/D/B
DECIPHER	(none)	ENC + DEC-ONLY		A,B,C	D	D, T	
CIPHER	(none)	ENC + ENCDEC		A,B,C	B	D, T	
DATA	enc bit(18)=1, dec bit(19)=1 or zeroCV	ENC + ENCDEC		A,B,C	B	D, T	Permit DATA to D0:B

Note: There is asymmetry in the TR-31 to CCA and CCA to TR-31 translation. CCA keys can be exported to TR-31 'D0' keys from CCA type ENCIPHER, DECIPHER, or CIPHER, or type DATA with proper Encipher and Decipher CV bits on. A TR-31 'D0' key can only be imported to CCA types ENCIPHER, DECIPHER, or CIPHER, not the lower security DATA key type. This eliminates conversion to the lower security DATA type by export / re-import.

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
Key Encrypting Keys							
EXPORTER or OKEYXLAT		KEK + ENC-ONLY	K0	A,B,C	E	T	Permit EXPORTER/OKEYXLAT to K0:E
IMPORTER or IKEYXLAT		KEK + DEC-ONLY	K0	A,B,C	D	T	Permit IMPORTER/IKEYXLAT to K0:D

Table 125. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
EXPORTER or OKEYXLAT		KEK-WRAP + ENC-ONLY	K1	B,C	E	T	Permit EXPORTER/OKEYXLAT to K1:E
IMPORTER or IKEYXLAT		KEK-WRAP + DEC-ONLY	K1	B,C	D	T	Permit IMPORTER/IKEYXLAT to K1:D
Note: 1. To be exported a KEK must have either the EXPORTER/IMPORTER bit or the XLAT bit on in the CV. A KEK with only the Key Generate bits on will not be exportable. 2. 'K1' keys are not distinguished from 'K0' keys within CCA. The 'K1' key is a particular KEK for deriving keys used in the 'B' or 'C' version wrapping of TR-31 key blocks. CCA does not distinguish between targeted protocols currently and so there is no good way to represent the difference; also note that most wrapping mechanisms now involve derivation or key variation steps 3. The CCA KEK to TR-31 K0-B transition for export will not be allowed for security reasons, even with ACP control this gives an immediate path to turn a CCA EXPORTER to an IMPORTER and vice versa. Export of NO-CV KEKs will be allowed, exporter keys become 'E' mode normal K0 keys, importer keys become 'D' mode K0 keys. A user can turn any KEK to a NO-CV KEK by setting the flag bit and recalculating the TVV, the flag is not bound to the key like the CV is.							
MAC Keys							
MAC	gen bit(20)=1	ISOMAC0 + GEN-ONLY	M0	A,B,C	G	T	Permit MAC/DATA/DATAM to M0:G/C
DATA	gen bit(20)=1 or zeroCV	ISOMAC0 + GEN-ONLY		A,B,C	G	T	
MAC	gen bit(20)=1, ver bit(21)=1	ISOMAC0 + GENVER		A,B,C	C	T	
DATAM	gen bit(20)=1, ver bit(21)=1	ISOMAC0 + GENVER		A,B,C	C	T	
DATA	gen bit(20)=1, ver bit(21)=1 or zeroCV	ISOMAC0 + GENVER		A,B,C	C	T	
MACVER	gen bit(20)=0, ver bit(21)=1	ISOMAC0 + VER-ONLY		A,B,C	V	T	Permit MACVER/DATAMV to M0:V
DATAMV	gen bit(20)=0, ver bit(21)=1	ISOMAC0 + VER-ONLY		A,B,C	V	T	
MAC	gen bit(20)=1	ISOMAC1 + GEN-ONLY	M1	A,B,C	G	D,T	Permit MAC/DATA/DATAM to M1:G/C
DATA	gen bit(20)=1 or zeroCV	ISOMAC1 + GEN-ONLY		A,B,C	G	D,T	
MAC	gen bit(20)=1, ver bit(21)=1	ISOMAC1 + GENVER		A,B,C	C	D,T	
DATAM	gen bit(20)=1, ver bit(21)=1	ISOMAC1 + GENVER		A,B,C	C	D,T	
DATA	gen bit(20)=1, ver bit(21)=1 or zeroCV	ISOMAC1 + GENVER		A,B,C	C	D,T	
MACVER	gen bit(20)=0, ver bit(21)=1	ISOMAC1 + VER-ONLY		A,B,C	V	D,T	Permit MACVER/DATAMV to M1:V
DATAMV	gen bit(20)=0, ver bit(21)=1	ISOMAC1 + VER-ONLY		A,B,C	V	D,T	

Table 125. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
MAC	gen bit(20)=1	ISOMAC3 + GEN-ONLY	M3	A,B,C	G	D,T	Permit MAC/DATA/DATAM to M3:G/C
DATA	gen bit(20)=1 or zeroCV	ISOMAC3 + GEN-ONLY		A,B,C	G	D,T	
MAC	gen bit(20)=1, ver bit(21)=1	ISOMAC3 + GENVER		A,B,C	C	D,T	
DATAM	gen bit(20)=1, ver bit(21)=1	ISOMAC3 + GENVER		A,B,C	C	D,T	
DATA	gen bit(20)=1, ver bit(21)=1 or zeroCV	ISOMAC3 + GENVER		A,B,C	C	D,T	
MACVER	gen bit(20)=0, ver bit(21)=1	ISOMAC3 + VER-ONLY		A,B,C	V	D,T	Permit MACVER/DATAMV to M3:V
DATAMV	gen bit(20)=0, ver bit(21)=1	ISOMAC3 + VER-ONLY		A,B,C	V	D,T	
Note: 1. M0 and M1 are identical (ISO 16609 based on ISO 9797) normal DES/TDES (CBC) MAC computation, except M1 allows 8 byte and 16 byte keys while M0 allows only 16 byte keys. Mode M3 is the X9.19 style triple-DES MAC. 2. CCA does not support M2, M4, or M5. 3. Although export of DATAM/DATAMV keys to TR-31 M0/M1/M3 key types is allowed, import to DATAM/DATAMV CCA types is not allowed since they are obsolete types.							
PIN Keys							
OPINENC	(none)	PINENC + ENC-ONLY	P0	A,B,C	E	T	Permit OPINENC to P0:E
IPINENC	(none)	PINENC + DEC-ONLY		A,B,C	D	T	Permit IPINENC to P0:D
(none)	(none)	(none)	(none)	A,B,C	B	T	(none)
PINVER	NO-SPEC	PINVO + ANY	V0	A	N	T	Permit PINVER:NO-SPEC to V0, Permit PINGEN/PINVER to V0/V1/V2:N
	[no GEN bits on in CV]	PINVO + VER-ONLY		A,B,C	V		Permit PINVER:NO-SPEC to V0
PINGEN	NO-SPEC	PINVO + ANY		A	N	T	Permit PINGEN:NO-SPEC to V0, Permit PINGEN/PINVER to V0/V1/V2:N
	[EPINVER bit off in CV]	PINVO + GEN-ONLY		A,B,C	G		Permit PINGEN:NO-SPEC to V0
	[EPINVER bit on in CV]	PINVO + GENVER		A,B,C	C		Permit PINGEN:NO-SPEC to V0
PINVER	IBM or NO-SPEC	PINV3624 + ANY	V1	A	N	T	Permit PINVER:NO-SPEC/IBM-PIN/IBM-PINO to V1, Permit PINGEN/PINVER to V0/V1/V2:N
	[no GEN bits on in CV]	PINV3624 + VER-ONLY		A,B,C	V		Permit PINVER:NO-SPEC/IBM-PIN/IBM-PINO to V1
PINGEN	IBM or NO-SPEC	PINV3624 + ANY		A	N	T	Permit PINGEN:NO-SPEC/IBM-PIN/IBM-PINO to V1, Permit PINGEN/PINVER to V0/V1/V2:N
	[EPINVER bit off in CV]	PINV3624 + GEN-ONLY		A,B,C	G		Permit PINGEN:NO-SPEC/IBM-PIN/IBM-PINO to V1

Table 125. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
	[EPINVER bit on in CV]	PINV3624 + GENVER		A,B,C	C		Permit PINGEN:NO-SPEC/IBM-PIN/ IBM-PINO to V1
PINVER	VISAPVV or NO-SPEC	VISAPVV + ANY	V2	A	N	T	Permit PINVER:NO-SPEC/VISA-PVV to V2, Permit PINGEN/PINVER to V0/V1/V2:N
	[no GEN bits on in CV]	VISAPVV + VER-ONLY		A,B,C	V		Permit PINVER:NO-SPEC/VISA-PVV to V2
PINGEN	VISAPVV or NO-SPEC	VISAPVV + ANY		A	N	T	Permit PINGEN:NO-SPEC/VISA-PVV to V2, Permit PINGEN/PINVER to V0/V1/V2:N
	[EPINVER bit off in CV]	VISAPVV + PINGEN		A,B,C	G		Permit PINGEN:NO-SPEC/VISA-PVV to V2
	[EPINVER bit on in CV]	VISAPVV + PINGEN		A,B,C	C		Permit PINGEN:NO-SPEC/VISA-PVV to V2
Note: There is a subtle difference between TR-31 V0 mode and CCA 'NO-SPEC' subtype. V0 mode restricts keys from 3224 or PVV methods, while CCA 'NO-SPEC' allows any method. Turning on the ACP or ACPs controlling export of PINVER to usage:mode V*:N and import of V*:N to PINGEN at the same time will allow changing PINVER keys to PINGEN keys. This is not recommended. This is possible because legacy (TR-31 2005-based) implementations used the same mode 'N' for PINGEN as well as PINVER keys.							
EMV Chip / Issuer Master Keys							
DKYGENKY	DKYL0 + DMAC	EMVACMK + ANY	E0	A	N	T	Permit DKYGENKY:DKYL0+DMAC to E0
		EMVACMK + DERIVE		B,C	X	T	
	DKYL0 + DMV	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DMV to E0 0x019A
		EMVACMK + DERIVE		B,C	X	T	
	DKYL0 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E0 0x019B
		EMVACMK + DERIVE		B,C	X	T	
	DKYL1 + DMAC	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DMAC to E0
		EMVACMK + DERIVE		B,C	X	T	
	DKYL1 + DMV	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DMV to E0
		EMVACMK + DERIVE		B,C	X	T	
	DKYL1 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DALL to E0
		EMVACMK + DERIVE		B,C	X	T	
DKYGENKY	(DKYL0 + DDATA)	EMVSCMK + ANY	E1	A	N	T	Permit DKYGENKY:DKYL0+DDATA to E1
		EMVSCMK + DERIVE		B,C	X	T	

Table 125. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
	(DKYL0 + DMPIN)	EMVSCMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DMPIN to E1
		EMVSCMK + DERIVE		B,C	X	T	
	DKYL0 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E1
		EMVACMK + DERIVE		B,C	X	T	
	(DKYL1 + DDATA)	EMVSCMK + ANY		A	N		Permit DKYGENKY:DKYL1+DDATA to E1
		EMVSCMK + DERIVE		B,C	X		
	(DKYL1 + DMPIN)	EMVSCMK + ANY		A	N		Permit DKYGENKY:DKYL1+DMPIN to E1
		EMVSCMK + DERIVE		B,C	X		
	DKYL1 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DALL to E1
		EMVACMK + DERIVE		B,C	X	T	
DKYGENKY	DKYL0 + DMAC	EMVSIMK + ANY	E2	A	N	T	Permit DKYGENKY:DKYL0+DMAC to E2
		EMVSIMK + DERIVE		B,C	X	T	
	DKYL0 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E2
		EMVACMK + DERIVE		B,C	X	T	
	DKYL1 + DMAC	EMVSIMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DMAC to E2
		EMVSIMK + DERIVE		B,C	X	T	
	DKYL1 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DALL to E2
		EMVACMK + DERIVE		B,C	X	T	

Table 125. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
DATA	(none)	EMVDAMK + ANY	E3	A	N	T	Permit DATA/MAC/CIPHER/ ENCIPHER to E3
		EMVDAMK + DERIVE		B,C	X		
MAC (not MACVER)	(none)	EMVDAMK + ANY		A	N		
		EMVDAMK + MACGEN		A	G		
		EMVDAMK + DERIVE		B,C	X		
CIPHER	(none)	EMVDAMK + ANY		A	N		
		EMVDAMK + DERIVE		B,C	X		
ENCIPHER		EMVDAMK + ENC-ONLY		A	E		
		EMVDAMK + DERIVE		B,C	X		
DKYGENKY	DKYL0 +DDATA	EMVDNMK + ANY	E4	A	N	T	Permit DKYGENKY:DKYL0+DDATA to E4
		EMVDNMK + DERIVE		B,C	X		
	DKYL0 +DALL	EMVDNMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E4
		EMVDNMK + DERIVE		B,C	X		
DKYGENKY	DKYL0 + DEXP	EMVCPMK + ANY	E5	A	N	T	Permit DKYGENKY:DKYL0+DEXP to E5
		EMVCPMK + DERIVE		B,C	X		
	DKYL0 + DMAC	EMVCPMK + ANY		A	N		Permit DKYGENKY:DKYL0+DMAC to E5
		EMVCPMK + DERIVE		B,C	X		
	DKYL0 +DDATA	EMVCPMK + ANY		A	N		Permit DKYGENKY:DKYL0+DDATA to E5
		EMVCPMK + DERIVE		B,C	X		
	DKYL0 +DALL	EMVDNMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E5
		EMVDNMK + DERIVE		B,C	X		

Note: EMV Chip Card Master Keys are used by the chip cards to perform cryptographic operations, or in some cases to deriver keys used to perform operations. In CCA, these are:

- Key Gen Keys of level DKYL0 or DYKL1 allowing derivation of operational keys, or
- operational keys.

EMV support in CCA is significantly different from TR-31. CCA key types do not match TR-31 types.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 126. TR-31 export required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	TR-31 key support requires the Sep. 2011 or later LIC.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

TR-31 Import (CSNBT31I and CSNET31I)

Use the TR-31 Import callable service to convert 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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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, 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 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 129 on page 345 for valid combinations of Usage and Mode.

Table 127. Keywords for TR-31 Import Rule Array Control Information

Keyword	Meaning
Key Wrapping Method (One Required)	
INTERNAL	Desired *output_key_identifier* is a CCA internal key token, wrapped using the card master key.
EXTERNAL	Desired *output_key_identifier* is a CCA external key token, wrapped using the key represented by the *unwrap_kek_identifier*.
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, 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.)	
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)	
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)	

Table 127. Keywords for TR-31 Import Rule Array Control Information (continued)

Keyword	Meaning
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.
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.
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAP-ECB	Specifies that the original wrapping method is to be used.
Translation Control (One Optional)	

Table 127. Keywords for TR-31 Import Rule Array Control Information (continued)

Keyword	Meaning
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. **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.

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. The value in this parameter must currently be 64, since only CCA internal key tokens are supported for the *unwrap_kek_identifier* parameter.

unwrap_kek_identifier

Direction	Type
Input/Output	String

This parameter contains either the label or the key token for the key that is used to unwrap and check integrity of the imported key passed in the

TR31_key_block parameter. The key must be a CCA internal token for a KEK IMPORTER or IKEYXLAT type. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped 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, or will have ,'E' exportability. This is because ECB-mode does not comply with ANSI X9.24 Part 1.

wrap_kek_identifier_length

Direction	Type
Input	Integer

This parameter specifies the length of the *wrap_kek_identifier* parameter, in bytes. If the *unwrap_kek_identifier* is also to be used to wrap the output CCA token, specify 0 for this parameter. Otherwise, this parameter must be 64.

wrap_kek_identifier

Direction	Type
Input/Output	String

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. Otherwise, this parameter contains either the label or the key token for the KEK to use for wrapping the output CCA token. It must be a CCA internal token for a KEK EXPORTER or OKEYXLAT type and must have the same clear key as the *unwrap_kek_identifier*. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped 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. This is because ECB-mode does not comply with ANSI X9.24 Part 1.

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.

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 callable service 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’.

Any value other than these are reserved for future use and are currently invalid.

Restrictions

This callable service only imports DES and TDES keys.

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 1119). 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 128. Export attributes of an imported CCA token

TR-31 export 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 export) 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 below 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 export) 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 '00007D000300000000000000000000000' for a single length key or '00007D000341000000000000000000000007D0003210000000000000000000' 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 Export 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 Export 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.

Access control points

The access control points in the domain role that control the general function of this service are:

- TR31 Import – Permit version A TR-31 key blocks
- TR31 Import – Permit version B TR-31 key blocks
- TR31 Import – Permit version C TR-31 key blocks

When the WRAP-ECB or WRAP-ENH keywords are specified and the default key-wrapping method setting does not match the keyword, 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.

The following table lists the valid attribute translations for import of TR-31 key blocks to CCA keys 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 import a TR-31 key block to a CCA key, the appropriate key block version ACP needs to be enabled in addition to any required translation specific ACPs from below.

Table 129. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg’m	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
DUKPT Base Derivation Keys							

Table 129. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

<table>
<tr><th>Import T31 Usage</th><th>T31 Key Blk Vers.</th><th>T31 Mode</th><th>T31 Alg'm</th><th>Keywords</th><th>Output CCA Type (CSNBCVG keywords)</th><th>Output CCA Usage (CSNBCVG keywords)</th><th>Required TR31 Import ACP</th></tr>
<tr><td>B0</td><td>A</td><td>N</td><td>T</td><td>(none)</td><td>KEYGENKY</td><td>UKPT</td><td rowspan="3">(none)</td></tr>
<tr><td>B0</td><td>B,C</td><td>X</td><td>T</td><td>(none)</td><td>KEYGENKY</td><td>UKPT</td></tr>
<tr><td>B1</td><td>B,C</td><td>(none)</td><td>(none)</td><td>(none)</td><td>(none)</td><td>(none)</td></tr>
<tr><td colspan="8">Note: These are the base keys from which DUKPT initial keys are derived for individual devices such as PIN pads.</td></tr>
<tr><td colspan="8">Card Verification Keys</td></tr>
<tr><td>C0</td><td>A,B,C</td><td>G, C</td><td>D</td><td>CVK-CSC</td><td>MAC</td><td>AMEX-CSC</td><td rowspan="4">Permit C0 to MAC/MACVER:AMEX-CSC</td></tr>
<tr><td></td><td>A,B,C</td><td></td><td>T</td><td>CVK-CSC</td><td>MAC</td><td>AMEX-CSC</td></tr>
<tr><td></td><td>A,B,C</td><td>V</td><td>D</td><td>CVK-CSC</td><td>MACVER</td><td>AMEX-CSC</td></tr>
<tr><td></td><td>A,B,C</td><td></td><td>T</td><td>CVK-CSC</td><td>MACVER</td><td>AMEX-CSC</td></tr>
<tr><td></td><td>A,B,C</td><td>G, C</td><td>T</td><td>CVK-CVV</td><td>MAC</td><td>CVVKEY-A</td><td rowspan="2">Permit C0 to MAC/MACVER:CVVKEY-A</td></tr>
<tr><td></td><td>A,B,C</td><td>V</td><td>T</td><td>CVK-CVV</td><td>MACVER</td><td>CVVKEY-A</td></tr>
<tr><td colspan="8">The card verification keys are keys for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2 and CVV2 algorithms.
Note:
1. In CCA, this corresponds to keys used with two different APIs.
• Visa CVV and MasterCard CVC codes are computed with CVV_Generate and verified with CVV_Verify. Keys must be DATA or MAC with sub-type (in bits 0-3) “ANY-MAC” , “CVVKEY-A” or “CVVKEY-B”. The GEN bit (20) or VER bit (21) must be set appropriately.
• American Express CSC codes are generated and verified with the Transaction_Validate verb. The key must be a MAC or MACVER key with sub-type “ANY-MAC” or “AMEX-CSC”. The GEN bit (20) or VER bit (21) must be set appropriately.
2. CCA and TR-31 represent CVV keys incompatibly. CCA represents the “A” and “B” keys as two 8 B keys, while TR-31 represents these as one 16 B key. The CVV generate and verify verbs now accept a 16 B CVV key, using left and right parts as A and B. Current Visa standards require this.
3. Import and export of the 8 B CVVKEY-A and CVVKEY-B types will only be allowed using the proprietary TR-31 usage+mode values to indicate encapsulation of the IBM CV in an optional block, since the 8 B CVVKEY-A is meaningless / useless as a TR-31 C0 usage key of any mode.
4. Import of a TR-31 key of usage C0 to CCA key type ‘ANY-MAC’ will not be allowed, although the ANY-MAC key is also usable for card verification purposes.
5. It is possible to convert a CCA CVV key into a CSC key or vice-versa, since the translation from TR-31 usage “C0” is controlled by rule array keywords on the import verb. This can be restricted by using ACPs, but if both of translation types are required they cannot be disabled and control is up to the development, deployment, and execution of the applications themselves.
CCA does not have a ‘MAC GEN ONLY’ key type, so TR-31 usage of G will translate to a full MAC key.</td></tr>
<tr><td colspan="8">Data Encryption Keys</td></tr>
<tr><td>D0</td><td>A,B,C</td><td>E</td><td>D, T</td><td>(none)</td><td>ENCIPHER</td><td>(none)</td><td rowspan="3">(none)</td></tr>
<tr><td></td><td>A,B,C</td><td>D</td><td>D, T</td><td>(none)</td><td>DECIPHER</td><td>(none)</td></tr>
<tr><td></td><td>A,B,C</td><td>B</td><td>D, T</td><td>(none)</td><td>CIPHER</td><td>(none)</td></tr>
</table>

Table 129. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
Note: 1. There is asymmetry in the TR-31 to CCA and CCA to TR-31 translation. CCA keys can be exported to TR-31 'D0' keys from CCA type ENCIPHER, DECIPHER, or CIPHER, or type DATA with proper Encipher and Decipher CV bits on. A TR-31 'D0' key can only be imported to CCA types ENCIPHER, DECIPHER, or CIPHER, not the lower security DATA key type. This eliminates conversion to the lower security DATA type by export / re-import. 2. There are no ACPs controlling import since the intent of the TR-31 key's control is not interpreted, just directly translated to CCA control.							
Key Encrypting Keys							
K0	A,B,C	E	T	OKEYXLAT	OKEYXLAT	(none)	Permit K0:E to EXPORTER/OKEYXLAT
	A,B,C			EXPORTER	EXPORTER	(none)	
	A,B,C	D	T	IKEYXLAT	IKEYXLAT	(none)	Permit K0:D to IMPORTER/IKEYXLAT
	A,B,C			IMPORTER	IMPORTER	(none)	
	A,B,C	B	T	OKEYXLAT	OKEYXLAT	(none)	Permit K0:B to EXPORTER/OKEYXLAT
	A,B,C			EXPORTER	EXPORTER	(none)	
	A,B,C			IKEYXLAT	IKEYXLAT	(none)	Permit K0:B to IMPORTER/IKEYXLAT
	A,B,C			IMPORTER	IMPORTER	(none)	
K1	B,C	E	T	OKEYXLAT	OKEYXLAT	(none)	Permit K1:E to EXPORTER/OKEYXLAT
	B,C			EXPORTER	EXPORTER	(none)	
	B,C	D	T	IKEYXLAT	IKEYXLAT	(none)	Permit K1:D to IMPORTER/IKEYXLAT
	B,C			IMPORTER	IMPORTER	(none)	
	B,C	B	T	OKEYXLAT	OKEYXLAT	(none)	Permit K1:B to EXPORTER/OKEYXLAT
	B,C			EXPORTER	EXPORTER	(none)	
	B,C			IKEYXLAT	IKEYXLAT	(none)	Permit K1:B to IMPORTER/IKEYXLAT
	B,C			IMPORTER	IMPORTER	(none)	
Note: 1. K1' keys are not distinguished from 'K0' keys within CCA. The 'K1' key is a particular KEK for deriving keys used in the 'B' or 'C' version wrapping of TR-31 key blocks. CCA does not distinguish between targeted protocols currently and so there is no good way to represent the difference; also note that most wrapping mechanisms now involve derivation or key variation steps. 2. It is possible to convert a CCA EXPORTER key to an OKEYXLAT, or to convert an IMPORTER to an IKEYXLAT by export / re-import. This can be restricted by using ACPs, but if both translations are required they cannot be disabled and control is up to the development, deployment, and execution of the applications themselves. 3. It will not be possible to export a CCA key to TR-31 type K0-B, in order to avoid the ability to translate a CCA EXPORTER to a CCA IMPORTER via export/import to the TR-31 token type. When a TR-31 key block does not have an included CV as an optional block, the default CV will be used to construct the output token. For IMPORTER / EXPORTER keys this means that the Key Generate bits will also be on in the KEK.							
MAC Keys							

Table 129. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
M0	A,B,C	G,C	T	(none)	MAC	ANY-MAC	Permit M0/M1/M3 to MAC/MACVER:ANY-MAC
	A,B,C	V	T	(none)	MACVER	ANY-MAC	
M1	A,B,C	G,C	D, T	(none)	MAC	ANY-MAC	
	A,B,C	V	D, T	(none)	MACVER	ANY-MAC	
M3	A,B,C	G,C	D, T	(none)	MAC	ANY-MAC	
	A,B,C	V	D, T	(none)	MACVER	ANY-MAC	
Note: 1. M0 and M1 are identical (ISO 16609 based on ISO 9797) normal DES/TDES (CBC) MAC computation, except M1 allows 8 byte and 16 byte keys while M0 allows only 16 byte keys. Mode M3 is the X9.19 style triple-DES MAC. 2. CCA does not support M2, M4, or M5. 3. Although export of DATAM/DATAMV keys to TR-31 M0/M1/M3 key types is allowed, import to DATAM/DATAMV CCA types is not allowed since they are obsolete types							
PIN Keys							
P0	A,B,C	E	T	(none)	OPINENC	(none)	Permit P0:E to OPINENC
	A,B,C	D		(none)	IPINENC	(none)	Permit P0:D to IPINENC
	A,B,C	B – not supp		(none)	(none)	(none)	(none)
V0	A	N	T	PINGEN [NOOFFSET]	PINGEN	NO-SPEC [+NOOFFSET]	Permit V0 to PINGEN:NO-SPEC, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	G,C		[NOOFFSET]	PINGEN	NO-SPEC [+NOOFFSET]	Permit V0 to PINGEN:NO-SPEC
	A	N		PINVER [NOOFFSET]	PINVER	NO-SPEC [+NOOFFSET]	Permit V0 to PINVER:NO-SPEC, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	V		[NOOFFSET]	PINVER	NO-SPEC [+NOOFFSET]	Permit V0 to PINVER:NO-SPEC
V1	A	N	T	PINGEN [NOOFFSET]	PINGEN	IBM-PIN /IBM-PINO	Permit V1 to PINGEN:IBM-PIN/IBM-PINO, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	G,C		[NOOFFSET]	PINGEN	IBM-PIN /IBM-PINO	Permit V1 to PINGEN:IBM-PIN/IBM-PINO
	A	N		PINVER [NOOFFSET]	PINVER	IBM-PIN /IBM-PINO	Permit V1 to PINVER:IBM-PIN/IBM-PINO, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	V		[NOOFFSET]	PINVER	IBM-PIN /IBM-PINO	Permit V1 to PINVER:IBM-PIN/IBM-PINO
V2	A	N	T	PINGEN	PINGEN	VISA-PVV	Permit V2 to PINGEN:VISA-PVV, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	G,C			PINGEN	VISA-PVV	Permit V2 to PINGEN:VISA-PVV

Table 129. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
	A	N		PINVER	PINVER	VISA-PVV	Permit V2 to PINVER:VISA-PVV, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	V			PINVER	VISA-PVV	Permit V2 to PINVER:VISA-PVV

Note:

1. NOOFFSET keyword may be passed to specify resultant CCA key to have NOOFFSET bit (bit 37) on in CV. However this will be automatic if CV is included and has NOOFFSET bit set.
2. NOOFFSET keyword is not supported for V2 usage since VISA-PVV algorithm does not support that concept.
3. There is a subtle difference between TR-31 V0 mode and CCA 'NO-SPEC' subtype. V0 mode restricts keys from 3224 or PVV methods, while CCA 'NO-SPEC' allows any method.
4. Turning on the ACP or ACPs controlling export of PINVER to usage:mode V*:N and import of V*:N to PINGEN at the same time will allow changing PINVER keys to PINGEN keys. This is not recommended. This is possible because legacy (TR-31 2005-based) implementations used the same mode 'N' for PINGEN as well as PINVER keys.

EMV Chip / Issuer Master Keys

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
E0	A	N	T	DKYL0 +DMAC	DKYGENKY	DKYL0 +DMAC	Permit E0 to DKYGENKY:DKYL0+DMAC
	B,C	X		DKYL0 +DMAC		DKYL0 +DMAC	
	A	N		DKYL0 +DMV		DKYL0 +DMV	Permit E0 to DKYGENKY:DKYL0+DMV
	B,C	X		DKYL0 +DMV		DKYL0 +DMV	
	A	N		DKYL1 +DMAC		DKYL1 +DMAC	Permit E0 to DKYGENKY:DKYL1+DMAC
	B,C	X		DKYL1 +DMAC		DKYL1 +DMAC	
	A	N		DKYL1 +DMV		DKYL1 +DMV	Permit E0 to DKYGENKY:DKYL1+DMV
	B,C	X		DKYL1 +DMV		DKYL1 +DMV	
E1	A	N, E, D, B	T	DKYL0 +DMPIN	DKYGENKY	DKYL0 +DMPIN	Permit E1 to DKYGENKY:DKYL0+DMPIN
	B,C	X		DKYL0 +DMPIN		DKYL0 +DMPIN	
	A	N, E, D, B		DKYL0 +DDATA		DKYL0 +DDATA	Permit E1 to DKYGENKY:DKYL0+DDATA
	B,C	X		DKYL0 +DDATA		DKYL0 +DDATA	
	A	N, E, D, B		DKYL1 +DMPIN		DKYL1 +DMPIN	Permit E1 to DKYGENKY:DKYL1+DMPIN
	B,C	X		DKYL1 +DMPIN		DKYL1 +DMPIN	

Table 129. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
	A	N, E, D, B		DKYL1 +DDATA		DKYL1 +DDATA	Permit E1 to DKYGENKY:DKYL1+DDATA
	B,C	X		DKYL1 +DDATA		DKYL1 +DDATA	
E2	A	N	T	DKYL0 +DMAC	DKYGENKY	DKYL0 +DMAC	Permit E2 to DKYGENKY:DKYL0+DMAC
	B,C	X		DKYL0 +DMAC		DKYL0 +DMAC	
	A	N		DKYL1 +DMAC		DKYL1 +DMAC	Permit E2 to DKYGENKY:DKYL1+DMAC
	B,C	X		DKYL1 +DMAC		DKYL1 +DMAC	
E3	A	N, E, D, B, G	T	(none)	ENCIPHER	(none)	Permit E3 to ENCIPHER
	B,C	X		(none)		(none)	
E4	A	N, B	T	(none)	DKYGENKY	DKYL0 +DDATA	Permit E4 to DKYGENKY:DKYL0+DDATA
	B,C	X		(none)		DKYL0 +DDATA	
E5	A	G, C, V, E, D, B, N	T	DKYL0 +DMAC	DKYGENKY	DKYL0 +DMAC	Permit E5 to DKYGENKY:DKYL0+DMAC
	B,C	X		DKYL0 +DMAC		DKYL0 +DMAC	
	A	G, C, V, E, D, B, N		DKYL0 +DDATA		DKYL0 +DDATA	Permit E5 to DKYGENKY:DKYL0+DDATA
	B,C	X		DKYL0 +DDATA		DKYL0 +DDATA	
	A	G, C, V, E, D, B, N		DKYL0 +DEXP		DKYL0 +DEXP	Permit E5 to DKYGENKY:DKYL0+DEXP
	B,C	X		DKYL0 +DEXP		DKYL0 +DEXP	

Note: EMV Chip Card Master 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:

- Key Gen Keys of level DKYL0 or DKYL1 allowing derivation of operational keys, or
- operational keys.

EMV support in CCA is significantly different. CCA key types do not match TR-31 types.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 130. TR-31 Import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	TR-31 key support requires the Sept. 2011 or later LIC.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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

```
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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 Export 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 1119). 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 Export. If the TR-31 Export 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

```
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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 131. 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.

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 n*um_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 four times the number of optional blocks and the application program must supply a buffer with a length at least four 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 1119). 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 step above.
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 and reason codes,” on page 943 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 and reason codes," on page 943 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.

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. CCA only supports "D" for a Single-DES key and "T" for a Triple-DES key.

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 1119). 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 step above.
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.

Unique Key Derive (CSNBUKD and CSNEUKD)

Unique Key Derive (CSNBUKD and CSNEUKD) will perform the key derivation process as defined in ANSI X9.24 Part 1.

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. This must be a KEYGENKY with the UKPT bit (bit 18) set to 1 in the Control Vector.
- 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, 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 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, either enhanced wrapping mode (WRAP-ENH) or original ECB wrapping mode (WRAP-ECB).

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 and reason codes," on page 943 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 and reason codes," on page 943 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 5.

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 132. Keywords for Unique Key Derive

Keyword	Meaning
Algorithm (One, optional. The default is DES)	
DES	Specifies that the keys to be generated are DES (Triple DES) keys. All input skeleton tokens must be DES tokens and all generated output tokens will be DES tokens.
Token output type (One, required for K3IPEK)	
TDES-TOK	Specifies that the output IPEK should be wrapped by the TDES transport key and returned in an external TDES token.

Table 132. Keywords for Unique Key Derive (continued)

Keyword	Meaning
TR31-TOK	Specifies that the output IPEK should be wrapped by the TDES transport key and returned in a TR-31 key block.
Key wrapping method (One, optional. The default is USECONFG. The **Access Control Point Unique Key Derive – Override Default Wrapping Method** must be enabled to specify these keywords.	
USECONFG	Specifies to wrap the key using the configuration setting for the default wrapping method.
WRAP-ECB	Specifies to wrap the key using the original wrapping method.
WRAP-ENH	Specifies to wrap the key using the enhanced wrapping method.
Output Key Selection Keywords. (One required, up to 3 can be specified.) Neither the PIN-DATA nor the K3IPEK keyword can be specified with any other Output Key Selection keywords. Any combination of the other keywords (K1DATA, K2MAC, and K3PIN) can be specified, enabling a program to produce up to 3 different output keys with one call.	
PIN-DATA	The returned key type for this keyword is a PIN key, which is returned in a DATA key token. This is an output key selection keyword for the *generated_key_identifier3_length* and *generated_key_identifier3* parameters. 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 133 on page 368. 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.
K1DATA	The returned key type for this keyword is a DATA ENCRYPTION key. This is the output key selection keyword for the *generated_key_identifier1_length* and *generated_key_identifier1* parameters. 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 133 on page 368.
K2MAC	The returned key type for this keyword is a MAC key. This is the output key selection keyword for the *generated_key_identifier2_length* and *generated_key_identifier2* parameters. 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 133 on page 368.

Table 132. Keywords for Unique Key Derive (continued)

Keyword	Meaning
K3PIN	The returned key type for this keyword is a PIN key. This is an output key selection keyword for the *generated_key_identifier3_length* and *generated_key_identifier3* parameters. 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 133 on page 368.
K3IPEK	The returned key for this keyword is the IPEK. This is an output key selection keyword for the *generated_key_identifier3_length* and *generated_key_identifier3* parameters. Output value *generated_key_identifier3* will be created and will be the initial PIN encryption key wrapped by the TDES transport key and returned in an external symmetric token or TR-31 key block as indicated by the token output type keyword. 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 133 on page 368. This keyword may not be combined with any other output key selection keyword.

base_derivation_key_identifier_length

Direction	Type
Input	Integer

The length of the base_derivation_key parameter. This value must be 64

base_derivation_key_identifier

Direction	Type
Input/Output	String

The base derivation key is the key from which the operational keys are derived using the DUKPT algorithms defined in ANSI X9.24 Part 1. The base derivation key must be an internal key token or the label of an internal key token containing a double-length KEYGENKY key with the UKPT bit (bit 18) set to 1 in the Control Vector.

derivation_data_length

Direction	Type
Input	Integer

The length of the *derivation_data* parameter. This value must be 10.

derivation_data

Direction	Type
Input	String

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.

generated_key_identifier1_length

Direction	Type
Input/Output	Integer

The length of the *generated_key_identifier1* parameter. Values are 0 and 64.

generated_key_identifier1

Direction	Type
Input/Output	String

On input, this must be a DES Data encryption key token or a skeleton token of a DES Data encryption key, with one of the Data encryption control vectors as shown in Table 133 on page 368.

On output, *generated_key_identifier1* will contain the data encryption token with the derived data encryption key.

generated_key_identifier2_length

Direction	Type
Input/Output	Integer

The length of the *generated_key_identifier2* parameter. Values are 0 and 64.

generated_key_identifier2

Direction	Type
Input/Output	String

On input, this must be a DES MAC key token or a skeleton token of a DES MAC key, with one of the MAC control vectors as shown in Table 133 on page 368. On output, *generated_key_identifier2* will contain the MAC token with the derived MAC key.

generated_key_identifier3_length

Direction	Type
Input/Output	Integer

The length of the *generated_key_identifier3* parameter. When the rule array keyword is K3IPEK, the length must be at least 64 bytes. Otherwise, values are 0 and 64.

generation_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 *generation_key_identifier3* parameter can be either PIN-DATA or K3PIN.

- 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 133 on page 368. On output, this parameter will contain the Data token with the derived PIN key.

- When Rule Array Keyword is K3PIN, input must be 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 133 on page 368. On output, this parameter will contain the PIN token with the derived PIN key.
- When Rule Array Keyword is K3IPEK, input must be a DES PIN key token or a skeleton token of a DES PIN key left-justified in the field, with one of the PIN control vectors as shown in Table 133 on page 368. On output, this parameter will contain the TDES wrapped IPEK in an external symmetric key token or TR-31 key block.

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. Otherwise, the length must be 64.

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. The transport key must be a DES EXPORTER KEK. 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 133. 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 (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 following table indicates the variants used for each output key type to be derived.

Table 134. Derivation Variants

Key Type	DUKPT derivation variant	DUKPT key usage description
IPINENC OPINENC PIN key (using PIN-DATA rule array keyword)	00000000000000FF 00000000000000FF	PIN Encryption
MAC	000000000000FF00 000000000000FF00	MAC, request or both ways
MACVER	00000000FF000000 00000000FF000000	MAC, response only
CIPHER ENCIPHER	0000000000FF0000 0000000000FF0000	Data Encryption, request or both ways
DECIPHER	000000FF00000000 000000FF00000000	Data Encryption, response only

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 Method** 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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 135. Unique Key Derive required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114		This callable service is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	The K3IPEK, TDES-TOK, and TR31-TOK keywords require the Sep. 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

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:

- "Ciphertext Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3)" on page 374
- "Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)" on page 386
- "Decode (CSNBDCO and CSNEDCO)" on page 393
- "Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)" on page 395
- "Encode (CSNBECO and CSNEECO)" on page 402
- "Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)" on page 404
- "Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)" on page 411
- "Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)" on page 417
- "Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)" on page 428

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 1104 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, 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 DATA 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 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.

Ciphertext 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 Ciphertext Translate2 Callable Service" on page 68 provides some tips on using the callable service.

Use the ciphertext 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 and reason codes," on page 943 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 and reason codes,” on page 943 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 136. Keywords for Ciphertext 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.
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.

Table 136. Keywords for Ciphertext Translate2 (continued)

Keyword	Meaning
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)	
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.

Table 136. Keywords for Ciphertext Translate2 (continued)

Keyword	Meaning
IKEY-AES	Specifies that the inbound key identifier is an AES key.
Outbound Key Identified (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. When the key identifier is a key token, the value is the length of the token. The maximum value is 725.

key_identifier_in

Direction	Type
Input/Output	String

An internal key token or the label of the CKDS record containing the cipher translation key for the inbound ciphertext.

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-, double-, 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.

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. When the key identifier is a key token, the value is the length of the token. The maximum value is 725.

key_identifier_out

Direction	Type
Input/Output	String

An internal key token or the label of the CKDS record containing the cipher translation key for the outbound ciphertext.

Acceptable DES key types are DATA, CIPHER, CIPHERXL, CIPHERXO, and ENCIPHER. 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 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.

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 137. 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.

Table 137. 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	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.
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.

Table 137. 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	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.
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.

Table 137. 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 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 138. Ciphertext translate2 key usage

key_identifier_in (DEC bit except DATA and AESDATA)	key_identifier_out (ENC bit except DATA and AESDATA)
DATA CIPHER DECIPHER CIPHERXI CIPHERXL	DATA CIPHER ENCIPHER CIPHERXO CIPHERXL AESDATA vCIPHER vCIPHERe vCIPHERex vCIPHERedx
AESDATA vCIPHER vCIPHERd vCIPHERdx vCIPHERdex	DATA (must be at least double-length key with ACP) CIPHER (requires ACP to be enabled) ENCIPHER (requires ACP to be enabled) CIPHERXO (requires ACP to be enabled) CIPHERXL (requires ACP to be enabled) AESDATA vCIPHER vCIPHERe vCIPHERex vCIPHERedx

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 **Ciphertext translate2 – Allow translate to weaker DES** access control point to be enabled.
3. When the **Ciphertext 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 139. Ciphertext translate2 access control points

Access control point	Description
Cipher Text translate2	Enable Ciphertext 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 139. Ciphertext 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.

Table 140. Ciphertext translate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114		This callable service is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing a data-encrypting key, to be used for deciphering the data. If the key token or key label contains a single-length key, single-DES decryption is performed. If the key token or key label contains a double-length or triple-length key, triple-DES decryption is performed.

Single and double length CIPHER and DECIPHER keys are also supported.

text_length

Direction	Type
Input/Output	Integer

On entry, you supply the length of the ciphertext. The maximum length of text is 214783647 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 141. 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 141. 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	
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.

Table 141. Keywords for the Decipher Rule Array Control Information (continued)

Keyword	Meaning
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 1104 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 1104 discusses the cipher processing rules.

The Encipher callable services are described under “Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)” on page 395.

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.

Table 142. Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	If the token is marked as CDMF, the service fails.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	If the token is marked as CDMF, the service fails.

Table 142. Decipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	If the token is marked as CDMF, the service fails.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	If the token is marked as CDMF, the service fails.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	If the token is marked as CDMF, the service fails.
IBM z13	Crypto Express5 CCA Coprocessor	If the token is marked as CDMF, the service fails.

Decode (CSNBDCO and CSNEDCO)

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 clear key import service, and get back a token that will allow you to use the encipher and decipher callable services.
- Consider using the Symmetric Key Decipher service ("Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)" on page 417).

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 and reason codes," on page 943 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 and reason codes," on page 943 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.

Table 143. Decode required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions	
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions	
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions	
IBM z13	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 144 on page 399 and “Cipher Processing Rules” on page 1104.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing the data-encrypting key, to be used for encrypting the data. If the key token or key label contains a single-length key, single-DES encryption is performed. If the key token or key label contains a double-length or triple-length key, triple-DES encryption is performed.

On an IBM eServer zSeries 990 and subsequent releases, single and double length CIPHER and ENCIPHER keys are also supported.

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 2,14783647 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 144. 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 144. 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.

Table 144. Keywords for the Encipher Rule Array Control Information (continued)

Keyword	Meaning
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.
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 1104 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 1104 discusses the cipher processing rules.

The Decipher callable services are described under "Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)" on page 386.

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.

Table 145. Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	If the token is marked as CDMF, the service fails.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	If the token is marked as CDMF, the service fails.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	If the token is marked as CDMF, the service fails.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	If the token is marked as CDMF, the service fails.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	If the token is marked as CDMF, the service fails.
IBM z13	Crypto Express5 CCA Coprocessor	If the token is marked as CDMF, the service fails.

Encode (CSNBECO and CSNEECO)

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 clear key import service, and get back a token that will allow you to use the encipher and decipher callable services.

- Consider using the Symmetric Key Encipher service (“Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)” on page 428).

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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.

Table 146. Encode required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions	
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions	
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions	
IBM z13	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. Data is deciphered that has been enciphered in either CBC mode or ECB mode.

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 unpaded clear text data. PKCS padding is described in "PKCS Padding Method" on page 1107.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 147. Symmetric Algorithm Decipher Rule Array Keywords

Keyword	Meaning
Algorithm (one keyword, 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.
Processing Rule (optional - zero or 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).
PKCS-PAD	Deciphers with cipher block chaining and text length reduced to the original value. The ciphertext length must be an exact multiple of 16 bytes. Padding is removed from the plaintext. 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)	
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 - zero or 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.
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.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter. The length must be 64 bytes for an AES DATA Internal Key Token (version X'04') or a CKDS label, or between the actual length of the token and 725 for an AES CIPHER Internal Key Token (version X'05').

key_identifier

Direction	Type
Input	String

This specifies an internal secure AES token or the labelname of a secure AES token in the CKDS. Normal CKDS labelname syntax is required.

The AES key identifier must be an encrypted key contained in an internal key token, where the key is enciphered under the AES master key. The key can be 128-, 192-, or 256-bits in length.

key_parms_length

Direction	Type
Input	Integer

The length of the *key_parms* parameter. This must be 0.

key_parms

Direction	Type
Ignored	String

This parameter is ignored. It is reserved for future use.

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. The length should be equal to the block length for the algorithm specified. This parameter is ignored if the process rule is ECB.

initialization_vector

Direction	Type
Input	String

This parameter contains the initialization vector (IV) for CBC mode decryption, including CBC mode invoked using the PKCS-PAD keyword. This parameter is ignored if the process rule is ECB. For AES CBC mode decryption, the initialization vector length must be 16 bytes, the length of an AES block. The IV must be the same value used when the data was encrypted.

chain_data_length

Direction	Type
Input/Output	Integer

The length of the *chain_data* parameter. 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.

chain_data

Direction	Type
Input/Output	String

A buffer that is used as a work area for sequences of chained symmetric algorithm decipher requests. 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.

The exact content and layout of *chain_data* is not described. For AES CBC encryption, the field must be at least 32-bytes in length. For AES ECB encryption the field is not used and any length is acceptable including zero.

cipher_text_length

Direction	Type
Input	Integer

The length of the cipher text. The length must be a multiple of the algorithm block size.

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. On output, this parameter has the actual length of the text stored in the *clear_text* parameter.

If process rule PKCS-PAD is used, the clear text length 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. This parameter must be 0.

optional_data

Direction	Type
Ignored	String

Optional data required by a specified algorithm.

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.

Access control point

The **Symmetric Algorithm Decipher - secure AES keys** 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.

Table 148. Symmetric Algorithm Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC)
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC)
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	AES Variable-length Symmetric Internal Key Tokens require the Sep. 2011 or later licensed internal code (LIC).

Table 148. Symmetric Algorithm Decipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)

The symmetric algorithm encipher callable service enciphers data with the AES algorithm. Data is enciphered that has been deciphered in either CBC mode or ECB mode.

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 and reason codes," on page 943 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 and reason codes," on page 943 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

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 149. Symmetric Algorithm Encipher Rule Array Keywords

Keyword	Meaning
Algorithm (one keyword, required)	
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).
Processing Rule (optional - zero or 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).
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 1107. 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)	
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 - zero or one keyword)	

Table 149. Symmetric Algorithm Encipher Rule Array Keywords (continued)

Keyword	Meaning
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.
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.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter. The length must be 64 bytes for an AES DATA Internal Key Token (version X'04') or a CKDS label, or between the actual length of the token and 725 for an AES CIPHER Internal Key Token (version X'05').

key_identifier

Direction	Type
Input	String

This specifies an internal secure AES token or the labelname of a secure AES token in the CKDS. Normal CKDS labelname syntax is required.

The AES key identifier must be an encrypted key contained in an internal key token, where the key is enciphered under the AES master key. The key can be 128-, 192-, or 256-bits in length.

key_parms_length

Direction	Type
Input	Integer

The length of the *key_parms* parameter in bytes. It must be set to 0.

key_parms

Direction	Type
Ignored	String

This parameter is ignored. It is reserved for future use.

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. This parameter is ignored if the process rule is ECB.

initialization_vector

Direction	Type
Input	String

This parameter contains the initialization vector (IV) for CBC mode encryption, including the CBC mode invoked using the PKCS-PAD keyword. This parameter is ignored if the process rule is ECB. For AES CBC mode encryption, the initialization vector length must be 16 bytes, the length of an AES block. The same IV must be used when decrypting the data.

chain_data_length

Direction	Type
Input/Output	Integer

The length in bytes of the *chain_data* parameter. 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.

chain_data

Direction	Type
Input/Output	String

A buffer that is used as a work area for sequences of chained symmetric algorithm encipher requests. 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.

The exact content and layout of *chain_data* is not described. For AES CBC encryption, the field must be at least 32-bytes in length. For AES ECB encryption the field is not used and any length is acceptable including zero.

clear_text_length

Direction	Type
Input	Integer

The length of the clear text data in the *clear_text* parameter. Unless process rule PKCS-PAD is used, the length must be a multiple of the algorithm block size. The length must be 1 or greater.

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.

If process rule PKCS-PAD is used, 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.

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. This parameter is reserved for future use. It must be set to 0.

optional_data

Direction	Type
Ignored	String

The optional data used in processing the request. This parameter is ignored.

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.

Access control point

The **Symmetric Algorithm Encipher - secure AES keys** 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.

Table 150. Symmetric Algorithm Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	AES Variable-length Symmetric Internal Key Tokens require the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 371 for more information.

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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 151. 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.

Table 151. Symmetric Key Decipher Rule Array Keywords (continued)

Keyword	Meaning
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.
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 release 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.

Table 151. Symmetric Key Decipher Rule Array Keywords (continued)

Keyword	Meaning
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 clear keys, the length is in bytes and includes only the value of the key. The maximum size is 256 bytes.

For the KEYIDENT keyword, this parameter value must be 64.

key_identifier

Direction	Type
Input	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 and the key type must be DATA. Encrypted key support is available on IBM System z10 and later servers.

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

This 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.

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.

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. On output, this parameter has the actual length of the text stored in the *clear_text* parameter. The *clear_text* parameter must be at least the same length as the *cipher_text* parameter, except for the PKCS-PAD and X9.23 processing rules, where the padding is automatically dropped on output.

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 DES or AES encrypted DATA 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*.
- 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 listed below must be enabled.

Table 152. Required access control points for Symmetric Key Decipher

Key algorithm	Access control point
AES	Symmetric Key Encipher/Decipher - Encrypted AES keys
DES	Symmetric Key Encipher/Decipher - Encrypted DES keys

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 153. Symmetric Key Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	GCM processing rule is not supported. CFB-LCFB processing rule is supported only when *key_parms* specifies a segment size equal to the blocksize.
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	GCM processing rule is not supported. CFB-LCFB processing rule is supported only when *key_parms* specifies a segment size equal to the blocksize.
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions Crypto Express3 Coprocessor	GCM processing rule is not supported. CFB-LCFB processing rule is supported only when *key_parms* specifies a segment size equal to the blocksize. Encrypted keys require CEX3C with the Nov. 2009 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions Crypto Express3 Coprocessor	Encrypted keys require CEX3C with the Nov. 2009 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	

Related information

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 1104 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 371 for more information.

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.

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 decryption, 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 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 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,
            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 and reason codes," on page 943 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 and reason codes," on page 943 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	Integer

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 154. 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.

Table 154. Symmetric Key Encipher Rule Array Keywords (continued)

Keyword	Meaning
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 release 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.
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.

Table 154. Symmetric Key Encipher Rule Array Keywords (continued)

Keyword	Meaning
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 clear keys, the length is in bytes and includes only the value of the key.

For the KEYIDENT keyword, this parameter value must be 64.

key_identifier

Direction	Type
Input	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 and the key type must be DATA. Encrypted key support is available on IBM System z10 and later servers.

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

This 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.

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.

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 is 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 DES or AES encrypted DATA 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*.
- 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 listed below must be enabled.

Table 155. Required access control points for Symmetric Key Encipher

Key algorithm	Access control point
AES	Symmetric Key Encipher/Decipher - Encrypted AES keys
DES	Symmetric Key Encipher/Decipher - Encrypted DES keys

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 156. Symmetric Key Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	GCM processing rule is not supported. CFB-LCFB processing rule is supported only when key_parms specifies a segment size equal to the blocksize.
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	GCM processing rule is not supported. CFB-LCFB processing rule is supported only when key_parms specifies a segment size equal to the blocksize.
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions Crypto Express3 Coprocessor	GCM processing rule is not supported. CFB-LCFB processing rule is supported only when key_parms specifies a segment size equal to the blocksize. Encrypted keys require the CEX3C with the Nov. 2009 or later licensed internal code (LIC).

Table 156. Symmetric Key Encipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions Crypto Express3 Coprocessor	Encrypted keys require the CEX3C with the Nov. 2009 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	CP Assist for Cryptographic Functions Crypto Express5 CCA Coprocessor	

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 1104 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 10, “Using Digital Signatures,” on page 707) 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:

How MACs are Used

When a message is sent, an application program can generate an authentication code for it using the MAC generation, MAC generation2, or HMAC generate callable service. ICSF supports:

- The ANSI X9.9-1 basic procedure and both 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 or double-length MAC verification key, the single-length or double-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 10, "Using Digital Signatures," on page 707).

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 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 and reason codes," on page 943 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 and reason codes," on page 943 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 157. 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)	
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)	

Table 157. Keywords for HMAC Generate Control Information (continued)

Keyword	Meaning
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

The length of the *key_identifier* parameter. The maximum value is 725.

key_identifier

Direction	Type
Input	Integer

The 64-byte label or internal token of an encrypted HMAC key.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of *text* is 214783647 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. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

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.

Access control points

This table lists the access control points in the domain role that control the function for this service.

Table 158. 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.

Table 159. HMAC generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.

Table 159. HMAC generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	This service is not supported.
	Crypto Express3 Coprocessor	This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 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 and reason codes," on page 943 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 and reason codes," on page 943 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 160. 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.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

key_identifier_length

Direction	Type
Input	Integer

The length of the *key_identifier* parameter. The maximum value is 725.

key_identifier

Direction	Type
Input/Output	String

The 64-byte label or internal token of an encrypted HMAC or HMACVER key.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of *text* is 214783647 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. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

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.

Access control points

This table lists the access control points in the domain role that control the function for this service.

Table 161. 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.

Table 162. HMAC Verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	This service is not supported.
	Crypto Express3 Coprocessor	This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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. You can specify that the callable service uses either the ANSI X9.9-1 procedure or the ANSI X9.19 optional double key MAC procedure to compute the MAC. For the ANSI X9.9-1 procedure you identify either a MAC generate key or a DATA key, and the message text. For the ANSI X9.19 optional double key MAC procedure, you identify a double-length MAC key and the message text.

The MAC generate callable service also supports the padding rules specified in the EMV Specification and ISO 16609. For the EMV MAC procedure, you identify a single- or double-length MAC key and the message text. For the ISO 16609 procedure you identify a double-length MAC or DATA key and the message text.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 64-byte key label or internal key token that identifies a single or double-length MAC generate key, a DATAM key, or a single-length DATA key. The type of key depends on the MAC process rule in the *rule_array* parameter.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 214783647 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 163. 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 163. Keywords for MAC generate 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 MAC or a single-length DATA key. The text is always padded with 1 to 8 bytes 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 a double-length MAC key. The *key_identifier* parameter must identify a double-length MAC key. 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 padding rules are the same as for X9.9-1.

Table 163. Keywords for MAC generate Control Information (continued)

Keyword	Meaning
X9.9-1	ANSI X9.9-1 and X9.19 basic procedure. The *key_identifier* parameter must identify a single-length MAC or a single-length DATA key. 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 must identify a double-length MAC or a double-length DATA key. The text is padded only if the text length is not a multiple of 8 bytes.
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.
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 18-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

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

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 164. MAC generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	TDES-MAC not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 verification callable service is described in "MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)" on page 460.

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. A MAC key with key usage that can be used for generate is required to calculate the MAC.

The MAC generate key must be in a variable-length HMAC key token for HMAC and an AES MAC token for CMAC.

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 165. 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 725.

key_identifier

Direction	Type
Input/Output	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.

For the HMAC algorithm, the key algorithm must be HMAC and the key usage fields must indicate GENONLY or GENERATE and the hash method selected. For the AES algorithm, the key algorithm must be AES, the key type must be MAC, and the key usage fields must indicate GENONLY or GENERATE and must indicate CMAC.

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 supplied in the *text* parameter. The maximum length of *text* is 214783647 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. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

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

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 166. 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.

Table 167. MAC Generate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Requires the March 2014 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Requires the March 2014 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

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. You can specify that the callable service uses either the ANSI X9.9-1 procedure or the ANSI X9.19 optional double key MAC procedure to compute the MAC. For the ANSI X9.9-1 procedure you identify either a MAC verify key, a MAC generation key, or a DATA key, and the message text. For the ANSI X9.19 optional double key MAC procedure, you identify either a double-length MAC verify key or a double-length MAC generation key and the message text. The cryptographic feature compares the generated MAC with the one sent with the message. A return code indicates whether the MACs are the same. If the MACs are the same, the receiver knows the message was not altered. The generated MAC never appears in storage is not revealed outside the cryptographic feature.

The MAC verify callable service also supports the padding rules specified in the EMV Specification and ISO 16609. For the EMV MAC procedure, you identify a single- or double-length MAC key and the message text. For the ISO 16609 procedure you identify a double-length MAC or DATA key and the message text.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 64-byte key label or internal key token that identifies a single or double-length MAC verify key, a single or double-length MAC verify key, a single or double length MAC generation key, a DATAM or DATAMV key, or a single-length DATA key. The type of key depends on the MAC process rule in the *rule_array* parameter.

text_length

Direction	Type
Input	Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 214783647 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 168. 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 168. 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 MAC, MACVER, or DATA key. The text is always padded with 1 to 8 bytes 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 a double-length MAC key. The *key_identifier* parameter must identify a double-length MAC or MACVER key. The padding rules are the same as for EMVMAC.
X9.19OPT	ANSI X9.9-1 and X9.19 basic procedure. The *key_identifier* parameter must identify a single-length MAC, MACVER, or DATA key. 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.

Table 168. Keywords for MAC verify Control Information (continued)

Keyword	Meaning
X9.9-1	ANSI X9.9-1 and X9.19 basic procedure. The *key_identifier* parameter must identify a single-length MAC, or single-length DATA key. 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* must identify a double-length MAC or a double-length DATA key. The text is padded only if the text length is not a multiple of 8 bytes.
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; 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 18-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter to binary zeros.

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

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 169. MAC verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	TDES-MAC not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 generation callable service is described in "MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)" on page 451.

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. A MAC key with key usage that can be used for verify is required to verify the MAC.

The MAC verify key must be in a variable-length HMAC key token for HMAC and an AES MAC token for CMAC.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 170. 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 725.

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 the HMAC algorithm, the key algorithm must be HMAC and the key usage fields must indicate GENERATE or VERIFY and the hash method selected. For the AES algorithm, the key algorithm must be AES, the key type must be MAC, and the key usage fields must indicate GENERATE or VERIFY and must indicate CMAC.

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 supplied in the *text* parameter. The maximum length of *text* is 214783647 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. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

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 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

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 171. 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.

Table 172. MAC Verify2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Requires the March 2014 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Requires the March 2014 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 214783647 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 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 173. 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 173. 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 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.

Table 173. Keywords for MDC Generate Control Information (continued)

Keyword	Meaning
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 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 18-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

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

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 174. MDC generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions	
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions	
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions	
IBM z13	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 on specified text. This service supports the following methods:

- MD5 - software only
- SHA-1
- RIPEMD-160 - software only
- SHA-224
- SHA-256
- SHA-384
- SHA-512

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 and reason codes," on page 943 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 and reason codes," on page 943 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, 2 or 3.

`rule_array`

Direction	Type
Input	String

Keywords that provide control information to the callable service are listed in Table 175. 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 175. 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. Length of hash generated is 16 bytes.
MD5-LG	Hash algorithm is similar to the MD5 algorithm. Use this hash method for PKCS-1.0 and PKCS-1.1. Length of hash generated is 16 bytes. 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. Length of hash generated is 20 bytes. 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. Length of hash generated is 20 bytes.
SHA-1	Hash algorithm is SHA-1 algorithm. Use this hash method for DSS. Length of hash generated is 20 bytes.
SHA-224	Hash algorithm is SHA-256 algorithm. Length of hash generated is 28 bytes.
SHA-256	Hash algorithm is SHA-256 algorithm. Length of hash generated is 32 bytes.
SHA-384	Hash algorithm is SHA-384 algorithm. Length of hash generated is 48 bytes.
SHA-512	Hash algorithm is SHA-512 algorithm. 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.

Table 175. 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). 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.
Chaining Flag (optional)	
FIRST	Specifies this is the first call in a series of chained calls. 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. 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.

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. This must be 128 bytes.

`chaining_vector`

Direction	Type
Input/Output	String

This field is a 128-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 supplied *hash* field in bytes.

Note: For SHA-1 and RPMD-160 this must be at least 20 bytes; for MD5 this must be at least 16 bytes. For SHA-224 and SHA-256, the length must be at least 32 bytes long. Even though the length of the SHA-224 hash is less than SHA-256, the extra bytes are used as a work area during the generation of the hash value. The SHA-224 value is left-justified and padded with zeroes.

For SHA-384 and SHA-512, the length must be at least 64 bytes long. Even though the length of the SHA-384 hash is less than SHA-512, the extra bytes are used as a work area during the generation of the hash value. The SHA-384 value is left-justified and padded with zeroes.

`hash`

Direction	Type
Input/Output	String

This field contains the hash, left-justified. The processing of the rest of the field depends on the implementation. 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.

`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 MD5, SHA-1 and SHA-256 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

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 176. One-way hash generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	SHA-1 requires CPACF Keywords SHA-224, SHA-256, SHA-384 and SHA-512 are not supported.
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	Keywords SHA-384 and SHA-512 are not supported
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions	
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions	
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions	
IBM z13	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 and reason codes," on page 943 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 and reason codes," on page 943 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 177. 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.

Table 177. Keywords for symmetric MAC generate control information (continued)

Keyword	Meaning
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. Your application program must not change the data in this string. 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 the first call, initialize this parameter as binary zeros.

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, the text_length can be 0 when specifying the ONLY or LAST keyword. However, the last call should be preceded by a first or middle call.

Required hardware

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 178. Symmetric MAC generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions	
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions	
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions	
IBM z13	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 )
```

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 and reason codes," on page 943 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 and reason codes," on page 943 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 179. 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.

Table 179. Keywords for symmetric MAC verify control information (continued)

Keyword	Meaning
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. Your application program must not change the data in this string. 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 the first call, initialize this parameter as binary zeros.

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 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, the text_length can be 0 when specifying the ONLY or LAST keyword. However, the last call should be preceded by a first or middle call.

Required hardware

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 180. Symmetric MAC verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions	
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions	
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions	
IBM z13	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:

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

3. The VISA CVV and the MasterCard CVC refer to the same value. CVV is used here to mean both CVV and CVC.

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.

- 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 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* callable 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.

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* callable service. 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.

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 1093.

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
- VISA 2
- VISA 3
- VISA 4
- ECI 2
- ECI 3

The formats are discussed in “PIN Formats and Algorithms” on page 1093.

PIN-Encrypting Keys

A unique master key variant enciphers each type of key. For further key separation, an installation can choose to have each PIN block format enciphered under a different PIN-encrypting key. The PIN-encrypting keys can have a 16-byte PIN block variant constant exclusive ORed on them prior to using to translate or verify PIN blocks. This is specified in the format control field in the Encrypted PIN translate and Encrypted PIN verify callable services.

You should only use PIN block variant constants when you are communicating with another host processor with the Integrated Cryptographic Service Facility.

Derived unique key per transaction algorithms

ICSF supports ANSI X9.24 derived unique key per transaction algorithms to generate PIN-encrypting keys from user data. ICSF supports both single-length and double-length key generation. Keywords for single-length and double-length key generation cannot be mixed.

Encrypted PIN Translate

The UKPTIPIN, IPKTOPIN and UKPTBOTH keywords will cause the service to generate single-length keys. DUKPT-IP, DKPT-OP and DUKPT-BH are the respective keywords to generate double-length keys. The *input_PIN_profile* and *output_PIN_profile* must supply the current key serial number when these keywords are specified.

Encrypted PIN Verify

The UKPTIPIN keyword will cause the service to generate single-length keys. DUKPT-IP is the keyword for double-length key generation. The input_PIN_profile must supply the current key serial number when these keywords are 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 the z196, 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 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 enable, the following restrictions will be enforced.

- CSNBPTR and CSNBSPN will not accept IBM 3624 PIN format in the output profile parameter when the input profile parameter is not IBM 3624.
- CSNBPTR 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 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 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 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 enable, CSNBPTR will allow reformatting of the PIN block as shown in the following table.

Table 181. ANSI X9.8 PIN - Allow only ANSI PIN blocks

Reformat To: **From:**	**ISO Format 0**	**ISO Format 1**	**ISO Format 3**
ISO Format 0	Reformat permitted Change of PAN not permitted	Not permitted	Reformat permitted Change of PAN not permitted

Table 181. ANSI X9.8 PIN - Allow only ANSI PIN blocks (continued)

Reformat To: / From:	ISO Format 0	ISO Format 1	ISO Format 3
ISO Format 1	Reformat permitted	Reformat permitted	Reformat permitted
ISO Format 3	Reformat permitted Change of PAN not permitted	Not permitted	Reformat permitted Change of PAN not permitted

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 doesn't 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.

The PIN Profile

The PIN profile consists of:

- PIN block format (see "PIN Block Format")
- Format control (see "Format Control" on page 500)
- Pad digit (see "Pad Digit" on page 500)
- Current Key Serial Number (for UKPT and DUKPT – see "Current Key Serial Number" on page 501)

Table 182 shows the format of a PIN profile.

Table 182. Format of a PIN Profile

Bytes	Description
0–7	PIN block format
8–15	Format control
16–23	Pad digit
24–47	Current Key Serial Number (for UKPT and DUKPT)

PIN Block Format

This keyword specifies the format of the PIN block. The 8-byte value must be left-justified and padded with blanks. Refer to Table 183 on page 498 for a list of valid values.

Table 183. 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
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 184. 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.
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.

Table 184. PIN Block Format and PIN Extraction Method Keywords (continued)

PIN Block Format	PIN Extraction Method Keywords	Description
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.
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.

Enhanced PIN Security Mode

An Enhanced PIN Security Mode is available. This optional mode is selected by enabling the PTR 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 185 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 185. 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	PIN-block format and PIN-extraction method Clear_PIN_Generate_Alternate Encrypted_PIN_Translate Encrypted_PIN_Verify	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	Clear_PIN_Generate_Alternate Encrypted_PIN_Translate Encrypted_PIN_Verify PIN Change/Unblock	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	Clear_PIN_Encrypt Encrypted_PIN_Generate Encrypted_PIN_Translate	PIN formatting does not examine the PIN, in the output PIN block, to see if it contains the pad digit.
3621 or 3624 format and PADDIGIT	Encrypted_PIN_Translate	Restricted to non-decimal digit for PAD digit.

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 186 on page 501 shows the format of a pad digit. The PIN profile pad digit must be specified in upper case.

Table 186. 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 187. 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
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 indicating that the PAD digit is not valid if all of these conditions are met:

1. The PTR Enhanced Security access control point is enabled in the active role
2. The output PIN profile specifies 3621 or 3624 as the PIN-block format
3. 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 nondecimal 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 188 on page 502 shows the format of the current key serial number.

When UKPT or DUKPT is specified, the PIN profile parameter is extended to a 48-byte field and must contain the current key serial number.

Table 188. 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 doesn’t 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 FPE services require some knowledge of the input clear text character set in order to create the appropriate output ciphertext. The CSNBFPEE, CSNBFPED, and CSNBFPET 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 189. Base-10 alphabet

VFPE 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
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

Base-16 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 base-16 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 VFPE alphabet number prior to encryption. Therefore, any of the data types shown in Table 190 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'): 16
```

Table 190. Base-16 alphabet

VFPE 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

Table 190. Base-16 alphabet (continued)

VFPE 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
9	9	11001	9	1001	0111001	11111001
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
15	?	11111	F	1111	1000110	11000110

Track 1 alphabet

This alphabet requires the following values to be used in the VFPE algorithm:

`Number of characters in alphabet('n'): 41`

Table 191. Track 1 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	$	0000100	0100100	01011011
2	(	0001000	0101000	01001101
3	)	1001001	0101001	01011101
4	-	0001101	0101101	01100000
5	0	0010000	0110000	11110000
6	1	1010001	0110001	11110001
7	2	1010010	0110010	11110010
8	3	0010011	0110011	11110011
9	4	1010100	0110100	11110100
10	5	0010101	0110101	11110101
11	6	0010110	0110110	11110110
12	7	1010111	0110111	11110111
13	8	1011000	0111000	11111000
14	9	0011001	0111001	11111001
15	A	1100001	1000001	11000001
16	B	1100010	1000010	11000010
17	C	0100011	1000011	11000011
18	D	1100100	1000100	11000100
19	E	0100101	1000101	11000101
20	F	0100110	1000110	11000110

Table 191. Track 1 alphabet (continued)

FPE alphabet number	Character	ISO 7811 modified 7-bit ASCII	Standard data types 7-bit ASCII	Standard data types 8-bit ASCII
21	G	1100111	1000111	11000111
22	H	1101000	1001000	11001000
23	I	0101001	1001001	11001001
24	J	0101010	1001010	11010001
25	K	1101011	1001011	11010010
26	L	0101100	1001100	11010011
27	M	1101101	1001101	11010100
28	N	1101110	1001110	11010101
29	O	0101111	1001111	11010110
30	P	1110000	1010000	11010111
31	Q	0110001	1010001	11011000
32	R	0110010	1010010	11011001
33	S	1110011	1010011	11100010
34	T	0110100	1010100	11100011
35	U	1110101	1010101	11100100
36	V	1110110	1010110	11100101
37	W	0110111	1010111	11100110
38	X	0111000	1011000	11100111
39	Y	1111001	1011001	11101000
40	Z	1111010	1011010	11101001

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 and reason codes," on page 943 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 and reason codes," on page 943 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 192. 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. This value must be 64.

inbound_PIN_encrypting_key_identifier

Direction	Type
Input	String

An operational key token or the label of the CKDS record containing a double length IPINENC key that decrypts the PIN block.

If the token supplied was encrypted under the old master key, the token 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, e.g. 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, the value is the length of the token. The maximum value is 725. The value may be 0 when the “CLEAR” rule array option is specified.

AP_encrypting_key_identifier

Direction	Type
Input	String

An internal key token or the label of the CKDS record containing a double length DATA key used to encrypt the *AP_value*. If the AP Protection Method was specified as CLEAR in the *rule_array* parameter, 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.

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 193. 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

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 194. Authentication Parameter Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114		This service is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Requires the Sept. 2013 or later LIC.
IBM z13	Crypto Express5 CCA Coprocessor	

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
- 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 PTR 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 and reason codes," on page 943 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 and reason codes," on page 943 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 64-byte string containing an internal key token or a key label of an internal key token. The internal key token contains the key that encrypts the PIN block. The control vector in the internal key token must specify an OPINENC key type and have the CPINENC usage bit set to 1.

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 195. 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 497 for additional information.

PAN_data

Direction	Type
Input	String

A 12-byte PAN in character format. The service uses this parameter if the PIN profile specifies the ISO-0 or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 12-byte variable in application storage. The information in this variable will be ignored, but the variable must be specified.

Note: When using the ISO-0 keyword, use the 12 rightmost digits of the PAN data, excluding the check digit. When using the VISA-4 keyword, use the 12 leftmost digits of the PAN data, excluding the check digit.

sequence_number

Direction	Type
Input	Integer

The 4-byte character integer. The service currently ignores the value in this variable. For future compatibility, the suggested value is 99999.

encrypted_PIN_block

Direction	Type
Output	String

The field that receives the 8-byte encrypted PIN block.

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 196. Clear PIN encrypt required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ISO-3 PIN block format is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).

Table 196. Clear PIN encrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes," on page 943 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 and reason codes," on page 943 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 64-byte key label or internal key token that identifies the PIN generation (PINGEN) 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 197 on page 515. The keyword is left-justified in an 8-byte field, and padded on the right with blanks.

Table 197. 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 198 describes the array elements.

Table 198. 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.

Table 198. Array Elements for the Clear PIN Generate Callable Service (continued)

Array Element	Description
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 199 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 199. 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 200. 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.

Table 201. Clear PIN generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

Related information

PIN algorithms are shown in "PIN Formats and Algorithms" on page 1093.

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).

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 PTR 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 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 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 and reason codes,” on page 943 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 and reason codes," on page 943 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

A 64-byte string consisting of an internal token that contains an IPINENC or OPINENC key or the label of an IPINENC or OPINENC key that is used to encrypt the PIN block. If you specify a label, it must resolve uniquely to either an IPINENC or OPINENC key.

PIN_generation_key_identifier

Direction	Type
Input/Output	String

A 64-byte string that consists of an internal token that contains a PIN generation (PINGEN) key or the label of a PINGEN 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 497 for additional information.

PAN_data

Direction	Type
Input	String

A 12-byte field that contains 12 characters of PAN data. The personal account number recovers the PIN from the PIN block if the PIN profile specifies ISO-0 or VISA-4 block formats. Otherwise it is ignored, but you must specify this parameter.

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.

encrypted_PIN_block

Direction	Type
Input	String

An 8-byte field that contains the encrypted PIN that is input to the VISA PVV generation algorithm. The service uses the IPINENC or OPINENC 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 202. 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 203. 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 184 on page 498 may be specified for the given PIN block format. See "PIN Block Format and PIN Extraction Method

Keywords" on page 498 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.

Note: The PIN check length must be less than or equal to the integer specified in the *PIN_length* parameter. If the PIN_check_length variable is greater than the PIN length, the PIN_check_length variable will be set to the PIN length.

`data_array`

Direction	Type
Input	String

Three 16-byte elements. Table 204 on page 522 describes the format when IBM-PINO is specified. Table 205 on page 522 describes the format when VISA-PVV is specified.

Table 204. 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 205. 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 4-byte PVV left-justified and padded 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.

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 206. 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

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.

Table 207. Clear PIN generate alternate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 208. Keywords for CVV Key Combine Rule Array Control Information

Keyword	Meaning
Key Wrapping Method (One 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.
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. 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:** 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 Export 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 table shows the action taken by the service for different combinations of input key types.

Table 209. 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					

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 210. 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
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

Table 210. 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)
yes	yes	no to both	yes	error

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 WRAP-ECB or WRAP-ENH keywords are specified and default key-wrapping method setting does not match the keyword, 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.

Table 211. CVV key combine required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	This service requires the Sep. 2011 or later LIC.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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
- 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 PTR 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 64-byte internal key token or a key label of an internal key token in the CKDS. The internal key token contains the PIN-generating key. The control vector must specify the PINGEN key type and have the EPINGEN usage bit set to 1.

outbound_PIN_encrypting_key_identifier

Direction	Type
Input	String

A 64-byte internal key token or a key label of an internal key token in the CKDS. The internal key token contains the key to be used to encrypt the formatted PIN and must contain a control vector that specifies the OPINENC key type and has the EPINGEN usage bit set to 1.

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 212. 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 213 describes the array elements.

Table 213. 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.

Table 213. Array Elements for the Encrypted PIN Generate Callable Service (continued)

Array Element	Description
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 214 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 214. 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 497 for additional information.

PAN_data

Direction	Type
Input	String

A 12-byte string that contains 12 digits of Personal Account Number (PAN) data. The service uses this parameter if the PIN profile specifies the ISO-0 or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 4-byte variable in application storage. The information in this variable will be ignored, but the variable must be specified.

Note: When using the ISO-0 keyword, use the 12 rightmost digit of the PAN data, excluding the check digit. When using the VISA-4 keyword, use the 12 leftmost digits of the PAN data, excluding the check digit.

sequence_number

Direction	Type
Input	Integer

The 4-byte string that contains the sequence number used by certain PIN block formats. The service uses this parameter if the PIN profile specifies the 3621 or 4704-EPP keyword for the PIN block format. Otherwise, ensure that this parameter is a 4-byte variable in application data storage. The information in the variable will be ignored, but the variable must be declared. To enter a sequence number, do this:

- Enter 99999 to use a random sequence number that the service generates.
- For the 3621 PIN block format, enter a value in the range from 0 to 65535.
- For the 4704-EPP PIN block format, enter a value in the range from 0 to 255.

encrypted_PIN_block

Direction	Type
Output	String

The field where the service returns the 8-byte encrypted PIN.

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 215. 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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 216. Encrypted PIN generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ISO-3 PIN block format is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).

Table 216. Encrypted PIN generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

Encrypted PIN Translate (CSNBPTR and CSNEPTR)

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 PTR 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 and reason codes," on page 943 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 and reason codes," on page 943 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 497 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 217. 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 498 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 184 on page 498 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.

Table 217. Keywords for Encrypted PIN Translate (continued)

Keyword	Meaning
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.
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 497 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 497 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 218. 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 219. 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 **UKPT - PIN Verify, PIN Translate** 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.

Encrypted PIN Translate

Table 220. Encrypted PIN translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ISO-3 PIN block format is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 unique-key-par-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 PTR 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 and reason codes," on page 943 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 and reason codes," on page 943 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 64-byte key label or internal key token containing the PIN-encrypting key (IPINENC) that enciphers the PIN block. If keyword UKPTIPIN or DUKPT-IP 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.

PIN_verifying_key_identifier

Direction	Type
Input/Output	String

The 64-byte key label or internal key token that identifies the PIN verify (PINVER) 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 in the *rule_array* parameter, then the *input_PIN_profile* is extended to a 48-byte field and must contain the current key serial number. See "The PIN Profile" on page 497 for additional information.

If you specify DUKPT-IP in the *rule_array* parameter, then the *input_PIN_profile* is extended to a 48-byte field and must contain the current key serial number. See "The PIN Profile" on page 497 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 personal account number (PAN) is required for ISO-0 and 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.

encrypted_PIN_block

Direction	Type
Input	String

The 8-byte enciphered PIN block that contains the PIN to be verified.

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 221. 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.
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 498 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 184 on page 498 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. 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-IP	The *input_PIN_encrypting_key_identifier* is to be derived using the DUKPT algorithm. The *input_PIN_encrypting_key_identifier* must be a KEYGENKY key with the DUKPT usage bit enabled. The *input_PIN_profile* must be 48 bytes and contain the key serial number.

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 222 describes the array elements.

Table 222. 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.
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 223 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 223. 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 224. 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, the **UKPT - 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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 225. Encrypted PIN verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ISO-3 PIN block format is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

Related information

“PIN Formats and Algorithms” on page 1093 discusses the PIN algorithms in detail.

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 and reason codes,” on page 943 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 and reason codes," on page 943 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 226. 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.
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.

Table 226. Rule array keywords for field level decipher (continued)

Keyword	Meaning
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.
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 clear keys, the length is in bytes and includes only the value of the key. For the KEYIDENT keyword, this parameter must be 64.

Note: Single length DES keys are not supported.

key_identifier

Direction	Type
Input/Output	String

For the KEY-CLR keyword, *key_identifier* specifies the cipher key. The parameter must be left justified.

For the KEYIDENT keyword, *key_identifier* specifies the internal clear DES or AES DATA key token, or the label name of a clear or encrypted DES or AES DATA key token.

If the token supplied was encrypted under the old master key, the token 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.

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 below 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 CKDS encrypted key, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES).

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 listed below must be enabled.

Algorithm	Access Control Point
AESVFPE	Symmetric Key Encipher/Decipher – Encrypted AES keys
TDESVFPE	Symmetric Key Encipher/Decipher – Encrypted DES keys

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 227. Field level decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	Encrypted keys are not supported.
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	Encrypted keys are not supported.
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions Crypto Express 3 Coprocessor	Encrypted keys require the CEX3C with the Nov. 2009 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions Crypto Express 3 Coprocessor	Encrypted keys require the CEX3C with the Nov. 2009 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions Crypto Express 3 Coprocessor Crypto Express 4 Coprocessor	
IBM z13	CP Assist for Cryptographic Functions Crypto Express 5 CCA Coprocessor	

Related information

You **cannot** overlap the source and target text fields. For example:

```
ssssss
     tttttt  is not supported.

tttttt
     ssssss  is not supported.

sssssstttttt 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 228. 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 228. 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 clear keys, the length is in bytes and includes only the value of the key. For the KEYIDENT keyword, this parameter must be 64.

Note: Single length DES keys are not supported.

key_identifier

Direction	Type
Input/Output	String

For the KEY-CLR keyword, *key_identifier* specifies the cipher key. The parameter must be left justified.

For the KEYIDENT keyword, *key_identifier* specifies the internal clear DES or AES DATA key token, or the label name of a clear or encrypted DES or AES DATA key token.

If the token supplied was encrypted under the old master key, the token 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 much 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 information above 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 CKDS encrypted key, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES).

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 listed below must be enabled.

Algorithm	Access Control Point
AESVFPE	Symmetric Key Encipher/Decipher – Encrypted AES keys
TDESVFPE	Symmetric Key Encipher/Decipher – Encrypted DES keys

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 229. Field level encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	CP Assist for Cryptographic Functions	Encrypted keys are not supported.
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	Encrypted keys are not supported.
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions Crypto Express 3 Coprocessor	Encrypted keys require the CEX3C with the Nov. 2009 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	CP Assist for Cryptographic Functions Crypto Express 3 Coprocessor	Encrypted keys require the CEX3C with the Nov. 2009 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	CP Assist for Cryptographic Functions Crypto Express 3 Coprocessor Crypto Express 4 Coprocessor	
IBM z13	CP Assist for Cryptographic Functions Crypto Express 5 CCA Coprocessor	

Related information

You **cannot** overlap the source and target text fields. For example:

```
ssssss
     tttttt  is not supported.

tttttt
     ssssss  is not supported.

sssssstttttt is supported.

ssssss
tttttt  is supported.

s represents the source text and t represents the target text.
```

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 key serial number if DUKPT key management is used, 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 and reason codes," on page 943 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 and reason codes," on page 943 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 230. 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.
Algorithm (required)	
TDES	Specifies the use of CBC mode triple-DES encryption.
Mode (one required)	
CBC	Specifies the user 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 191 on page 504 for valid characters.

Table 230. Rule array keywords for FPE decipher (continued)

Keyword	Meaning
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 191 on page 504 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'.
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 191 on page 504 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 191 on page 504 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. The value must be 64 because only fixed length DES tokens are supported as the key identifier.

`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 token or the key label of an operational token in key storage.

For key management 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 ANS X9.24 Part 1. 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.

If the token supplied was encrypted under the old master key, the token 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. The value must be 10 if a DUKPT key is specified in the *key_identifier* parameter. If a data encryption key is specified in the *key_identifier* parameter, this value must be set to zero.

`derivation_data`

Direction	Type
Input	String

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.

`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

Specifies the length in bytes of the *DUKPT_PIN_key_identifier* parameter. If the PINKEY rule-array keyword is specified, set this value to 64. Otherwise, set this value to 0. 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

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.

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 231. FPE decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114		This service is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12		This service is not supported.
IBM z13	Crypto Express5 CCA Coprocessor	

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 key serial number if DUKPT key management is used, 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 and reason codes," on page 943 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 and reason codes," on page 943 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 232. 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.
Algorithm (required)	
TDES	Specifies the use of CBC mode triple-DES encryption.
Mode (one required)	
CBC	Specifies the user 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 191 on page 504 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 191 on page 504 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'.
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.)	

Table 232. Rule array keywords for FPE encipher (continued)

Keyword	Meaning
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 191 on page 504. Only characters defined in Table 191 on page 504 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 191 on page 504. Only characters defined in Table 191 on page 504 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. The value must be 64.

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 token or the key label of an operational token in key storage.

For key management 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 ANS X9.24 Part 1. 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.

If the token supplied was encrypted under the old master key, the token 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. The value must be 10 if a DUKPT key is specified in the *key_identifier* parameter. If a data encryption key is specified in the *key_identifier* parameter, this value must be set to zero.

derivation_data

Direction	Type
Input	String

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.

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

Specifies the length in bytes of the *DUKPT_PIN_key_identifier* parameter. If the PINKEY rule-array keyword is specified, set this value to 64. Otherwise, set this value to 0. 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

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.

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 233. FPE encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114		This service is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12		This service is not supported.
IBM z13	Crypto Express5 CCA Coprocessor	

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 key serial number if DUKPT key management is used, or a double-length TDES key if standard key management is used to recover the plaintext.
- 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 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 234. 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.

Table 234. Rule array keywords for FPE translate (continued)

Keyword	Meaning
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.
Algorithm (required)	
TDES	Specifies the use of CBC mode triple-DES encryption.
Mode (one required)	
CBC	Specifies the user 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 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 191 on page 504 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 191 on page 504 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.)	

Table 234. Rule array keywords for FPE translate (continued)

Keyword	Meaning
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 191 on page 504 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 191 on page 504 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. The value must be 64 because only fixed length DES tokens are supported as the key identifier.

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 token or the key label of an operational token in key storage.

For key management 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 ANS X9.24 Part 1. 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.

If the token supplied was encrypted under the old master key, the token 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. The value must be 64 because only fixed length DES tokens are supported as the key identifier.

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 token or the key label of an operational token in key storage.

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.

If the token supplied was encrypted under the old master key, the token 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. The value must be 10 if a DUKPT key is specified in the *key_identifier* parameter. If a data encryption key is specified in the *key_identifier* parameter, this value must be set to zero.

derivation_data

Direction	Type
Input	String

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.

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. 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

Specifies the length in bytes of the *DUKPT_PIN_key_identifier* parameter. If the PINKEY rule-array keyword is specified, set this value to 64. Otherwise, set this value to 0. 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

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.

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 235. FPE translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114		This service is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12		This service is not supported.
IBM z13	Crypto Express5 CCA Coprocessor	

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 using the TDES-ENC algorithm of the diversified key generation 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

The 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 PIN block format is 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.

- 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.

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 PTR 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 and reason codes," on page 943 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 and reason codes," on page 943 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	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 or two of these options:

Table 236. Rule Array Keywords for PIN Change/Unblock

Keyword	Meaning
Algorithm (optional)	
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.

authentication_issuer_master_key_length

Direction	Type
Input	Integer

The length of the *authentication_issuer_master_key_identifier* parameter. The value must be 64.

authentication_issuer_master_key_identifier

Direction	Type
Input/Output	String

The label name or internal token of a DKYGENKY key type that is to be used to generate the card-unique diversified key. The control vector of this key must be a DKYL0 key that permits the generation of a double-length MAC key (DMAC). This DKYGENKY may not have replicated key halves.

encryption_issuer_master_key_length

Direction	Type
Input	Integer

The length of the *encryption_issuer_master_key_identifier* parameter. The value must be 64.

encryption_issuer_master_key_identifier

Direction	Type
Input/Output	String

The label name or internal token of a DKYGENKY key type that is to be used to generate the card-unique diversified key and the secure messaging session key for the protection of the output PIN block. The control vector of this key must be a DKYL0 key that permits the generation of a SMPIN key type. This DKYGENKY may not have replicated key halves.

key_generation_data_length

Direction	Type
Input	Integer

The length of the *key_generation_data* parameter. This value must be 10, 18, 26 or 34 bytes.

key_generation_data

Direction	Type
Input	String

The data provided to generate the card-unique session key. 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 120 for more information.

new_reference_PIN_key_length

Direction	Type
Input	Integer

The length of the *new_reference_PIN_key_identifier* parameter. The value must be 64.

new_reference_PIN_key_identifier

Direction	Type
Input/Output	String

The label name or internal token of a PIN encrypting key that is to be used to decrypt the *new_reference_PIN_block*. This must be an IPINENC or OPINENC key. If the label name is supplied, the name must be unique in the CKDS.

new_reference_PIN_block

Direction	Type
Input	String

This is an 8-byte field that contains the enciphered PIN block of the new PIN.

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

This is a 12-byte field containing PAN in character format. This data may be needed to recover the new reference PIN if the format is ISO-0 or VISA-4. If neither is used, this parameter may be blanks.

current_reference_PIN_key_length

Direction	Type
Input	Integer

The length of the *current_reference_PIN_key_identifier* parameter. The value must be 64. If the *rule_array* contains VISAPCU1 or AMEXPCU2, this value must be 0.

current_reference_PIN_key_identifier

Direction	Type
Input/Output	String

The label name or internal token of a PIN encrypting key that is to be used to decrypt the *current_reference_PIN_block*. This must be an IPINENC or OPINENC key. If the labelname is supplied, the name must be unique on the CKDS. If the *rule_array* contains VISAPCU1 or AMEXPCU2, this value is ignored.

current_reference_PIN_block

Direction	Type
Input	String

This is an 8-byte field that contains the enciphered PIN block of the new PIN. If the *rule_array* contains VISAPCU1 or AMEXPCU2, this value is ignored.

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

This is a 12-byte field containing PAN in character format. If the VISAPCU2 or the AMEXPCU1 *rule_array* keyword is present and the PIN-profile specifies an ISO-0 or ISO-3 PIN-block format, the variable contains the PAN data. PAN data is used to recover a PIN from an ISO-0 or ISO-3 PIN block.

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 or AMEXPCU2), a format control keyword, NONE, (left aligned and padded on the right with space characters) and 8 byte spaces.

output_PIN_message_length

Direction	Type
Input/Output	Integer

The length of the *output_PIN_message* field. The value must be at least 16. 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.

Access control points

The following table shows the access control points in the domain role that control the function of this service.

Table 237. 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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 238. PIN Change/Unblock hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ISO-3 PIN block format is not supported. AMEXPCU1 and AMEXPCU2 keywords not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC). AMEXPCU1 and AMEXPCU2 keywords require May, 2012 or later version of LIC.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC). AMEXPCU1 and AMEXPCU2 keywords require May, 2012 or later version of LIC for Crypto Express2. AMEXPCU1 and AMEXPCU2 keywords require June, 2012 or later version of LIC for Crypto Express3.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	AMEXPCU1 and AMEXPCU2 keywords require June, 2012 or later version of LIC.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	AMEXPCU1 and AMEXPCU2 keywords require June, 2012 or later version of LIC for Crypto Express3. AMEXPCU1 and AMEXPCU2 keywords require September, 2012 or later version of LIC for Crypto Express4.
IBM z13	Crypto Express5 CCA Coprocessor	

Recover PIN from Offset (CSNBPFO and CSNEPFO)

Use the RecoverPINfromOffset 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 and reason codes," on page 943 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 and reason codes," on page 943 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. This value must be 64.

PIN_encryption_key_identifier

Direction	Type
Input/Output	String

An internal key token or the label of the CKDS record containing an OPINENC key that is used to encrypt the *returned_encrypted_PIN_block*.

If the token supplied was encrypted under the old master key, the token 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. This value must be 64.

PIN_generation_key_identifier

Direction	Type
Input	String

An internal key token or the label of the CKDS record containing a PINGEN key that is used to generate the bank reference PIN.

If the token supplied was encrypted under the old master key, the token 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 12-byte PAN in character format. The personal account number is used in formatting the PIN block if the PIN profile specifies ISO-0, ISO-3, or VISA-4 block formats. Otherwise, ensure that this parameter is a 12-byte variable in application storage. The information in this variable will be ignored, but the variable must be specified.

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 Clean 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.

encrypted_PIN_block

Direction	Type
Output	String

his parameter is an 8-byte field that contains the encrypted customer PIN that was originally used in the Clear PIN Generate Alternate 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.

Access control point

The **Recover PIN From Offset** 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.

Table 239. Recover PIN from Offset required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114		This callable service is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Recover PIN From Offset requires the Sep. 2013 or later LIC.
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes," on page 943 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 and reason codes," on page 943 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 240. 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 EBC mode to encipher the message.

input_key_identifier

Direction	Type
Input/Output	String

The internal token, external token, or key label of an internal token of a double length DES key. The key is recovered in the clear and placed in the text to be encrypted. The control vector of the DES key must not prohibit export.

key_encrypting_key_identifier

Direction	Type
Input/Output	String

If the *input_key_identifier* is an external token, then this parameter is the internal token or the key label of the internal token of IMPORTER or EXPORTER. If it is not, it is a null token. If a key label is specified, the key label must be unique.

secmsg_key_identifier

Direction	Type
Input/Output	String

The internal token or key label of a secure message key for encrypting keys. This key is used to encrypt the updated *clear_text* containing the recovered DES 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.

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 241. Secure messaging for keys required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 242. Rule Array Keywords for Secure Messaging for PINs

Keyword	Meaning
Enciphering mode (optional)	
TDES-CBC	Use CBC mode to encipher the message (default).
TDES-ECB	Use EBC 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 8-byte input PIN block that is to be recovered in the clear and perhaps reformatted, and then placed in the *clear_text* to be encrypted.

PIN_encrypting_key_identifier

Direction	Type
Input/Output	String

The internal token or key label of the internal token of the PIN encrypting key used in encrypting the *input_PIN_block*. The key must be an IPINENC 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 and ISO-3. See “The PIN Profile” on page 497 for additional information.

input_PAN_data

Direction	Type
Input	Character String

The 12 digit personal account number (PAN) if the input PIN format is ISO-0 only. Otherwise, the parameter is ignored.

secmsg_key_identifier

Direction	Type
Input/Output	String

The internal token or key label of an internal token of a secure message key for encrypting PINs. This key is used to encrypt the updated *clear_text*.

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 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.

`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 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.

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.

Access control point

The **Secure Messaging for PINs** 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.

Table 243. Secure messaging for PINs required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ISO-3 PIN block format is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

SET Block Compose (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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 244. 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 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 bytes. 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 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.

Table 245. SET block compose required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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 and reason codes," on page 943 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 and reason codes," on page 943 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 246. 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

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 247. 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.

Table 248. SET block decompose required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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 and reason codes," on page 943 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 and reason codes," on page 943 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 or 2.

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 inTable 249 on page 623.

Table 249. 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)	
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.

transaction_key_identifier

Direction	Type
Input	String

The labelname or internal token of a MAC or MACVER class key. Key may be single or double length. When the CSC-V2 keyword is specified, the key must be a double-length 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 250. 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

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 251. 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.

Table 252. Transaction validation required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Requires May 2004 or later version of Licensed Internal Code (LIC) CSC-V1 and CSC-V2 keywords not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	CSC-V1 and CSC-V2 keywords require May, 2012 or later version of LIC.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	CSC-V1 and CSC-V2 keywords require May, 2012 or later version of LIC for Crypto Express2. CSC-V1 and CSC-V2 keywords require June, 2012 or later version of LIC for Crypto Express3.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	CSC-V1 and CSC-V2 keywords require June, 2012 or later version of LIC.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	CSC-V1 and CSC-V2 keywords require June, 2012 or later version of LIC for Crypto Express3. CSC-V1 and CSC-V2 keywords require September, 2012 or later version of LIC for Crypto Express4.
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes," on page 943 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 and reason codes," on page 943 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 253. 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. Requires z990, z890, z9 EC or z9 BC with Jan. 2005 or higher version of Licensed Internal Code (LIC).
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

A 64-byte string that is the internal key token containing a single-length DATA or MAC key or double-length MAC key or the label of a CKDS record containing a single-length DATA or MAC key or double-length MAC key. MACVER keys are not supported.

When this key is a double-length key, *CVV_key_B_identifier* must be 64 byte of binary zero. When a double-length MAC key is used, the CV bits 0-3 must indicate a CVVKEY-A key (0010).

A single-length key contains the key-A key that encrypts information in the CVV process. The left half of a double-length key contains the key-A key that encrypts information in the CVV process and the right half contains the key-B key that decrypts information.

CVV_key_B_Identifier

Direction	Type
Input/Output	String

A 64-byte string that is the internal key token containing a single-length DATA or MAC key or the label of a CKDS record containing a single-length DATA or

MAC key. MACVER keys are not supported. When *CVV_key_A_identifier* a double-length key, this parameter must be 64 byte of binary zero. The key contains the key-B key that decrypts information in the CVV process.

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.

Table 254. VISA CVV service generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Combined CVV keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Combined CVV keys are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Combined CVV keys are not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Combined CVV keys require the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes," on page 943 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 and reason codes," on page 943 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 255. 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. Requires z990, z890, z9 EC or z9 BC with Jan. 2005 or higher version of Licensed Internal Code (LIC).
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.

Table 255. CVV Verify Rule Array Keywords (continued)

Keyword	Meaning
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

A 64-byte string that is the internal key token containing a single-length DATA or MAC key or double-length MAC key or the label of a CKDS record containing a single-length DATA or MAC key or double-length MAC key.

When this key is a double-length key, *CVV_key_B_identifier* must be 64 byte of binary zero. When a double-length MAC key is used, the CV bits 0-3 must indicate a CVVKEY-A key (0010).

A single-length key contains the key-A key that encrypts information in the CVV process. The left half of a double-length key contains the key-A key that encrypts information in the CVV process and the right half contains the key-B key that decrypts information.

CVV_key_B_Identifier

Direction	Type
Input/Output	String

A 64-byte string that is the internal key token containing a single-length DATA or MAC key or the label of a CKDS record containing a single-length DATA or MAC key. When *CVV_key_A_identifier* a double-length key, this parameter must be 64 byte of binary zero. The key contains the key-B key that decrypts information in the CVV process.

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.

VISA CVV Service Verify

Table 256. VISA CVV service verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	Combined CVV keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Combined CVV keys are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Combined CVV keys are not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Combined CVV keys require the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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:

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 635.

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 CSFBDDPG(
            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 and reason codes," on page 943 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 and reason codes," on page 943 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 257. 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.

Table 258. DK Deterministic PIN Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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 and reason codes," on page 943 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 and reason codes," on page 943 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 259. 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.
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 IOS-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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 260. DK Migrate PIN required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.

Table 260. DK Migrate PIN required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the June 2014 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the June 2014 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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 CSNBPDPMT(
            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 and reason codes," on page 943 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 and reason codes," on page 943 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	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. 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	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.

ISO_encrypted_PIN_block

Direction	Type
Input	String

The 8-byte encrypted PIN block with the PIN in ISO-1 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. 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* 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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 261. DK PAN Modify in Transaction required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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.

Table 262. DK PAN Translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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 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_ISO1_PIN_block_length,
             cur_ISO1_PIN_block,
             new_ISO1_PIN_block_length,
             new_ISO1_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 and reason codes," on page 943 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 and reason codes," on page 943 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, 1, 2, 3, 4, or 5.

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 263. Rule array keywords for the Card Replace PRW 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 to verify the encrypted PIN block.
Script selection keyword (One, optional)	
NOSCRIPT	Do not return an encrypted SMPIN message with a MAC. This is the default value.
TDES-CBC	Use CBC mode to encrypt the script.
TDES-ECB	Use ECB mode to encrypt the script.
Pin encryption keyword (One, optional) Only valid if TDES-CBC or TDES-ECB is selected above.	
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, optional) Only valid if TDES-CBC or TDES-ECB is selected above.	
EMVMACD	Specifies the EMV-related message-padding and calculation method.
TDES-MAC	Specifies the ANS X9.9 Option 1 (binary data) procedure and a CBC Triple-DES encryption of the data.
X9.19OPT	Specifies the ANS X9.19 Optional Procedure. A double-length key is required. This is the default value.
MAC Length and presentation (One, optional) Only valid if TDES-CBC or TDES-ECB is selected above.	
MACLEN8	Specifies a 8-byte MAC. This is the default value.

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_ISO1_PIN_block_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *cur_ISO1_PIN_block* parameter. This value must be 8.

cur_ISO1_PIN_block

Direction	Type
Input	String

The 8-byte encrypted PIN block with the current PIN in ISO-1 format.

new_ISO1_PIN_block_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *new_ISO1_PIN_block* parameter. This value must be 8.

new_ISO1_PIN_block

Direction	Type
Input	String

The new encrypted PIN block with the customer chosen PIN. The PIN block must be in ISO-1 format.

card_script_data_length

Direction	Type
Input	Integer

Specifies the length in bytes of the *card_script_data* parameter. If NOSCRIPT is specified in the rule array, this value must be 0. Otherwise, the value must be no greater than 4096 and a multiple of 8.

card_script_data

Direction	Type
Input	String

The clear text string to be updated with the clear PIN block and encrypted.

script_offset

Direction	Type
Input	Integer

The offset to the location for the PIN block in the script. Specify the first byte of the clear text as offset 0. This offset plus the value of *script_offset_field_length* must be less than or equal to the *card_script_data_length*. If NOSCRIPT is specified in the rule array, this parameter is ignored.

script_offset_field_length

Direction	Type
Input	Integer

The length of the field within *card_script_text* parameter at *script_offset* where the new PIN value is to be placed. Length must be 8. The PIN block must fit entirely within the *card_script_text*. 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. If NOSCRIPT or TDES-ECB is specified in the rule array, this value must be 0. Otherwise, it must be 8.

script_initialization_vector

Direction	Type
Input	String

The 8-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' for additional information. You can use PIN-block formats ISO-0, ISO-1, ISO-2, and ISO-3 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. 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.

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. 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.

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 to decrypt the script. 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, the key type must be SECMSG key type 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. 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

Specifies the length in bytes of the *script* parameter. If the rule array specifies TDES-CBC or TDES-ECB, this value must be at least as long as the *script* parameter. Otherwise, it must be 0.

script

Direction	Type
Output	String

The encrypted output script. The length of the field must be at least as long as the input 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. Otherwise, this must be at least 8.

script_MAC

Direction	Type
Output	String

The 8-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.

Access control points

The **DK PIN Change** 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.

Table 264. DK PIN Change required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

DK PIN Verify (CSNBDPV and CSNEDPV)

Use the DK PIN Verify callable service to verify an ISO-1 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 and reason codes," on page 943 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 and reason codes," on page 943 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	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. There are no keywords for this service.

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.

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. 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.

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 265. DK PIN Verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 266. 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.

Table 267. DK PRW Card Number Update required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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.

Table 268. DK PRW CMAC Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 269. Rule array keywords for the PIN Generate2 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.

Table 270. DK Random PIN Generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 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.

Table 271. DK Regenerate PRW required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC		This service is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	DK AES PIN key support requires the November 2013 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	DK AES PIN key support requires the September 2013 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

Chapter 10. Using Digital Signatures

This topic describes the PKA callable services that support using digital signatures to authenticate messages.

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. The digital signature generate callable service may use an RSA or ECC private key, depending on the algorithm you are using.

Private keys must be valid for signature usage. This service supports these methods:

- ANSI X9.30 (ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

Note:

1. The maximum signature length is 512 bytes (4096 bits).
2. For secure PKCS #11 private keys, the only supported services are ANSI X9.30 (ECDSA) and RSA PKCS 1.1.

The input text should have been previously hashed using either the one-way hash generate callable service or the MDC generation callable service. If the signature formatting algorithm specifies ANSI X9.31, you must specify the hash algorithm used to hash the text (SHA-1 or RPMD-160). See "Formatting Hashes and Keys in Public-Key Cryptography" on page 1113.

If the *private_key_identifier* specifies an RSA private key, you select the method of formatting the text through the *rule_array* parameter. If the *private_key_identifier* specifies an ECC private key, the ECC signature is generated according to ANSI X9.30.

For secure PKCS #11 keys, if the *private_key_identifier* specifies an RSA private key, you must select the PKCS-1.1 method of formatting the text through the *rule_array* parameter. If the *private_key_identifier* specifies an ECC private key, the ECC signature is generated according to ANSI X9.30.

Note: For RSA PKCS 1.0 or 1.1, the message digest and the message-digest algorithm identifier are combined into an ASN.1 value of type DigestInfo, which is BER-encoded to give an octet string D (see Table 272 on page 709). D is the text string supplied in the *hash* variable.

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,
            hash_length,
            hash,
            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 and reason codes," on page 943 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 and reason codes," on page 943 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, or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. One keyword specifies the method for calculating the digital signature. Another keyword specifies formatting of the hash value for RSA digital signature generation. A third keyword specifies the hash method used to prepare the hash value for RSA digital signature generation. Table 272 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 272. Keywords for Digital Signature Generate Control Information

Keyword	Meaning
Digital Signature Formatting Method (optional, valid for RSA digital signature generation only)	
ISO-9796	Calculate the digital signature on the *hash* according to ISO-9796-1. Any hash method is allowed. This is the default.
PKCS-1.0	Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc. Public Key Cryptography Standards #1 V1.5 block type 00. The text must have been hashed prior to inputting to this service. Note: Use of block-type 00 has been effectively depreciated by the industry. Rule PKCS-1.1 is recommended.
PKCS-1.1	Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc. Public Key Cryptography Standards #1 V1.5 block type 01. The text must have been hashed prior to inputting to this service.
ZERO-PAD	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.
X9.31	Format according to the ANSI X9.31 standard. The input text must have been previously hashed with one of these hash algorithms:
Hash Method Specification: Required with X9.31	
RPMD-160	Hash the input text using the RIPEMD-160 hash method.
SHA-1	Hash the input text using the SHA-1 hash method.
Signature algorithm (optional, supported on the CEX3C or later coprocessor)	
RSA	RSA processing is to occur.
ECDSA	The elliptic curve digital signature algorithm is to be used. When specified, this is the only keyword permitted in the Rule Array.

private_key_identifier_length

Direction	Type
Input	Integer

The length of the *private_key_identifier* field. The maximum size is 3500 bytes.

private_key_identifier

Direction	Type
Input	String

This is an internal token or label of an RSA or ECC private key or Retained key. If the signature format is X9.31, the modulus of the RSA key must have a length of at least 1024 bits.

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.

hash_length

Direction	Type
Input	Integer

The length of the *hash* parameter in bytes. It must be the exact length of the text to sign. The maximum size is bytes. If you specify ZERO-PAD in the *rule_array* parameter, the length is restricted to 36 bytes unless the RSA key is a signature only key, then the maximum length is 512 bytes.

The hash length limit is controlled by an access control point. 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 always disabled by default in the domain role. You must have a TKE workstation to enable it.

hash

Direction	Type
Input	String

The application-supplied text on which to generate the signature. The input text must have been previously hashed, and for PKCS formatting, it must be BER-encoded as previously described. For X9.31, the hash algorithms must have been either SHA-1 or RIPEMD-160. See the *rule_array* parameter for more information.

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 is 512 bytes.

Note: For RSA, this must be at least the RSA modulus size (rounded up to a multiple of 32 bytes for the X9.31 signature format, or one byte for all other signature formats).

For RSA, this field is updated with the minimum byte length of the digital signature.

For ECDSA, signature algorithm R concatenated with S is the digital signature. The maximum output value will be 1042 bits (131 bytes). The size of the signature is determined by the size of P. Both R and S will have size P. For prime curves, the maximum is 2 * 521 bits. For brain pool curves, the maximum size is 2 * 512 bits.

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
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.

X9.31 requires the RSA token to have a modulus bit length of at least 1024 bits and the length must also be a multiple of 256 bits (or 32 bytes).

The length of the *hash* parameter in bytes. It must be the exact length of the text to sign. The maximum size is 512 bytes. If you specify ZERO-PAD in the *rule_array* parameter, the length is restricted to 36 bytes unless the RSA key is a signature only key, then the maximum length is 512 bytes.

For CCA RSA keys, the hash length limit is controlled by the **DSG ZERO-PAD unrestricted hash length** access 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.

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*.

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 273. Digital signature generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ECC not supported. RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ECC not supported. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).

Table 273. Digital signature generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ECC not supported. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express3 Coprocessor	ECC support requires the Sep. 2010 licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
	Crypto Express4 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key
IBM z13	Crypto Express5 CCA Coprocessor	
	Crypto Express5 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key

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 or ECC public key, depending on the digital signature algorithm used to generate the signature.
- The digital signature verify callable service can also use the 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**, an error will occur if the **PKA_public_key_identifier** does not contain a trusted block.

This service supports these methods:

- ANSI X9.30 (ECC)
- ANSI X9.31 (RSA)
- ISO 9796 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

Input text should have been previously hashed. You can use either the one-way hash generate callable service or the MDC generation callable service. See also "Formatting Hashes and Keys in Public-Key Cryptography" on page 1113.

Note: The maximum signature length is 512 bytes.

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,
            hash_length,
            hash,
            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 and reason codes,” on page 943 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 and reason codes,” on page 943 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

Contains an array of keywords that provide control information to the callable service. One keyword specifies the method to use to verify the RSA digital signature. Another keyword specifies the input token is a Trusted Block. A third keyword specifies the algorithm used to validate the signature. Table 274 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 274. Keywords for Digital Signature Verify Control Information

Keyword	Meaning
Digital Signature Formatting Method (optional, RSA only)	
X9.31	Format according to the ANSI X9.31 standard.
ISO-9796	Calculate the digital signature on the hash according to ISO 9796-1. Any hash method is allowed. This is the default.
PKCS-1.0	Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc., Public Key Cryptography Standards #1 V1.5 block type 00 and compare to the digital signature. The text must have been hashed prior to inputting to this service. Note: Use of block-type 00 has been effectively depreciated by the industry. Rule PKCS-1.1 is recommended.
PKCS-1.1	Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc., *Public Key Cryptography Standards #1* block type 01 and compare to the digital signature. The text must have been hashed prior to inputting to this service.
ZERO-PAD	Format the hash by padding it on the left with binary zeros to the length of the PKA key modulus. Any supported hash function is allowed.
PKA public key token type (one, optional)	
TPK-ONLY	The PKA_public_key_identifier must be a trusted block that contains, at a minimum, two sections: 1. Trusted Block Information section 0x14 which is required for all trusted blocks and 2. Trusted Public Key section 0x11 which contains the trusted public key and usage rules that indicate whether or not the trusted public key can be used in digital signature operations.
Signature Algorithm (optional, supported on the CEX3C or later coprocessor)	
RSA	RSA processing is to occur. This is the default value.
ECDSA	The elliptic curve digital signature algorithm is to be used. When specified, this is the only keyword permitted in the Rule Array.

PKA_public_key_identifier_length

Direction	Type
Input	Integer

The length of the *PKA_public_key_identifier* parameter containing the public key token or label. The maximum size is 3500 bytes.

PKA_public_key_identifier

Direction	Type
Input	String

A token or label of the RSA or ECC public key or internal trusted block. If this parameter contains a token or the label of an Internal Trusted Block, the *rule_array* parameter must specify TPK-ONLY.

hash_length

Direction	Type
Input	Integer

The length of the *hash* parameter in bytes. It must be the exact length of the text that was signed. The maximum size is 512 bytes.

hash

Direction	Type
Input	String

The application-supplied text on which the supplied signature was generated. The text must have been previously hashed and, for PKCS formatting, BER-encoded as previously described.

signature_field_length

Direction	Type
Input	Integer

The length in bytes of the *signature_field* parameter. The maximum size is 512 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 ability to recover a message from a signature (which ISO-9796 allows but does not require) is **not** supported.

The exponent of the RSA public key must be odd.

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.

X9.31 requires the RSA token to have a modulus bit length of at least 1024 bits and the length must also be a multiple of 256 bits (or 32 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 **Digital Signature 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.

Table 275. Digital signature verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor PCI Cryptographic Accelerator	ECC not supported. Trusted key block not supported. TPK-ONLY keyword not supported. RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor Crypto Express2 Accelerator	ECC not supported. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express2 Accelerator	ECC not supported. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express3 Coprocessor Crypto Express3 Accelerator	ECC support requires the Sep. 2010 licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor Crypto Express3 Accelerator	RSA clear key support with moduli within the range 2048-bit and 4096-bit requires the Sep. 2011 or later licensed internal code (LIC).

Table 275. Digital signature verify required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express3 Accelerator Crypto Express4 CCA Coprocessor Crypto Express4 Accelerator	
IBM z13	Crypto Express5 CCA Coprocessor (CEX5C) Crypto Express5 CCA Accelerator (CEX5A)	

Chapter 11. Managing PKA Cryptographic Keys

This topic describes the callable services that generate and manage PKA keys.

PKA Key Generate (CSNDPKG and CSNFPKG)

Use the PKA key generate callable service to generate RSA or ECC 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. 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 729 for a summary of restrictions.

ECC key generation requires this information in the skeleton token:

- The key type: ECC
- The type of curve: Prime or Brainpool
- The size of P in bits: 192, 224, 256, 384 or 521 for Prime curves and 160, 192, 224, 256, 320, 384, or 512 for Brainpool curves
- Key usage information
- 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

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 276 for a list. A keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 276. 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.
XPORT	Encipher the private key under the *transport_key_identifier*.
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.
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 value must be 0 for ECC tokens. For RSA tokens, the regeneration_data_length can be non-zero. If it is non-zero, it must be 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 3500 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 curve type and bit length for ECC key generation, or the required modulus length and public exponent for RSA key 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.

transport_key_identifier

Direction	Type
Input	String

A variable-length field containing an AES or DES key identifier used to encrypt the generated key. For RSA keys, this may be an AES or DES transport key. When there is an RSA private key section X'30' or X'31' in the skeleton token, an AES transport key must be specified. The key-usage field in the AES key must allow the key to wrap an RSA key. For all other RSA private key sections, a DES transport key must be specified. For ECC keys, this must be an AES transport key which is able to wrap an ECC key.

If the XPORT Rule is not specified, this parameter must be 64 bytes of binary zeros.

For XPORT rule, this is an IMPORTER or EXPORTER key or the label of an IMPORTER or EXPORTER key. If you specify a label, it must resolve uniquely to either an IMPORTER or EXPORTER key. This parameter is a:

- 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.

generated_key_token_length

Direction	Type
Input/Output	Integer

The length of the generated key token. The field is checked to ensure it is at least equal to the token being returned. The maximum size is 3500 bytes. 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 ECC or RSA 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 overlayed if an ECC token is specified in the *skeleton_key_identifier*
- A retained key is not overlayed 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

2048-bit RSA keys may have a public exponent in the range of 1-256 bytes. 2049- to 4096-bit RSA key public exponents are restricted to the values 3 and 65537.

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 277. 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

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 278. PKA key generate required hardware

Server	Required Cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ECC not supported. RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ECC not supported. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ECC not supported. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express3 Coprocessor	ECC support requires the Sep. 2010 licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	ECC Clear Key and Internal token support requires the Sep. 2010 licensed internal code (LIC). ECC External token and Diffie-Hellman support requires the Sep. 2011 or later licensed internal code (LIC). Wrapping of RSA keys with the ECC master key or AES transport keys is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

PKA Key Import (CSNDPKI and CSNFPKI)

Use this service to import an external PKA private key token. (The private key must consist of a PKA private key and public key.) The secret values of the key may be:

- Clear
- Encrypted under a limited-authority DES importer key or an AES importer key if the *source_key_identifier* is an RSA token
- Encrypted under an AES Key Encryption Key if the *source_key_identifier* is an ECC token

This service can also import a clear PKA key. The PKA key token build service creates a clear PKA key token.

This service can also import an external trusted block token for use with the remote key export callable service.

Output of this service is an ICSF internal token of the RSA or ECC private key or trusted block.

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 and reason codes,” on page 943 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 and reason codes," on page 943 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 or 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 279. Keywords for PKA Key Import

Keyword	Meaning
Token Type (optional)	
RSA	Specifies that the key token is for an RSA key. This is the default.
ECC	Specifies that the key token is for an ECC key.
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. The maximum size is 3500 bytes.

source_key_identifier

Direction	Type
Input	String

Contains an external token or label of a PKA private key, without section identifier 0x14 (Trusted Block Information), or the trusted block in external form as produced by the Trusted Block Create (CSNDTBC and CSNETBC) service with the ACTIVATE keyword.

If a PKA private key without the section identifier 0x14 is passed in:

- There are no qualifiers. A retained key cannot be used.
- The key token must contain both public-key and private-key information. The private key can be in cleartext or it can be enciphered.
- This is the output of the PKA key generate (CSNDPKG) callable service or the PKA key token build (CSNDPKB) callable service.
- If encrypted, it was created on another platform.

If a PKA key token with section 0x14 is passed in:

- This service will be used to encipher the MAC key within the trusted block under the PKA master key instead of the IMP-PKA key-encrypting key.
- The importer_key_identifier must contain an IMP-PKA KEK in this case.

importer_key_identifier

Direction	Type
Input/Output	String

A variable-length field containing an AES or DES key identifier used to wrap the imported key. 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.

This parameter contains one of the following:

- 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.

This parameter is ignored for clear tokens.

target_key_identifier_length

Direction	Type
Input/Output	Integer

The length of the *target_key_identifier* parameter. The maximum size is 3500 bytes. On output, and if the size is of sufficient length, the variable is updated with the actual length of the target_key_identifier field.

target_key_identifier

Direction	Type
Input/Output	String

This field 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. If no label is specified on input, this field is ignored.

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.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 280. PKA key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).

Table 280. PKA key import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	ECC External token and Diffie-Hellman support requires the Sep. 2011 or later licensed internal code (LIC). Importing RSA keys wrapped with an AES transport key is not supported.
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

PKA Key Token Build (CSNDPKB and CSNFPKB)

This callable service can be used 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 and RSA 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: Prime or Brainpool
- The size of P in bits: 192, 224, 256, 384 or 521 for Prime curves and 160, 192, 224, 256, 320, 384, or 521 for Brainpool curves
- Key usage information
- Optionally, application associated data

RSA key generation requires this 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). There are restrictions on the value and length of the public exponent when the length of the modulus is greater than 2048
  - the length of the private exponent d (optional)
  - the pubic 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 pubic exponent e (optional)
- other optional lengths

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,
            reserved_2_length,
            reserved_2,
            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 and reason codes," on page 943 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 and reason codes," on page 943 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, 2 or 3.

rule_array

Direction	Type
Input	String

Keywords that provide control information to the callable service. Table 281 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 281. Keywords for PKA Key Token Build Control Information

Keyword	Meaning
Key Type (required)	
RSA-CRT	This keyword indicates building a token containing an RSA private key in the optimized Chinese Remainder Theorem (CRT) form. The parameter *key_value_structure* identifies the input key values, if supplied.
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, if supplied.
RSA-PUBL	This keyword indicates building a token containing public RSA key information. The parameter *key_value_structure* identifies the input values, if supplied.
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.
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. The private key section id is X'30'
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. The private key section id is X'31.
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, if supplied.
ECC-PUBL	This keyword indicates building a token containing public ECC key information. The parameter *key_value_structure* identifies the input values, if supplied.

Table 281. Keywords for PKA Key Token Build Control Information (continued)

Keyword	Meaning
Key Usage Control (optional)	
KEY-MGMT	Indicates that a private key can be used in both the symmetric key import and the digital signature generate callable services.
KM-ONLY	Indicates that a private key can be used only in symmetric key distribution.
SIG-ONLY	Indicates that a private key cannot be used in symmetric key distribution. This is the default.
Translate Control (optional, only allowed with key types RSA-AESM, RSA-AESC, RSA-PRIV, RSAMEVAR, RSA-CRT, and ECC-PAIR 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.

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 282. 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
RSA-PRIV	648
RSA-PUBL	520
ECC-PAIR	207
ECC-PUBL	139

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, don't count them in the length. Table 283 on page 733 defines the structure and contents as a function of key type.

Table 283. 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)		
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 283. 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, 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)		
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”. TThis 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.

Table 283. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
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.
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, 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
001	001	Reserved x'00'

Table 283. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
002	002	Length of p in bits **0x'00C0'** Prime P-192 **0x'00E0'** Prime P-224 **0x'0100'** Prime P-256 **0x'0180'** Prime P-384 **0x'0209'** Prime P-521 **0x'00A0'** Brain Pool P-160 **0x'00C0'** Brain Pool P-192 **0x'00E0'** Brain Pool P-224 **0x'0100'** Brain Pool P-256 **0x'0140'** Brain Pool P-320 **0x'0180'** Brain Pool P-384 **0x'0200'** Brain Pool P512.
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 which includes one byte to indicate if the value is compressed.
008	ddd	Private key d
008 + ddd	xxx	Public Key value Q
Key value Structure: ECC Public form (ECC_PUBL)		
000	001	Curve type: **0x'00'** Prime Curve **0x'01'** Brain Pool Curve
000	001	Reserved x'00'

Table 283. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
002	002	Length of p in bits **0x'00C0'** Prime P-192 **0x'00E0'** Prime P-224 **0x'0100'** Prime P-256 **0x'0180'** Prime P-384 **0x'0209'** Prime P-521 **0x'00A0'** Brain Pool P-160 **0x'00C0'** Brain Pool P-192 **0x'00E0'** Brain Pool P-224 **0x'0100'** Brain Pool P-256 **0x'0140'** Brain Pool P-320 **0x'0180'** Brain Pool P-384 **0x'0200'** Brain Pool P512.
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 which includes a one byte value indicating compressed or uncompressed key value.
006	xxx	Public key value Q

Note:
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.

`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 name of a private key. 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 upper case 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. The parameter is not valid with key type RSA-PUBL.

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.

reserved_2_length

Direction	Type
Input	Integer

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_2

Direction	Type
Input	String

The *reserved_2* parameter identifies a string that is reserved. The service ignores it.

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 3500 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.

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, DSS, 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.

- For RSA and DSS 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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 284. Rule Array Keywords for PKA Key Token Change

Keyword	Meaning
Algorithm (optional)	
RSA	Specifies that the key token is for a RSA or DSS key or trusted block token. This is the default.
ECC	Specifies that the key token is for an ECC key.
Reencipherment method (required)	

Table 284. Rule Array Keywords for PKA Key Token Change (continued)

Keyword	Meaning
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 key token, the service will change an ECC 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 3500 bytes.

key_identifier

Direction	Type
Input/Output	String

Contains an internal key token of an internal RSA, DSS, ECC, 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.

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 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.

Table 285. PKA key token change required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ECC not supported. Trusted key blocks are not supported. RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ECC not supported. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ECC not supported. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express3 Coprocessor	ECC support requires the Sep. 2010 licensed internal code (LIC).
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

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.

  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.

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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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.

rule_array

Direction	Type
Input	String

A keyword that provides control information to the callable service. See Table 286 for a list. A keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 286. Keywords for PKA Key Translate Rule Array

Keyword	Meaning
Output format, one required	
Output formats for conversion	
EXTDWAKW	Specifies that the source key is an external DES wrapped token to be converted to an AESKW wrapped token.
INTDWAKW	Specifies that the source key is an internal DES wrapped token to be converted to an AESKW wrapped token.
Output formats for translation	
EMVCRT	This keyword indicates translating an external RSA CRT key into EMV CRT format and wrapped using TDES-ECB. The XLATE bit (bit 22) must be set in the *target_transport_key_identifier* control vector.
EMVDDA	This keyword indicates translating an external RSA CRT key into EMV DDA format and wrapped using TDES-CBC. The XLATE bit (bit 22) must be set in the *target_transport_key_identifier* control vector.
EMVDDAE	This keyword indicates translating an external RSA CRT key into EMV DDAE format and wrapped using TDES-ECB. The XLATE bit (bit 22) must be set in the *target_transport_key_identifier* control vector.
SCCOMCRT	This keyword indicates translating the key into the smart card Chinese Remainder Theorem format.
SCCOMME	This keyword indicates translating the key into the smart card Modulus-Exponent format.
SCVISA	This keyword indicates translating the key into the smart card Visa proprietary format.

source_key_identifier_length

Direction	Type
Input	Integer

Length in bytes of the *source_key_identifier* variable. The maximum length is 3500 bytes.

source_key_identifier

Direction	Type
Input	String

This field contains either a key label identifying an RSA private key token or an RSA public-private key token. For smart card processing, the key must be in an external key token. 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.

source_transport_key_identifier_length

Direction	Type
Input	Integer

Length in bytes of the *source_transport_key_identifier* parameter. This value must be 64. For format rule INTDWAKW, the length must be zero.

source_transport_key_identifier

Direction	Type
Input/Output	String

This field contains an internal token or label of a DES key-encrypting key. This key is used to unwrap the input RSA key token specified with parameter *source_key_identifier*. See "Access control points" on page 747 for details on the type of transport key that can be used

target_transport_key_identifier_length

Direction	Type
Input	Integer

Length in bytes of the *target_transport_key_identifier* parameter. When a DES key-encrypting is used, this value must be 64. When an AES key-encrypting key is used, this value is the length of the token. The maximum length is 725. For INTDWAKW, the length must be zero.

target_transport_key_identifier

Direction	Type
Input/Output	String

This field contains an internal token or label of a DES key-encrypting key. This key is used to wrap the output RSA key returned with parameter target_key_token. See "Access control points" on page 747 for details on the type of transport key that can be used.

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 3500 bytes.

target_key_token

Direction	Type
Output	String

This field contains the RSA key in the smartcard format specified in the rule array and is protected by the key-encrypting key specified in the *target_transport_key* parameter. This is not a CCA token, and cannot be stored in the PKDS.

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 287. Required access control points for PKA Key Translate

Rule array keyword	Access control point
INTDWAKW	PKA Key Translate – Translate internal key token
EXTDWAKW	PKA Key Translate – Translate external key token
SCVISA	PKA Key Translate - from CCA RSA to SC Visa Format
SCCOMME	PKA Key Translate - from CCA RSA to SC ME Format
SCCOMCRT	PKA Key Translate - from CCA RSA to SC 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
EMVCRT	PKA Key Translate - from CCA RSA CRT to EMV CRT Format

These access control points control the key type combination shown in this table. One of these access control points must be enabled.

Table 288. 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)

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 289. PKA key translate required hardware

Server	Required Cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Requires the Apr. 2009 or later licensed internal code (LIC). The *rule_array* keywords EMVDDA, EMVDDAE, and EMVCRT are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Requires the Apr. 2009 or later licensed internal code (LIC). The *rule_array* keywords EMVDDA, EMVDDAE, and EMVCRT are not supported.
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	Support for the *rule_array* keywords EMVDDA, EMVDDAE, and EMVCRT requires the March 2014 or later licensed internal code (LIC).
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	Support for the *rule_array* keywords EMVDDA, EMVDDAE, and EMVCRT requires the March 2014 or later licensed internal code (LIC).
IBM z13	Crypto Express5 CCA Coprocessor	

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 and reason codes," on page 943 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 and reason codes," on page 943 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 3500 bytes. When the *source_key_identifier* parameter is a key label, this field specifies the length of the label.

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 supported on the CEX3C and later.

target_public_key_token_length

Direction	Type
Input/Output	Integer

The length of the *target_public_key_token* parameter. The maximum size is 3500 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.

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 isn't 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 doesn't 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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.

Table 290. Retained key delete required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 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 coprocessor.

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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.

Table 291. Retained key list required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor	

Chapter 12. 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 (DSS, RSA, and ECC) and trusted blocks are stored in the PKA Key Data Set (PKDS).

This topic describes the callable services that manage key tokens in the key stores.

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.

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.

Last used reference date
The date that the key material was last referenced. The date is dependent on the setting of the KDSREFDAYS option.

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).

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 key token set to binary zeros 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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. Your applications perform better if you use KGUP to create the initial records and REFRESH the entire in-storage copy of the CKDS, rather than using CKDS key record create to create the initial NULL key entries. This is particularly true if you are creating a large number of key records. CKDS key record create adds a record to a portion of the CKDS that is searched sequentially during key retrieval. Using KGUP followed by a REFRESH puts the null key records in the portion of the CKDS that is ordered in key-label/type sequence. A binary search of the key-label/type sequenced part of the CKDS is more efficient than searching the sequentially ordered section.

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 or the key token supplied in the *key_token* parameter. The record is identified by the label passed in the *key_label* parameter.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 725.

key_token

Direction	Type
Input/Output	String

A symmetric internal token to be written to the CKDS if *key_token_length* is non-zero. If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

Required hardware

No cryptographic hardware is required by this callable service.

CKDS Key Record Delete (CSNBKRD and CSNEKRD)

Use the CKDS key record delete callable service to delete a key record containing a DES or AES token 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 and reason codes," on page 943 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 and reason codes," on page 943 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 can contain an AES or a DES key token. 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).

Usage notes

Secure key tokens cannot be processed when the master key is not loaded.

Clear AES and DES tokens can be processed on a system without a cryptographic coprocessor or accelerator.

Required hardware

No cryptographic hardware is required by this callable service.

CKDS Key Record Read (CSNBKRR and CSNEKRR)

Use the CKDS key record read callable service to copy an internal 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 and reason codes," on page 943 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 and reason codes," on page 943 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).

Usage notes

Clear AES and DES tokens can be processed on a system without a cryptographic coprocessor or accelerator.

Required hardware

No cryptographic hardware is required by this callable service.

CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)

Use this callable service to copy a key token from the in-storage CKDS to application storage. Other cryptographic services can then use the copied key token directly.

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 and reason codes,” on page 943 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 and reason codes," on page 943 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 retrieved.

key_token_length

Direction	Type
Input/Output	Integer

The length of the buffer for the output token. On input, the length of the buffer. The minimum length is 64 bytes and the maximum length is 725 bytes. 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 is written.

Required hardware

No cryptographic hardware is required by this callable service.

CKDS Key Record Write (CSNBKRW and CSNEKRW)

Use the CKDS key record write callable service to write an internal AES or DES key token 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.

This service supports writing a clear AES or DES key token with non-zero key values to 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 and reason codes," on page 943 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 and reason codes," on page 943 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).

Usage notes

Secure AES tokens in the CKDS can only be overwritten by a secure AES token encrypted under the same AES master keys. The same is true for secure DES tokens.

DES tokens cannot be overwritten by an AES token. AES tokens cannot be overwritten by a DES token.

Secure key tokens cannot be processed when the master key is not loaded.

Clear AES and DES 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.

Required hardware

No cryptographic hardware is required by this callable service.

CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)

Use the CKDS key record write2 callable service to write an internal symmetric key token to the variable-length 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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 725.

key_token

Direction	Type
Input/Output	String

An internal symmetric key token to be written to the CKDS. If the token supplied was encrypted under the old master key, the token 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.

Usage notes

The Usage Notes for the CKDS Key Record Write callable service also apply to the CKDS Key Record Write2 callable service when writing fixed-length symmetric key tokens (versions X'00', X'01', and X'04').

A key token cannot be overwritten by another key token that doesn't have the exact same algorithm and key type. For example:

- a DES key token cannot be overwritten by an AES token, and an AES key token cannot be overwritten by a DES token

- an HMAC key token cannot be overwritten by an AES key token, and an AES token cannot be overwritten by an HMAC token.

Required hardware

No cryptographic hardware is required by this callable service.

Coordinated KDS Administration (CSFCRC and CSFCRC6)

Use the coordinated KDS administration callable service to perform a coordinated KDS refresh or a coordinated KDS master key change or a conversion to a KDS capable of reference date tracking (KDSR format).

Coordinated KDS refresh is only supported for the CKDS and PKDS. Coordinated KDS refresh is not supported for TKDS.

When used for master key change or conversion, applications can continue to run KDS update workloads in parallel, and ICSF guarantees that any dynamic updates will be reflected in the target data set. For coordinated KDS refresh, you should disable KDS update workloads when refreshing to a target data set that is different from the currently-active KDS. This is recommended, because updates occurring to the currently-active KDS might not be reflected in the target data set. ICSF does not enforce manual disablement of dynamic KDS updates prior to a coordinated refresh operation, and will itself internally suspend such updates until the coordinated refresh operation completes. Note that the recommendation to disable KDS updates does not apply to a coordinated refresh when the target data set is the same as the currently-active KDS. In this case, the updates to the currently-active KDS are guaranteed to be in the resulting in-storage KDS when the operation completes.

In a sysplex environment, this callable service enables an application to perform a coordinated sysplex-wide KDS refresh, KDS conversion or KDS change master key operation from a single ICSF instance.

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 and reason codes," on page 943 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 and reason codes," on page 943 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.

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.

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.

Table 292. Coordinated KDS administration required hardware

Server	Required cryptographic hardware for Coordinated Change Master Key	Required cryptographic hardware for Coordinated Refresh and Conversion	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor or Crypto Express2 Coprocessor	None.	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	None.	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor or Crypto Express3 Coprocessor	None.	
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	None.	
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor, Crypto Express4 CCA Coprocessor, or Crypto Express4 Enterprise PKCS #11 coprocessor	None.	

Table 292. Coordinated KDS administration required hardware (continued)

Server	Required cryptographic hardware for Coordinated Change Master Key	Required cryptographic hardware for Coordinated Refresh and Conversion	Restrictions
IBM z13	Crypto Express5 CCA Coprocessor or Crypto Express5 Enterprise PKCS #11 coprocessor	None.	

ICSF Multi-Purpose Service (CSFMPS and CSFMPS6)

Use the ICSF multi-purpose callable service 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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 2.

rule_array

Direction	Type
Ignored	String

This field is ignored.

Table 293. Keywords for ICSF multi-purpose service

Keyword	Meaning
Key data set (Required)	
CKDS	Specifies that the active CKDS is to be used.
PKDS	Specifies that the active PKDS is to be used.
Operation (Required)	
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

This field is ignored. It is recommended to specify 0 for this parameter.

reserved1

Direction	Type
Output	String

This field is ignored.

reserved2_length

Direction	Type
Input	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

reserved2

Direction	Type
Output	String

This field is ignored.

reserved3_length

Direction	Type
Input	Integer

This field is ignored. It is recommended to specify 0 for this parameter.

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.

Table 294. ICSF multi-purpose service required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	None.	The validate rule is not supported.
IBM System z9 EC IBM System z9 BC	None.	The validate rule is not supported.
IBM System z10 EC IBM System z10 BC	None.	The validate rule is not supported.
IBM zEnterprise 196 IBM zEnterprise 114	None.	The validate rule is not supported.

Table 294. ICSF multi-purpose service required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12 IBM zEnterprise BC12	None.	The validate rule is not supported.
IBM z13	None.	The validate rule requires the Nov. 2014 or later licensed internal code (LIC).

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.

The KDSR format of the key data sets (introduced in HCR77A1) contains metadata that can be used for the search criteria. The older key data set formats only have 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 53 to convert your key data sets to KDSR 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. Note that only PKCS #11 objects have metadata. While the token has a record in the TKDS, it does not have metadata.

Tokens in the TKDS cannot be listed 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 and reason codes," on page 943 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 and reason codes," on page 943 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 3.

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 295. Keywords for KDS list control information

Keyword	Meaning
Key data set (Required)	
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.
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.
State of record (Required)	
ACTIVE	Specifies that only records that are not archived and within the start/end dates if specified will be checked. Note that if the start and end dates have not be 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 will be checked.
ARCHIVED	Specifies that only records that have been archived will be checked.
ALL	Specifies that all records will be 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 may 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 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, this 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 may 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 as well as 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 shown in Table 296.

Table 296. 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.
2	2	Length of structure. The length includes the search criterion and this length field.
4		Additional information (Required): • For dates, see Table 297 on page 782. • For metadata tag, see Table 298 on page 782. • For TKDS object type, see Table 299 on page 783. • For CKDS key type, see Table 300 on page 783. • For metadata flags, see Table 301 on page 785.

- 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 the date has not been set.
- When searching for dates equal to zero, the record will match 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 will match only if there is a non-zero date for that type in the record.
- Searching for dates being less than or equal to zero will be treated in the same manner as a request for dates equal to zero.

Table 297. 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.
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 298. 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'-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 299. 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 300. 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 300. 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): **Value (EBCDIC)** **Meaning** **ADATA** DES ANSI X9.17 DATA keys. **AKEK** DES ANSI X9.17 key-encrypting keys. **CIPHER** AES CIPHER keys. **CV** Any of the following DES key types: CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR, DATAC, DATAM, DATAMV, DECIPHER, DKYGENKY, ENCIPHER, IKEYXLAT, KEYGENKY, OKEYXLAT, SECMSG. **DATA** AES and DES DATA keys (encrypted and clear). **DATAXLAT** DES data-translating keys. **DKYGENKY** AES DKYGENKY. **EXPORTER** AES and DES exporter keys. **IMPORTER** AES and DES importer keys. **IMP-PKA** DES limit authority importer keys. **IPINENC** DES input PIN encrypting 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. **OPINENC** DES output PIN encrypting keys. **PINGEN** DES PIN generation keys. **PINCALC** AES PIN calculation keys. **PINPROT** AES PIN protection keys. **PINPRW** AES PIN reference value keys.

Table 300. Search criteria with CKDS key type (continued)

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
4	8		**PINVER** DES PIN verification keys.

To use a metadata flag as a search criterion, specify the flag and value to be used.

Table 301. 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.
4	1	1	Flag type: **Value** **Date** **X'01'** Prohibit archive flag.
5	1		CKDS key type in the label (bytes 65-72): **Value (Decimal)** **State of the flag** **1** Enabled. **0** Disabled.

`label_count`

Direction	Type
Output	Integer

The number of labels or handles that are found that match the search criteria. This 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 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. 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

The number of labels 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 3033. The *continuation_area* is used to continue the list for subsequent calls.

This parameter is only returned when the LABELS keyword is specified in the rule array.

Note: The CKDS label consist 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 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 in order to get the complete list of all labels that meet the search criteria. Your application must not change the data in this string.

For the first request, initialize this area to binary zero. The return code will indicate whether the list is complete.

This parameter is ignored when the COUNT keyword is specified in the rule array.

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 not be listed by this service.

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 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 4 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.

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 dates 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 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 the KDSR format:

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.

Variable-length metadata blocks
Both IBM and installation tags may be specified.

Note: See "Coordinated KDS Administration callable service (CSFCRC and CSFCRC6)" on page 53 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 and reason codes," on page 943 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 and reason codes,” on page 943 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 302. 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 consist 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 303.

Table 303. 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.

Table 303. Metadata entry (continued)

Offset (Decimal)	Number of bytes	Description
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 304. 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'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 305. 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 306. 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 307. 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.

Table 307. Output structure for flags (continued)

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
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 format of the key data set must be KDSR. See "Coordinated KDS Administration callable service (CSFCRC and CSFCRC6)" on page 53 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 and reason codes," on page 943 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 and reason codes," on page 943 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 308. Keywords for KDS metadata write control information

Keyword	Meaning
Key data set (Required)	
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 consist 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 309.

Table 309. 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. • See Table 310. • See Table 311 on page 799. • See Table 312 on page 800.

- 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 310. 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.

Table 310. Structure for variable metadata block (continued)

Offset (Decimal)	Number of bytes	Value (Decimal)	Description
4	2		Tag: **Value** Meaning **X'0001'** Installation user data. **X'8000'-X'FFFF'** Installation metadata.
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 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 311. 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 312. 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.
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.

PKDS Key Record Create (CSNDKRC and CSNFKRC)

This callable service writes a new record to 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 and reason codes," on page 943 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 and reason codes," on page 943 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 RSA or DSS token.

token

Direction	Type
Input	String

Data to be written to the PKDS if *token_length* is non-zero. An RSA, DSS, or ECC private token in either external or internal format, or a DSS, RSA, or ECC public token.

Usage notes

To use this service, PKA callable services must be enabled for all RSA and DSS token types. For systems with CEX3C or later coprocessors, 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 clear key ECC tokens, a current ECC master key is not required.

To use this service for encrypted key ECC tokens, the ECC master key must be valid.

Required hardware

No cryptographic hardware is required by this callable service.

PKDS Key Record Delete (CSNDKRD and CSNFKRD)

Use PKDS key record delete to delete a record from the PKDS.

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 313. 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.
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.

Usage notes

To use this service, PKA callable services must be enabled for all RSA and DSS token types. For systems with CEX3C or later coprocessors, 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 clear key ECC tokens, a current ECC master key is not required.

To use this service for encrypted key ECC tokens, the ECC master key must be valid.

Required hardware

No cryptographic hardware is required by this callable service.

PKDS Key Record Read (CSNDKRR and CSNFKRR)

Reads a record from the PKDS and returns the content of the record. This is true even when the record contains a null PKA token.

The callable service name for AMODE(64) invocation is CSNFKRR.

Format

```
CALL CSNDKRR(
            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 and reason codes," on page 943 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 and reason codes," on page 943 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 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 record.

Usage notes

To use this service, PKA callable services must be enabled for all RSA and DSS token types. For systems with CEX3C or later coprocessors, 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 clear key ECC tokens, a current ECC master key is not required.

To use this service for encrypted key ECC tokens, the ECC master key must be valid.

Required hardware

No cryptographic hardware is required by this callable service.

PKDS Key Record Write (CSNDKRW and CSNFKRW)

Writes over an existing record in 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 314. Keywords for PKDS Key Record Write

Keyword	Meaning
Write Mode (optional) specifies the circumstances under which the record is to be written.	
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 a DSS, RSA, or ECC private token in either external or internal format, or a DSS, RSA, or ECC public token.

Restrictions

This service cannot update a PKDS record for a retained key.

Usage notes

The PKDS Key Record Write service will only overwrite NULL tokens and tokens of the same type. For example an RSA token cannot overwrite an ECC or DSS token.

To use this service, PKA callable services must be enabled for all RSA and DSA token types. For systems with CEX3C or later coprocessors, 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 clear key ECC tokens, a current ECC master key is not required.

To use this service for encrypted key ECC tokens, the ECC master key must be valid.

Required hardware

No cryptographic hardware is required by this callable service.

Chapter 13. Utilities

This topic describes these callable services:
- "Character/Nibble Conversion (CSNBXBC and CSNBXCB)"
- "Code Conversion (CSNBXEA and CSNBXAE)" on page 813
- "ICSF Query Algorithm (CSFIQA and CSFIQA6)" on page 815
- "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 819
- "ICSF Query Facility2 (CSFIQF2 and CSFIQF26)" on page 843
- "SAF ACEE Selection (CSFACEE and CSFACEE6)" on page 846
- "X9.9 Data Editing (CSNB9ED)" on page 848

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 and reason codes," on page 943 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 and reason codes,” on page 943 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).

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 and reason codes," on page 943 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 and reason codes," on page 943 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 1119 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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 315. 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-PRIM	Elliptic Curve Cryptography using NIST approved PRIME curves
HMAC	FIPS-198 keyed-hash message authentication code 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 315. 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
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. Currently, the value must be large enough to handle the request. 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

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 316. 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-PRIM ECC-BP (Brain Pool) HMAC MDC-2 MDC-4 MD5 RNGL RPMD-160 RSA-GEN RSA-KM RSA-SIG SHA-1 SHA-2 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. For RNGL, the size is in bytes.
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.

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 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 and reason codes," on page 943 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 and reason codes," on page 943 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 317.

Table 317. Keywords for ICSF Query Service

Keyword	Meaning
Coprocessor (optional) - parameter is ignored for ICSFSTAT, ICSFST2, and ICSFSP11	
COPROCxx	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 a 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 a Enterprise PKCS #11 coprocessor. For all other rules, it must be that of a CCA coprocessor.
Information to return (required)	
ICSFSTAT	Get ICSF related status information.
ICSFST2	Get coprocessor-related basic status information.
ICSFSP11	Get ICSF-related PKCS #11 status information
NUM-DECT	Get the number of bytes of storage required for the output of a STATDECT request.
STATAES	Get status information on AES enablement and the AES master key registers.
STATCCA	Get CCA-related status information.
STATCCAE	Get CCA-related extended status information.

Table 317. Keywords for ICSF Query Service (continued)

Keyword	Meaning
STATCARD	Get coprocessor-related basic status information.
STATDECT	Get the PIN decimalization tables loaded. The format of the data is shown under the *returned_data* parameter. 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.
STATDIAG	Get coprocessor-related basic status information.
STATAPKA	Get status information on ECC enablement and the ECC master key registers.
STATEID	Get coprocessor-related basic status information.
STATEXPT	Get coprocessor-related basic status information.
WRAPMTHD	Get coprocessor-related default configuration setting for the wrapping method.
STATP11	Get Enterprise PKCS #11 coprocessor-related status information.
STATWPIN	Get the weak PIN table loaded. The format of the data is shown under the *returned_data* parameter. The table is up to 460 bytes long.
SIZEWPIN	Get the number of bytes of storage required for the output of a STATWPIN request.
Additional Master Key Information (optional) - rule is only allowed 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. Currently, the value must be at least eight times the number of elements returned for the *rule_array* 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 tables below, 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, the output is a 4-byte integer.

The output *returned_data* for the ICSFSTAT keyword is defined in Table 318.

Table 318. Output for option ICSFSTAT

Element Number	Name	Description
1	FMID	8-byte ICSF FMID
2	ICSF Status Field 1	Status of ICSF **Number** **Meaning** **0** ICSF started **1** ICSF initialized (CCVINIT is on) **2** SYM-MK (DES master key) valid (CCVTMK is on) **3** PKA callable services enabled (see Usage Notes)
3	ICSF Status Field 2	Status of ICSF **Number** **Meaning** **0** 64-bit callers not supported **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** **0** CPACF not available **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-384 and SHA-512, DES and TDES are available **7** Encrypted CPACF functions available. **8** OFB, CFB, and GCM CPACF functions are available. **9** DRNG is available.

Table 318. Output for option ICSFSTAT (continued)

Element Number	Name	Description
5	AES	AES availability for clear keys **Number** **Meaning** **0** AES not available **1** AES software only **2** AES-128 **3** AES-192 and AES-256
6	DSA	DSA algorithm availability **Number** **Meaning** **0** DSA not available **1** DSA 1024 key size **2** DSA 2048 key size
7	RSA Signature	RSA Signature key length **Number** **Meaning** **0** RSA not available **1** RSA 1024 key size **2** RSA 2048 key size **3** RSA 4096 key size
8	RSA Key Management	RSA Key Management key length **Number** **Meaning** **0** RSA not available **1** RSA 1024 key size **2** RSA 2048 key size **3** RSA 4096 key size
9	RSA Key Generate	RSA Key Generate **Number** **Meaning** **0** Service not available **1** Service available - 2048 bit modulus **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.

Table 318. Output for option ICSFSTAT (continued)

Element Number	Name	Description
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** **0** RSA-ME key size of 2048, CRT key size of 2048. **1** RSA-ME key size of 4096, CRT key size of 4096.
12	ICSF Status Field 3	An 8-byte numeric character string. 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 output *returned_data* for the ICSFSP11 keyword is defined in Table 319.

Table 319. 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 ICSFST2 keyword is defined in Table 320.

Table 320. Output for option ICSFST2

Element Number	Name	Description
1	Version	Version of the ICSFST2 returned_data. Initial value is 1. It covers elements 1 through 12.
2	FMID	8–byte ICSF FMID.
3	ICSF Status Field 1	Status of ICSF **Number** **Meaning** **0** PKA callable services disabled **1** PKA callable services enabled (see Usage Notes)
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** **0** ICSF started **1** ICSF initialized **2** AES master key valid
6	ICSF Status Field 4	Status of ICSF **Number** **Meaning** **0** Secure key AES not available **1** Secure key AES is available

Table 320. Output for option ICSFST2 (continued)

Element Number	Name	Description
7	ICSF Status Field 5	An 8-character numeric character string summarizing the current Key Store Policy. 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. 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.**

Table 320. Output for option ICSFST2 (continued)

Element Number	Name	Description
		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. 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.**

Table 320. Output for option ICSFST2 (continued)

Element Number	Name	Description
		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 320. 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, and, if so, 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** Symmetric Key Label Export controls are 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 first character of this string are: **Number** **Meaning** **0** Allow Archived Key Use control is disabled. **1** Allow Archived Key Use control is enabled.

Table 320. Output for option ICSFST2 (continued)

Element Number	Name	Description
8	ICSF Status Field 6	Status of ICSF **Number** **Meaning** **0** ICSF started **1** ICSF initialized **2** ECC master key valid, internal keys supported **3** ECC master key valid, external keys also supported
9	ICSF Status Field 7	Status of ICSF **Number** **Meaning** **0** ICSF started **1** ICSF initialized **2** RSA master key valid
10	ICSF Status Field 8	Status of ICSF **Number** **Meaning** **0** ICSF started **1** ICSF initialized **2** DES master key valid
11	ICSF Status Field 9	Status of ICSF **Number** **Meaning** **0** PKA callable services disabled. **1** PKA callable services enabled. See Usage Notes for additional information.
12	Reserved	Unpredictable

Table 321. Output for option NUM-DECT

Element Number	Description
1	The number of bytes required for the output of a STATDECT request. This is the number of decimalization tables loaded times 20 bytes. This is a four-byte binary number.

Table 322. 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.

Table 323. 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 323. 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	A character string that identifies the version of the CCA application program that is running in the coprocessor.
5	CCA Application Build Date	A character string containing the build date for the CCA application program that is running in the coprocessor.
6	User Role	A character string containing the Role identifier which defines the host application user's current authority.

Table 324. 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	A character string that identifies the version of the CCA application program that is running in the coprocessor.
5	CCA Application Build Date	A character string containing the build date for the CCA application program that is running in the coprocessor.
6	User Role	A character string containing the Role identifier which defines the host application user's current authority.

Table 324. Output for option STATCCAE (continued)

Element Number	Name	Description
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

Table 325. Output for option STATCARD

Element Number	Name	Description
1	Number of installed adapters	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	A numeric character string containing an integer value identifying the version of DES hardware that is on the coprocessor.
3	RSA hardware level	A numeric character string containing an integer value identifying the version of RSA hardware that is on the coprocessor.
4	POST Version	A character string identifying the version of the coprocessor's 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	A character string identifying the operating system firmware on the coprocessor. Padding characters are blanks.
6	Coprocessor Operating System Version	A character string identifying the version of the operating system firmware on the coprocessor.
7	Coprocessor Part Number	A character string containing the eight-character part number identifying the version of the coprocessor.
8	Coprocessor EC Level	A character string containing the eight-character EC (engineering change) level for this version of the coprocessor.

Table 325. Output for option STATCARD (continued)

Element Number	Name	Description
9	Miniboot Version	A character string identifying 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	A numeric character string containing the operating speed of the microprocessor chip, in megahertz.
11	Adapter ID (Also see element number 15)	A unique identifier manufactured into the coprocessor. The coprocessor's Adapter ID is an eight-byte binary value.
12	Flash Memory Size	A numeric character string containing the size of the flash EPROM memory on the coprocessor, in 64-kilobyte increments.
13	DRAM Memory Size	A numeric character string containing the size of the dynamic RAM (DRAM) on the coprocessor, in kilobytes.
14	Battery-Backed Memory Size	A numeric character string containing the size of the battery-backed RAM on the coprocessor, in kilobytes.
15	Serial Number	A character string containing 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.

For STATDECT, the output is a table of up to 100 PIN decimalization tables as shown in the following table. The maximum size is 2000 bytes.

Table 326. 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

Table 327. Output for option STATDIAG

Element Number	Name	Description
1	Battery State	A numeric character string containing a value which indicates whether the battery on the coprocessor needs to be replaced: **Number** **Meaning** **1** Battery is good **2** Battery should be replaced

Table 327. Output for option STATDIAG (continued)

Element Number	Name	Description
2	Intrusion Latch State	A numeric character string containing a value which indicates whether the intrusion latch on the coprocessor is set or cleared: **Number** **Meaning** **1** Latch is cleared **2** Latch is set
3	Error Log Status	A numeric character string containing a value which indicates whether there is data in the coprocessor CCA error log. **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	A numeric character string containing a value to indicate whether the coprocessor has detected tampering with 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	A numeric character string containing a value to indicate whether a 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
6	High Voltage Detected	A numeric character string containing a value to indicate whether a 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

Table 327. Output for option STATDIAG (continued)

Element Number	Name	Description
7	Temperature Range Exceeded	A numeric character string containing a value to indicate 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	A numeric character string containing a value to indicate 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 rule-array 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 rule-array elements contain the SAPI return codes and reason codes corresponding to the five commands in rule-array 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.

Table 328. Output for option STATEID

Element Number	Name	Description
1	EID	During initialization, a value of zero is set in the coprocessor.

Table 329. Output for option STATEXPT

Element Number	Name	Description
1	Base CCA Services Availability	A numeric character string containing a value to indicate whether base CCA services are available. **Number** **Meaning** **0** Base CCA services are not available **1** Base CCA services are available
2	CDMF Availability	A numeric character string containing a value to indicate whether CDMF is available. **Number** **Meaning** **0** CDMF encryption is not available
3	56-bit DES Availability	A numeric character string containing a value to indicate whether 56-bit DES encryption is available. **Number** **Meaning** **0** 56-bit DES encryption is not available **1** 56-bit DES encryption is available
4	Triple-DES Availability	A numeric character string containing a value to indicate whether triple-DES encryption is available. **Number** **Meaning** **0** Triple-DES encryption is not available **1** Triple-DES encryption is available
5	SET Services Availability	A numeric character string containing a value to indicate whether SET (Secure Electronic Transaction) services are available. **Number** **Meaning** **0** SET Services are not available **1** SET Services are available

Table 329. Output for option STATEXPT (continued)

Element Number	Name	Description
6	Maximum Modulus for Symmetric Key Encryption	A numeric character string containing the 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

Table 330. 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.
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.

Table 331. 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
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

Table 332. 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	Status State of the P11 current master key register: **Number** **Meaning** **1** Register is clear **2** Register contains a key
3	Compliance Mode	Current compliance mode for the coprocessor. An 8-byte hexadecimal number that is the sum of the active compliance modes: **Number** **Meaning** **n** An 8-byte hexadecimal number that is the sum of the active compliance modes: • 1 - FIPS 2009 • 2 - BSI 2009 • 4 - FIPS 2011 • 8 - BSI 2011

Table 332. Output for option STATP11 (continued)

Element Number	Name	Description
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 – 12	Future use	Currently blanks

Table 333. Output for option SIZEWPIN

Description
The number of bytes of storage required for the output of a STATWPIN request. The value is a 4-byte binary number.

For STATWPIN, the output is a table of up to 20 weak PINs. Each entry in the table is formatted as shown in the following table. The maximum size is 460 bytes. The data in the table in character format (EBCDIC).

Table 334. 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

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 the PKA callable services are enabled and there is at least one ACTIVE coprocessor.

The options ICSFSTAT and ICSFST2 report on the state of PKA callable services. ICSFSTAT reports it in element 2. ICSFST2 reports it in elements 3 and 11. There is a subtle difference between the three options. ICSFSTAT reports PKA callable services as enabled only after the DES master key is loaded and valid. ICSFSTAT does not report PKA callable services as enabled when only the AES master key is loaded and valid. Option ICSFST2 element 3 reports PKA callable services as enabled when the DES and/or AES master key is loaded and valid. Option ICSFST2 element 11 reports PKA callable services as enabled when neither the DES nor AES master keys are loaded and valid.

Note: If your system has CEX3C or later coprocessors, the PKA callable services control may not be available. The PKA callable services state will be the same as the RSA master key. If the RSA master key is active, the PKA callable services will be enabled in the ICSFSTAT and ICSFST2 reports.

Required hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 335. ICSF Query Service required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	None.	
IBM System z9 EC IBM System z9 BC	None.	
IBM System z10 EC IBM System z10 BC	None.	
IBM zEnterprise 196 IBM zEnterprise 114	None.	
IBM zEnterprise EC12 IBM zEnterprise BC12	None.	
IBM z13	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 and reason codes," on page 943 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 and reason codes," on page 943 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.

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 336.

Table 336. Format of returned ICSF Query Facility 2 data

Bytes	Description
0-7	ICSF FMID
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 336. 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** RESERVED **7** RESERVED

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 846 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 and reason codes," on page 943 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 and reason codes," on page 943 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 14. Trusted Key Entry Workstation Interfaces

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.

- On the CEX3C, all operational keys may be loaded with TKE 6.0 or higher.
- On the CEX4C, all operational keys may be loaded with TKE 7.2 or higher.
- On the CEX5C, all operational keys may be loaded with TKE 8.0 or higher.
- DES and RSA master keys are available on all CCA coprocessors.
- The AES master key and AES operational keys are supported on the z9 and newer systems with the Nov. 2008 or later licensed internal code (LIC)
- The ECC master key is supported on the z196 and newer systems with the Sep. 2011 or later licensed internal code.
- The P11 master key is supported on Enterprise PKCS #11 coprocessors.

This topic describes these callable services:

- "PCI Interface Callable Service (CSFPCI and CSFPCI6)"

PCI Interface Callable Service (CSFPCI and CSFPCI6)

TKE uses this callable service to send a request to a specific PCI card queue and remove the corresponding response when complete. This service also allows the TKE workstation to query the list of access control points which may be enabled or disabled by a TKE user. This service is synchronous. The return and reason codes reflect the success or failure of the queue functions rather than the success or failure of the actual PCI request.

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 and reason codes," on page 943, 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 and reason codes," on page 943 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 337. 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.
CX2MASK	This keyword is a request to return both the 64-bit mask indicating which of the CEX2Cs are online and the 64-bit mask indicating which of the CEX2Cs are active. See the *masks_data* parameter description for more information.
CX3MASK	This keyword is a request to return both the 64-bit mask indicating which of the CEX3Cs are online and the 64-bit mask indicating which of the CEX3Cs are active. See the *masks_data* parameter description for more information.
CX4MASK	This keyword is a request to return both the 64-bit mask indicating which of the Crypto Express4 coprocessors are online and the 64-bit mask indicating which of the Crypto Express4 coprocessors are active. Only the coprocessors configured for CCA are to be examined. See the *masks_data* parameter description for more information.
PDECRYPT	This keyword is a request to decrypt a cryptogram received from TKE using the specified secret key, returning the clear 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 AP configuration of the system. Q-APINFO may be used to replace the coprocessor mask keywords as well as the QUERYDOM keyword because the returned data includes the information returned by those keywords.
QUERYDOM	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 a request to return both the 64-bit mask indicating which of the PCIXCCs and Crypto Express coprocessors are online and the 64-bit mask indicating which of the PCIXCCs and Crypto Express 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.
XPPOINTS	Queries the list of PKCS #11 access control points which may be enabled or disabled by a TKE user.
XP4MASK	This keyword is a request to return both the 64-bit mask indicating which of the Crypto Express4 coprocessors are online and the 64-bit mask indicating which of the Crypto Express4 coprocessors are active. Only the coprocessors configured for PKCS #11 are to be examined. See the *masks_data* parameter description for more information.
Reason (optional)	

Table 337. Keywords for PCI interface callable service (continued)

Keyword	Meaning
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, CX2MASK, CX3MASK CX4MASK, XP4MASK, Q-APINFO, 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 PKEYGEN or PDECRYPT 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 64.

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. 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. This field is updated on output with the actual length of the *reply_data_block* field. This length field must be a multiple of 4. For the XPNUM rule, the length must be zero. For the XPPOINTS keyword, the minimum length is 2206 bytes. For the ACPOINTS keyword, the minimum length is 17469 bytes.

reply_data_block

Direction	Type
Output	String

The data that accompanies the *reply_block* field.

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, CX2MASK, CX3MASK, CX4MASK, XP4MASK, or QUERYDOM. For all other rule_array keywords, hex 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.

Table 338. PCI Interface required hardware

Server	Required cryptographic hardware	Restrictions
IBM eServer zSeries 990 IBM eServer zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	The following rules are not supported: • Q-APINFO • QUERYDOM
IBM System z9 EC IBM System z9 BC	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	The following rules are not supported: • Q-APINFO • QUERYDOM
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	The following rules are not supported: • Q-APINFO • QUERYDOM
IBM zEnterprise 196 IBM zEnterprise 114	Crypto Express3 Coprocessor	

Table 338. PCI Interface required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12 IBM zEnterprise BC12	Crypto Express3 Coprocessor Crypto Express4 CCA Coprocessor Crypto Express4 Enterprise PKCS #11 Coprocessor	
IBM z13	Crypto Express5 CCA Coprocessor Crypto Express5 Enterprise PKCS #11 Coprocessor	

Part 3. PKCS #11 Callable Services

Chapter 15. 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)”
- “PKCS #11 Derive key (CSFPDVK and CSFPDVK6)” on page 869
- “PKCS #11 Get attribute value (CSFPGAV and CSFPGAV6)” on page 876
- “PKCS #11 Generate key pair (CSFPGKP and CSFPGKP6)” on page 878
- “PKCS #11 Generate secret key (CSFPGSK and CSFPGSK6)” on page 881
- “PKCS #11 Generate HMAC (CSFPHMG and CSFPHMG6)” on page 884
- “PKCS #11 Verify HMAC (CSFPHMV and CSFPHMV6)” on page 888
- “PKCS #11 One-way hash, sign, or verify (CSFPOWH and CSFPOWH6)” on page 892
- “PKCS #11 Private key sign (CSFPPKS and CSFPPKS6)” on page 899
- “PKCS #11 Public key verify (CSFPPKV and CSFPPKV6)” on page 902
- “PKCS #11 Pseudo-random function (CSFPPRF and CSFPPRF6)” on page 904
- “PKCS #11 Set attribute value (CSFPSAV and CSFPSAV6)” on page 908
- “PKCS #11 Secret key decrypt (CSFPSKD and CSFPSKD6)” on page 910
- “PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6)” on page 915
- “PKCS #11 Token record create (CSFPTRC and CSFPTRC6)” on page 922
- “PKCS #11 Token record delete (CSFPTRD and CSFPTRD6)” on page 925
- “PKCS #11 Token record list (CSFPTRL and CSFPTRL6)” on page 928
- “PKCS #11 Unwrap key (CSFPUWK and CSFPUWK6)” on page 932
- “PKCS #11 Wrap key (CSFPWPK and CSFPWPK6)” on page 936

The metadata of objects in the TKDS can be managed using the following services:

- “Key Data Set List (CSFKDSL and CSFKDSL6)” on page 778
- “Key Data Set Metadata Read (CSFKDMR and CSFKDMR6)” on page 788
- “Key Data Set Metadata Write (CSFKDMW and CSFKDMW6)” on page 795

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.

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 339. 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

Table 339. Keywords for derive multiple keys (continued)

Keyword	Meaning
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
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

Table 339. Keywords for derive multiple keys (continued)

Keyword	Meaning
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 97 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 97 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 340. 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
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 341. 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 342. 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
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 343. 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.

Table 343. parms_list parameter format for IKE1PHA2 and IKE2PHA2 mechanisms (continued)

Offset	Length in bytes	Direction	Description
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
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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 344. Keywords for derive key

Keyword	Meaning
Mechanism (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
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

Table 344. Keywords for derive key (continued)

Keyword	Meaning
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
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 97 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 97 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 345. 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 346. 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	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	x	Input	Client's random data
12+x	y	Input	Server's random data

Table 347. parms_list parameter format for EC-DH mechanism

Offset	Length in bytes	Direction	Description
0	1	Input	KDF function code, x'01' = NULL; x'02' = SHA1. x'05' = SHA224, x'06' = SHA256, x'07' = SHA384, and x'08' = SHA512
1	3	Not applicable	Reserved
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, the maximum shared data length 2147483647.
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 347. 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)
20	x<=136	Input	Binary value representing the other party’s public value with or without DER encoding.

Table 348. 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

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

The service does not support the derivation of secure keys. For rules EC-DH and PKCS-DH 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. All other key derivation operations are performed in software.

Key derivation functions 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=l*ength 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.

PKCS #11 Get attribute value (CSFPGAV and CSFPGAV6)

Use the 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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 97 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
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.

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 97 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 349. 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 generate key pair callable service to generate an RSA, DSA, Elliptic Curve, or Diffie-Hellman 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 and reason codes," on page 943 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 and reason codes," on page 943 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 97 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
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

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 97 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 97 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.

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 and reason codes,” on page 943 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 and reason codes,” on page 943 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 97 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 350. Keywords for generate secret key

Keyword	Meaning
Mechanism (One of the following must be specified)	
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)

Table 350. Keywords for generate secret key (continued)

Keyword	Meaning
KEY	Generate a secret key object according to the key type attribute in the *attribute_list* parameter (for CKM_GENERIC_SECRET_KEY_GEN, CKM_DES_KEY_GEN, CKM_DES2_KEY_GEN, CKM_DES3_KEY_GEN, CKM_AES_KEY_GEN, CKM_RC4_KEY_GEN, and CKM_BLOWFISH_KEY_GEN)
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)
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)

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 97 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 351. parms_list parameter format for SSL and TLS mechanism

Offset	Length in bytes	Direction	Description
0	2	input	SSL or TLS version number in binary, e.g., for version 3.01 this would be x'0301'

Table 352. 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

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 a 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.

Rule PBEKEY requires a z890/990 or later machine type.

PKCS #11 Generate HMAC (CSFPHMG and CSFPHMG6)

Use the PKCS #11 Generate HMAC callable service to generate a hashed message authentication code (MAC). This service does not support any recovery methods.

The key handle must be a handle of a PKCS #11 generic secret key object. The mechanism keyword specified in the rule array indicates the hash 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 and reason codes," on page 943 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 and reason codes," on page 943 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 353. Keywords for generate HMAC

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.
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.
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.
Chaining Selection (Optional)	
FIRST	Specifies this is the first call in a series of chained calls. Intermediate results are stored in the *hash* field.
MIDDLE	Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the *hash* field.
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 354. 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. See "Handles" on page 97 for the format of a *key_handle*.

hmac_length

Direction	Type
Ignored	Integer

Reserved field

hmac

Direction	Type
Output	String

Upon successful completion of an ONLY or LAST request, this field contains the generated HMAC value, left justified. The caller must provide an area large enough to hold the generated HMAC as defined by the mechanism specified. 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.

PKCS #11 Verify HMAC (CSFPHMV and CSFPHMV6)

Use the PKCS #11 Verify HMAC callable service to verify a hash message authentication code (MAC). This service does not support any recovery methods.

The key handle must be a handle of a PKCS #11 generic secret key object. The mechanism keyword specified in the rule array indicates the hash 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 and reason codes,” on page 943 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 and reason codes," on page 943 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 355. Keywords for verify HMAC

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.
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.

Table 355. Keywords for verify HMAC (continued)

Keyword	Meaning
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.
Chaining Selection (Optional)	
FIRST	Specifies this is the first call in a series of chained calls. Intermediate results are stored in the hash field.
MIDDLE	Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the hash field.
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 356. 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. See "Handles" on page 97 for the format of a *key_handle*.

hmac_length

Direction	Type
Ignored	Integer

Reserved field

hmac

Direction	Type
Input	String

This field contains the HMAC value to be verified on ONLY and LAST requests, left justified. The caller must provide an HMAC value of the required length as determined by the mechanism specified. 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 didn't 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.

PKCS #11 One-way hash, sign, or verify (CSFPOWH and CSFPOWH6)

Use the 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, or ECDSA
- 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 and reason codes," on page 943 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 and reason codes," on page 943 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 357. Keywords for one-way hash generate

Keyword	Meaning
Hash Method (required)	

Table 357. Keywords for one-way hash generate (continued)

Keyword	Meaning
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.
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.

Table 357. Keywords for one-way hash generate (continued)

Keyword	Meaning
NULL	For use with non-chained signature generate and verifies only. Hashing is not to be performed. 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.
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-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.
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. If not NULL, the hash method must match the chain_data Hash method.
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.
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.
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. If not NULL, the Hash method must match the chain_data Hash method.
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.

text_length

Direction	Type
Input	Integer

The length of the text parameter in bytes.

If you specify the FIRST or MIDDLE keyword, then 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 was specified then 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 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 358. 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)

Table 358. chain_data parameter format on input (FIRST and ONLY for SIGN-PSS and VER-PSS) (continued)

Offset	Length	Initialization values provided by the caller
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.
8	4	PSS salt length in bytes. For SIGN-PSS 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 SIGN-VER, 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 359. 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 360. 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 97 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. Even though the length of the SHA-224 hash is less than SHA-256, the extra bytes are used as a work area during the generation of the hash value. The SHA-224 value is left-justified and padded with 4 bytes of binary zeroes. For SHA-384 and SHA-512, thus must be at least 64 bytes. Even though the length of the SHA-384 hash is less than SHA-512, the extra bytes are used as a work area during the generation of the hash value. The SHA-384 value is left-justified 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

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 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 and reason codes," on page 943 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 and reason codes," on page 943 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 361. Keywords for private key sign

Keyword	Meaning
Mechanism (One of the following must be specified)	
RSA-ZERO	Mechanism is RSA decryption or signature generation using zero-pad formatting

Table 361. Keywords for private key sign (continued)

Keyword	Meaning
RSA-PKCS	Mechanism is RSA decryption or signature generation using PKCS #1 v1.5 formatting
DSA	Mechanism is DSA signature generation
ECDSA	Mechanism is Elliptic Curve with DSA signature generation
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.

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 RSA-PKCS signature requests, the data to be signed is expected to be a DER encoded DigestInfo structure. For DSA and ECDSA signature requests, the data to be signed is expected to be a SHA1, SHA224, SHA256, SHA384 or SHA512 digest.

key_handle

Direction	Type
Input	String

The 44-byte handle of a private key object. See “Handles” on page 97 for the format of a *key_handle*.

clear_value_length

Direction	Type
Input/Output	Integer

Length of the *clear_value* parameter in bytes. 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

DSA, ECDSA and RSA operations may be done in hardware or software.

Request type DECRYPT is not supported for an Elliptic Curve or DSA private 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 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 and reason codes," on page 943 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 and reason codes," on page 943 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 362. Keywords for public key verify

Keyword	Meaning
Mechanism (One of the following must be specified)	
RSA-ZERO	Mechanism is RSA encryption or signature verification using zero-pad formatting
RSA-PKCS	Mechanism is RSA encryption or signature verification using PKCS #1 v1.5 formatting
DSA	Mechanism is DSA signature verification
ECDSA	Mechanism is Elliptic Curve with DSA signature verification
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.

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 is be verified.

key_handle

Direction	Type
Input	String

The 44-byte handle of public key object. See "Handles" on page 97 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/Output	String

For encrypt, this is the encrypted value (output only). For signature verification, this is the data to be verified (input only). For RSA-PKCS signature verification requests, the data to be verified is expected to be a DER encoded DigestInfo structure. For DSA and ECDSA signature verification requests, the data to be verified is expected to be a SHA1, SHA224, SHA256, SHA384 or SHA512 digest.

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

DSA, ECDSA, and RSA operations may be done in hardware or software.

Request type ENCRYPT is not supported for an Elliptic Curve or DSA public 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 and reason codes," on page 943 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 and reason codes," on page 943 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 363. 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, 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.
PRNGFIPS	Generate pseudo-random bytes using the best source available. Possible sources are: Enterprise PKCS #11 coprocessors or a pseudo (deterministic) random algorithm, consistent with NIST SP 800-90. PRNGFIPS allows the caller to demand FIPS processing, in which case CCA coprocessors are only used for entropy seeding.

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 97 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 364. 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 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 and reason codes," on page 943 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 and reason codes," on page 943 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 97 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
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.

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 97 for the format of an *attribute_list*.

Authorization

Table 365. 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 clear symmetric key. AES, DES, BLOWFISH, and RC4 are supported. This service supports CBC, CTR, ECB, Galois/Counter, and stream modes and 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 and reason codes," on page 943 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 and reason codes," on page 943 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 366. 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.
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 DES, DES3, AES, and BLOWFISH. CBC cannot be specified for RC4.

Table 366. Keywords for secret key decrypt (continued)

Keyword	Meaning
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 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 encryption. The cipher text length must be a multiple of the block size for the specified algorithm. ECB cannot be specified for BLOWFISH or RC4.
GCM	Performs Galois/Counter mode encryption. 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 BLOWFISH, DES, DES3, or AES. STREAM is the default value for 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 rule ECB or GCM.
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.
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.
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 counter mode and Galois Counter mode, 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. See “Handles” on page 97 for the format of a *key_handle*.

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 CTR, this must be the size of the structure described in the *initialization_vector* parameter (17 bytes). For GCM, this must be the size of the *initialization_vector* field (28 bytes).

initialization_vector

Direction	Type
Input	String

This field has a varying format depending on the mechanism specified. For CBC and CBC-PAD this is the 8 or 16 byte initial chaining value. The formats for GCM and CTR are shown in the following tables.

Table 367. initialization_vector parameter format for GCM mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Length in bytes of the initialization vector. The minimum value is 1. The maximum value is 128. 12 is recommended.
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. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

Table 368. 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. 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 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 369. 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, the length can be up to 2147483647. For processing rule GCM, the length cannot exceed 1048576 plus the length of the tag.

cipher_text

Direction	Type
Input	String

Text to be decrypted.

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 CTR or GCM.

PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6)

Use the PKCS #11 secret key encrypt callable service to encipher data using a clear symmetric key. AES, DES, BLOWFISH, and RC4 are supported. This service
| supports CBC, CTR, ECB, Galois/Counter, and stream modes and 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 and reason codes," on page 943 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 and reason codes," on page 943 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 370. 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.
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 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 DES, DES3, AES, and BLOWFISH. CBC cannot be specified for RC4.
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 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 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.
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 BLOWFISH, DES, DES3, or AES. STREAM is the default value for 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. Intermediate results are stored in the chain_data field. Cannot be specified with processing rule ECB, GCM, or GCMIVGEN.
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.

Table 370. Keywords for secret key encrypt (continued)

Keyword	Meaning
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.
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 counter mode and Galois Counter mode, 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. See "Handles" on page 97 for the format of a *key_handle*.

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 CTR, this must be the size of the structure described in the *initialization_vector* parameter (17 bytes). For GCM and GCMVGEN, this must be the size of the *initialization_vector* field (28 bytes).

initialization_vector

Direction	Type
Input	String

This field has a varying format depending on the mechanism specified. For CBC and CBC-PAD this is the 8 or 16 byte initial chaining value. The formats for GCM, GCMIVGEN, and CTR are shown in the following tables.

Table 371. initialization_vector parameter format for GCM mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	length in bytes of the initialization vector area. The minimum value is 1. The maximum value is 128. 12 is recommended.
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. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

Table 372. 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 373. 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. 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 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 374. 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

Table 374. chain_data parameter format (continued)

Offset	Length	Description
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.

cipher_text

Direction	Type
Output	String

Encrypted 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 GCMVGEN. 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 CTR or GCM.

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

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 and reason codes," on page 943 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 and reason codes," on page 943 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 name of the z/OS PKCS #11 token to be initialized, or the token handle of the object to be created or copied. For the create or re-create functions, the first 32 bytes of the handle are meaningful on input. The remaining 12 bytes are filled in by the token record create service. For the copy function, all 44 bytes of the handle are significant on input.

On output, the 44-byte handle of the z/OS PKCS #11 token or object created.

See “Handles” on page 97 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-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 375. Token record create keywords

Keyword	Meaning
One of these two 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.
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 97 for the format of an *attribute_list*.

Authorization

Note: Session and token objects require the same SAF authority.

Table 376. 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)

Table 376. Authorization requirements for the token record create callable service (continued)

Action	Source object (Copy only)	Token / Object being created	PKCS #11 role Authority required
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

HMAC operations are performed in hardware or software.

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.

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 and reason codes," on page 943 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 and reason codes," on page 943 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 97 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-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 377. 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 378. 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 910 CSFPSKD
- “PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6)” on page 915 CSFPSKE
- “PKCS #11 One-way hash, sign, or verify (CSFPOWH and CSFPOWH6)” on page 892 CSFPOWH
- “PKCS #11 Generate HMAC (CSFPHMG and CSFPHMG6)” on page 884 CSFPHMG
- “PKCS #11 Verify HMAC (CSFPHMV and CSFPHMV6)” on page 888 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 and reason codes," on page 943 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 and reason codes," on page 943 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 97 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-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 379. 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)	
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.

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 97 for the format of an *attribute_list*.

list_length

Direction	Type
Input	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

Bytes	Description
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 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) 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
  - The unwrapping key must be a DES, DES2, DES3, or 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 and reason codes," on page 943 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 and reason codes," on page 943 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 380. 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	RSA PKCS #1 block type 02 will be used to recover the key value.
PKCS-8	The private key values are DER encoded as specified by PKCS-8. The encryption mechanism rule array keyword must be specified.
Encryption Mechanism (required when PKCS-8 specified, ignored otherwise)	
AES	For PKCS-8 processing, the unwrapping key must be an AES secret key object.
DES	For PKCS-8 processing, the unwrapping key must be a DES secret key object.
DES3	For PKCS-8 processing, the unwrapping key must be a DES2 or DES3 secret key object.

wrapped_key_length

Direction	Type
Input	Integer

Length of the wrapped key in the *wrapped_key* parameter.

wrapped_key

Direction	Type
Input	String

The key to be unwrapped.

initialization_vector_length

Direction	Type
Input	Integer

The length of the *initialization_vector* parameter. The initial value can only be used with PKCS-8 or ATTRBND. This parameter is ignored for PKCS-1.2. For PKCS-8, 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.

initialization_vector

Direction	Type
Input	String

For formatting method PKCS-8, 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.

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 97 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 97 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

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) is supported for wrapping a private or secret key with a secret key.
  – The wrapping key must be a DES, DES2, DES3, or 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 and reason codes," on page 943 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 and reason codes," on page 943 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 381. 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	RSA PKCS #1 block type 02 will be used to format the key value.
PKCS-8	The private key values are DER encoded as specified by PKCS-8. The encryption mechanism rule array keyword must be specified.
Encryption Mechanism (required when PKCS-8 specified, ignored otherwise)	
AES	For PKCS-8 processing, the wrapping key must be an AES secret key object.
DES	For PKCS-8 processing, the wrapping key must be a DES secret key object.

Table 381. Keywords for wrap key (continued)

Keyword	Meaning
DES3	For PKCS-8 processing, the wrapping key must be a DES2 or DES3 secret key object.

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 97 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 or ATTRBND. This parameter is ignored for PKCS-1.2. For PKCS-8, 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.

Initialization_vector

Direction	Type
Input	String

For formatting method PKCS-8, 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.

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.

wrapped_key

Direction	Type
Output	String

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

Part 4. Appendixes

Appendix A. ICSF and cryptographic coprocessor return and 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 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 IBM 4765 PCIe and IBM 4764 PCI-X coprocessor return and reason codes have been merged with the ICSF codes in this release. 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.

Return codes

Table 382 lists return codes from the ICSF callable services.

Table 382. 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.

Table 382. Return Codes (continued)

Return Code Hex (Decimal)	Description
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 383 lists reason codes returned from callable services that give return code 0.

Table 383. 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 one (perhaps of many) 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 service 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)
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 one (perhaps of many) 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 service 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)

Table 383. Reason codes for return code 0 (0) (continued)

Reason Code Hex (Decimal)	Description
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.
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.

Reason codes for return code 4 (4)

Table 384 lists reason codes returned from callable services that give return code 4.

Table 384. 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).

Table 384. Reason codes for return code 4 (4) (continued)

Reason Code Hex (Decimal)	Description
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) In addition, a key in a key identifier token has been reenciphered. **User action**: See reason code 10000 (return code 0) for more detail about the key reencipherment.
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.
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.
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)

Table 384. Reason codes for return code 4 (4) (continued)

Reason Code Hex (Decimal)	Description
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 doesn't 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 read 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 read services in 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 KDRS 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.
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.

Table 384. Reason codes for return code 4 (4) (continued)

Reason Code Hex (Decimal)	Description
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 was supplied does not match the message text provided. 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 successfully but the supplied digital signature failed verification. **User action**: None **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)
8D10 (36112)	CKDS conversion completed successfully but some tokens could not be rewrapped because the control vector prohibited rewrapping from the enhanced wrapping method.

Reason codes for return code 8 (8)

Table 385 on page 949 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 385. 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 is incorrect for VISA CVV.
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 the PKDS. Either the key could not be found, or a key record with that label and the specific type 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. **REASONCODES**: ICSF 272C (10028)
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. **REASONCODES**: ICSF 7E0 (2016)

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	**Description**
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 lower 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). Please 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). Please see those reason codes for additional information.
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)

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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). Please 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 private key can be used only for digital signature. Key management services are disallowed. **User action**: Supply a key with key management enabled. OR 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. OR This service requires an RSA private key that is translatable. The specified key may not be used in the PKA Key Translate callable service. **User action**: Re-invoke the service with a supported private key. To make a key translatable, XLATE-OK must be turned on.
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)

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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.
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). Please 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 date 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)	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 BE0 (3040)
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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 PAD data is not numeric character date. **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)
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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 generation callable service, or the MDC generation 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)
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). Please 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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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)
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 supplied in a key identifier parameter has replicated key values (the left and right key values are the same). The key cannot be used in the service called. **User action:** Supply a key that doesn't have replicated key values.
25D (605)	The number of output bytes is greater than the number that is permitted.
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. **User action**: Check that you specified the valid value for the parameter. **REASONCODES**: ICSF 834 (2100)

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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)
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). Please see those reason codes for additional information. **REASONCODES**: ICSF 2B00 (11008)
30F (783)	The command is not permitted by the Function Control Vector value. **REASONCODES**: ICSF Return code 12, reason code 2B0C (11020)
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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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 of subsections.
426 (1062)	Trusted block contains 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.
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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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 loads 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.
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.
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 your 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 doesn't match.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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) 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) 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.
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 generation callable service, or the MDC generation 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)

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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 UKPT 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)	The RSA public key is too small to encrypt the DES key.
808 (2056)	The *key_form* parameter is neither IM nor OP. Most constants, these included, can be supplied in lower 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)
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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. Either an encrypted key token is not supported for this service (CSNDPKE) or the required hardware is not present (CSNBSYD and CSNBSYE).
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.
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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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)
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.
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

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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)	The key verification pattern for the key-encrypting key is not valid.
85E (2142)	A key-usage field setting in a supplied key token prevents operation. This may be due to: • a conflict in key-usage setting between two key tokens • a key-usage bit required for the operation is not enabled • an enabled key-usage bit is disallowed for the key for the operation. **User action:** Supply key tokens with allowed key-usage settings.
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).
86A (2154)	At least one key token passed to this callable service does not have the required key type for the specified function. **User action:** Refer to the parameters described in this publication under the appropriate callable service for the correct parameter values.
86C (2156)	Multiple ECC tokens were passed to this callable service. The curve types of the all the token parameters do not match. **User action:** Check that the curve types of the input ECC tokens are the same.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
86F (2159)	A 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.
87E (2174)	The provided data is not hexadecimal digits. **User action**: Provide the data in the correct format.
87F (2175)	A weak PIN was presented. The PIN change has been rejected. **User action**: Provide another PIN.
87E (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 is 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.
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
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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.
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.
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.
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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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 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 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. **REASONCODES**: TSS 06A (106)
BC8 (3016)	A request was made to the Encrypted PIN Translate or the Encrypted PIN verify callable service, and the PIN block value or PADDIGIT 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. **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 data space or a request to delete a PKCS #11 session object failed because the token was not found in the session data space.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
BCD (3021)	The call to add a z/OS PKCS #11 token failed because the token already exists in the TKDS data space 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 tokens 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 data space, or if 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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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.
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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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 symmetric key label refers to 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 in calls to Symmetric Key Encipher (CSNBSYE) or Symmetric Key Decipher (CSNBSYD). Otherwise, use Encipher (CSNBENC) or Decipher (CSNBDEC) instead.
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.
C04 (3076)	A symmetric key token was supplied in a key identifier parameter which is wrapped using the enhanced X9.24 key wrapping method. The token cannot be rewrapped to the original method because the wrapping flag in the control vector prohibits this wrapping.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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)	Use of an ECC token has been attempted. The usage of this type of 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 not a DES or AES key token, but this callable service is only compatible with DES and AES key tokens. **User action:** Either modify the program logic to utilize only key labels for DES and/or AES key tokens, or use an ICSF callable service that supports all of the symmetric key token types.
C0D (3085)	Rule array keyword specifies a function not supported by this hardware. For example, ECC specified in rule array for PKA Key Token Change callable service but request is being executed on a system that does not support ECC keys. **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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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 members active KDS is not allowed as it would alter all sysplex members configured in that sysplex KDS cluster (same active KDS). **User action:** Specify a new KDS that is not currently the active KDS of another sysplex member and retry the function.
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. **User action:** Specify the correct key data set in the rule array.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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.
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 metadata blocks are read only. The metadata blocks cannot be changed. **User action:** Remove the IBM variable metadata block from the metadata list.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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 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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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)
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 the PKDS. Either the key could not be found, or a key record with that label and the specific type 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)

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
272C (10028)	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. **User action**: Determine which key identifier is in error and use the key identifier that is required by the service. **REASONCODES**: TSS 027 (039)
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. 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**: 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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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)
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. **User action**: Correct the appropriate length field parameter. **REASONCODES**: TSS 048 (072)

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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.) You cannot use the key. **User action**: You may need to re-create the token 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)
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)

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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.
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.

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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)
3E80 (16000)	RACF failed your request to use this service. **User action**: Contact your ICSF or RACF administrator if you need this service.
3E84 (16004)	RACF failed your request to use the key label. This may be caused by either CSFKEYS or XCSFKEY class, depending on the setting of the Granular Keylabel Access Controls and the type of token provided. **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.
3E90 (16016)	The input/output field contained a valid internal token with the NOCV bit on or encryption algorithm mark, but the key type was incorrect or did not match the type of the generated or imported key. Processing failed. **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 CKDS key record create callable service requires that the key created not already exist in the CKDS or PKDS. 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)

Table 385. Reason codes for return code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
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.
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 386 on page 983 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 386. 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.
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.

Table 386. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
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.
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.

Table 386. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
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 don't 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)	The sysplex KDS cluster members' new AES master key registers were loaded with different values 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.
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.
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.

Table 386. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
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 on a system with a level of hardware that is not supported by the function. This reason code is also used if the licensed internal code (LIC) level on the originating system is lower then the licensed internal code (LIC) level on 1 or more of the other sysplex KDS cluster members. **User action:** Refer to "Coordinated KDS Administration (CSFCRC and CSFCRC6)" on page 771 for a list of supported hardware levels. Perform the coordinated KDS function from the system running the highest level of licensed internal code (LIC).
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.
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.

Table 386. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
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.
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.
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.

Table 386. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
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 386. 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 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.
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.
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.

Table 386. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
178C (6028)	ESTAE could not be established in common I/O routines. **User action**: Contact your system programmer or the IBM Support Center.
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)

Table 386. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
274C (10060)	The I/O 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.
2B08 (11016)	The master key is not in a valid state. **User action**: Contact your ICSF administrator. **REASONCODES**: TSS 2FC (764)
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 system administrator has used the ICSF User Control Functions panel to disable the RSA functions. **User action**: Wait until administrator functions are complete and the RSA functions are again enabled.
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.

Table 386. Reason codes for return code C (12) (continued)

Reason Code Hex (Decimal)	Description
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.
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 load a key data set (CKDS, PKDS or TKDS) which has records which are in KDSR format. This level of ICSF does not support KDSR format records. **User Action:** Contact your ICSF administrator.

Reason codes for return code 10 (16)

Table 387 lists reason codes returned from callable services that give return code 16.

Table 387. 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)

Appendix B. Key Token Formats

For debugging purposes, this appendix provides the formats for AES, DES internal, external, and null key tokens and for PKA key tokens.

AES Key Token Formats

AES internal key token

Table 388 shows the format for an AES internal key token.

Table 388. Internal Key Token Format

Bytes	Description
0	X'01' (flag indicating this is an internal key token)
1–3	Implementation-dependent bytes (X'000000' for ICSF)
4	Key token version number (X'04')
5	Reserved - must be set to X'00'
6	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.
7	1-byte LRC checksum of clear key value.
8–15	Master key verification pattern (MKVP) (For a clear AES key token this value will be hex zeros.)
16–47	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.
48–55	8-byte control vector. (For a clear AES key token this value will be hex zeros.)
56–57	2-byte integer specifying the length in bits of the clear key value.
58–59	2-byte integer specifying the length in bytes of the encrypted key value. (For a clear AES key token this value will be hex zeros.)
60–63	Token validation value (TVV). See“Token Validation Value” for more information.

Token Validation Value

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.

DES Key Token Formats

DES internal key token

Table 389 shows the format for a DES internal key token.

Table 389. Internal Key Token Format

Bytes	Description
0	X'01' (flag indicating this is an internal key token)
1–3	Implementation-dependent bytes (X'000000' for ICSF)
4	Key token version number (X'00' or X'01')
5	Reserved (X'00')
6	Flag byte **Bit** **Meaning When Set On** **0** Encrypted key and master key verification pattern (MKVP) are present. **1** Control vector (CV) value in this token has been applied to the key. **2** Key is used for no control vector (NOCV) processing. Valid for transport keys only. **3** Reserved **4** Reserved **5** Reserved **6** Reserved **7** Export prohibited.
7	**Bit** **Meaning When Set On** **0-2** Key value encryption method. • 000 - the key is encrypted using the original CCA method (ECB). • 001 - the key is encrypted using the X9.24 enhanced method (CBC). These bits are ignored if the token contains no key or a clear key. **3-7** Reserved.
8–15	Master key verification pattern (MKVP)
16–23	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.
24–31	X'0000000000000000' if a single-length key, or the right half of a double-length operational key, or Part B of a triple-length operational 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.
32–39	The control vector (CV) for a single-length key or the left half of the control vector for a double-length key.
40–47	X'0000000000000000' if a single-length key or the right half of the control vector for a double-length operational key.
48–55	X'0000000000000000' if a single-length key or double-length key, or Part C of a triple-length operational 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.
56-58	Reserved (X'000000')
59 bits 0 and 1	X’00’

Table 389. Internal Key Token Format (continued)

Bytes	Description
59 bits 2 and 3	**B'00'** Indicates single-length key (version 0 only). **B'01'** Indicates double-length key (version 1 only). **B'10'** Indicates triple-length key (version 1 only).
59 bits 4 –7	B'0000'
60–63	Token validation value (TVV).

Note: A fixed-length key token 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. See "Token Validation Value" on page 996 for more information.

DES external key token

Table 390 shows the format for a DES external key token.

Table 390. Format of External Key Tokens

Bytes	Description
0	X'02' (flag indicating an external key token)
1	Reserved (X'00')
2–3	Implementation-dependent bytes (X'0000' for ICSF)
4	Key token version number (X'00' or X'01')
5	Reserved (X'00')
6	Flag byte **Bit** **Meaning When Set On** **0** Encrypted key is present. **1** Control vector (CV) value has been applied to the key. Other bits are reserved and are binary zeros.
7	**Bit** **Meaning When Set On** **0-2** Key value encryption method. • 000 - the key is encrypted using the original CCA method (ECB). • 001 - the key is encrypted using the X9.24 enhanced method (CBC). These bits are ignored if the token contains no key or a clear key. **3-7** Reserved.
8–15	Reserved (X'0000000000000000')
16–23	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.
24–31	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.
32–39	Control vector (CV) for single-length key or left half of CV for double-length key
40–47	X'0000000000000000' if single-length key or right half of CV for double-length key

Table 390. Format of External Key Tokens (continued)

Bytes	Description
48–55	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.
56–58	Reserved (X'000000')
59 bits 0 and 1	B'00'
59 bits 2 and 3	**B'00'** Indicates single-length key (version 0 only). **B'01'** Indicates double-length key (version 1 only). **B'10'** Indicates triple-length key (version 1 only).
59 bits 4–7	B'0000'
60-63	Token validation value (see "Token Validation Value" on page 996 for a description).

External RKX DES Key Token

Table 391 defines an external DES key-token called an *RKX key-token*. An RKX key-token is a special token used exclusively by the Remote Key Export (CSNDRKX and CSNFRKX) and DES key-storage callable services (for example, CKDS Key Record Write). No other callable services use or reference an RKX key-token or key-token record. For additional information about the usage of RKX key tokens, see "Remote Key Loading" on page 38.

Note: Callable services other than the Remote Key Export and the DES key-storage callable services do not support RKX key tokens or RKX key token records.

As can be seen in the table, RKX key tokens are 64 bytes in length, 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, and a token-validation value. In addition, they have a confounder value, a MAC value, and room for a third encrypted key.

Table 391. 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 391. External RKX DES key-token format, version X'10' (continued)

Offset	Length	Meaning
40	8	Rule ID The trusted block rule identifier used to create this key token. A subsequent call to Remote Key Export (CSNDRKX or CSNFRKX) 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 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.
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 using the same MAC key that is used to protect the trusted block itself. This MAC value guarantees 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 used to create the key, thus binding the key structure to that specific trusted block.

Note:

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.

DES null key token

Table 392 shows the format for a fixed length DES null key token.

Table 392. Format of Null Key Tokens

Bytes	Description
0	X'00' (flag indicating this is a null key token).
1–15	Reserved (set to binary zeros).
16–23	Single-length encrypted key, or left half of double-length encrypted key, or Part A of triple-length encrypted key.
24–31	X'0000000000000000' if a single-length encrypted key, the right half of double-length encrypted key, or Part B of triple-length encrypted key.
32–39	X'0000000000000000' if a single-length encrypted key or double-length encrypted key.
40–47	Reserved (set to binary zeros).
48–55	Part C of a triple-length encrypted key.

Table 392. Format of Null Key Tokens (continued)

Bytes	Description
56–63	Reserved (set to binary zeros).

Variable-length Symmetric Key Token Formats

Variable-length symmetric key token

The following table presents the presents the format for a variable-length symmetric key token. The length of the token depends on the key type and algorithm.

Table 393. Variable-length symmetric key token

Offset (Dec)	Length of Field (Bytes)	Description
		Header
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
10	16	Verification pattern of the key used to wrap the payload. Value is left justified.
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

Table 393. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
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')
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

Table 393. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
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 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.
45	*kuf* * 2	Key-usage fields (*kuf* * 2 bytes) • For HMAC algorithm keys, refer to Table 395 on page 1004. • For AES algorithm Key-Encrypting keys (Exporter or Importer), refer to Table 401 on page 1012. • For AES algorithm Cipher keys, refer to Table 402 on page 1014. • For AES algorithm MAC keys, refer to Table 396 on page 1005. • For AES algorithm PINCALC keys, refer to Table 397 on page 1007. • For AES algorithm PINPROT keys, refer to Table 398 on page 1007. • For AES algorithm PINPRW keys, refer to Table 399 on page 1009. • For AES algorithm DKYGENKY keys, refer to Table 400 on page 1010. • For DESUSECV keys, refer to Table 394 on page 1004
45 + *kuf* * 2	1	Key-management field count (*kmf*) - (2 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	*kuf* * 2	Key-management fields (kmf * 2 bytes): • For AES and HMAC algorithm keys, refer to Table 403 on page 1015 • For DESUSECV keys, refer to Table 404 on page 1017
46 + *kuf* * 2 + *kmf* * 2	*kl*	Key name

Table 393. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
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
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 below. **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 394. 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 395. HMAC algorithm key-usage fields

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (*kuf*): 2

Table 395. 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 396. 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

Table 396. AES algorithm MAC 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 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: 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) **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 397. 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.
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 398. AES algorithm PINPROT key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (kuf): 3

Table 398. AES algorithm PINPROT 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 encryption; key can be used for decryption. **B'10xx xxxx'** Key can be used for encryption; key cannot be used for decryption. **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'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) **X'02'** PIN_OPP (DKPINOPP) **X'03'** PIN_ADMIN1 (DKPINAD1) 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 399. AES algorithm PINPRW 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'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 400. AES algorithm DKYGENKY key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (kuf): 2, 4, 5, or 6
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) 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 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: 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. All other values are reserved and undefined.

Table 400. 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.
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.
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.
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.

Table 401. AES algorithm KEK key-usage fields

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (*kuf*): 4
45	2	Key-usage field 1 High-order byte for EXPORTER: **1xxx xxxx** Key can be used for EXPORT. **x1xx xxxx** Key can be used for TRANSLAT. **xx1x xxxx** Key can be used for GENERATE-OPEX. **xxx1 xxxx** Key can be used for GENERATE-IMEX. **xxxx 1xxx** Key can be used for GENERATE-EXEX. **xxxx x1xx** Key can be used for GENERATE-PUB. All unused bits are reserved and must be zero. High-order byte for IMPORTER: **1xxx xxxx** Key can be used for IMPORT. **x1xx xxxx** Key can be used for TRANSLAT. **xx1x xxxx** Key can be used for GENERATE-OPIM. **xxx1 xxxx** Key can be used for GENERATE-IMEX. **xxxx 1xxx** Key can be used for GENERATE-IMIM. **xxxx x1xx** Key can be used for GENERATE-PUB. 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 401. AES algorithm KEK key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
47	2	Key-usage field 2 High-order byte: **1xxx xxxx** Key can wrap a TR-31 key. 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
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 All unused bits are reserved and must be zero. Low-order byte: All bits are reserved and must be zero.

Table 401. AES algorithm KEK key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
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 402. AES algorithm Cipher Key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (*kuf*): 2
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 be used for cipher text translate only. 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 402. 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) All unused values are reserved and must not be used. Low-order byte: All bits are reserved and must be zero.

Table 403. AES and HMAC algorithm key-management fields

Offset (Dec)	Length of Field (Bytes)	Description
48	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. 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 403. 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.
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 403. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
50 + *kuf* * 2 (cont'd)	2 (cont'd)	**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) **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 404. DESUSECV key-management fields

Offset (Dec)	Length of Field (Bytes)	Description
47	1	Key-management field count (*kmf*): 1

Table 404. DESUSECV key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
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.

Variable-length symmetric null key token

The following table shows the format for a variable-length symmetric null key token.

Table 405. Variable-length symmetric null token

Bytes	Description
0	X'00' Token identifier (indicates that this is a null key token).
1	Version, X'00'.
2-3	X'0008' Length of the key token structure.
4-7	Ignored (zero).

PKA Key Token Formats

PKA null key token

Table 406 shows the format for a PKA null key token.

Table 406. Format of PKA Null Key Tokens

Bytes	Description
0	X'00' Token identifier (indicates that this is a null key token).
1	Version, X'00'
2–3	X'0008' Length of the key token structure.
4–7	Ignored (should be zero).

RSA Key Token Formats

RSA public key token

An RSA public key token contains the following sections:
- A required token header, starting with the token identifier X'1E'
- A required RSA public key section, starting with the section identifier X'04'

Table 407 presents the format of an RSA public key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first.

Table 407. RSA Public Key Token

Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1E' indicates an external token.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
RSA Public Key Section (required)		
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx+yyy.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, “xxx”.
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, “yyy”.
012	xxx	Public key exponent (this is generally a 1-, 3-, or 64- 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.

RSA private external key token

An RSA private external key token contains the following sections:

- A required PKA token header starting with the token identifier X'1E'
- A required RSA private key section starting with one of the following section identifiers:
  - X'02' which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 1024 bits.
  - X'08' which indicates an optimized Chinese Remainder Theorem form private key section with modulus bit length of up to 4096.
  - X'09' which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 4096 bits.
  - X'30' which indicates a modulus-exponent form RSA private key section with modulus length of up to 4096 bits with an AES object protection key.
  - X'31' which indicates an Chinese Remainder Theorem form private key section with modulus bit length of up to 4096 bits with an AES object protection key.
- A required RSA public key section, starting with the section identifier X'04'
- An optional private key name section, starting with the section identifier X'10'

Table 408 on page 1020 presents the basic record format of an RSA private external key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first. All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 408. RSA Private External Key Token Basic Record Format

Offset (Dec)	Number of Bytes	Description
		Token Header (required)
000	001	Token identifier. X'1E' indicates an external token. The private key is either in cleartext or enciphered with a transport key-encrypting key.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
		RSA Private Key Section (required) • For 1024-bit Modulus-Exponent form refer to "RSA private key token, 1024-bit modulus-exponent external format" on page 1021. • For 4096-bit Modulus-Exponent form refer to "RSA private key token, 4096-bit modulus-exponent external format" on page 1021. • For 4096-bit Chinese Remainder Theorem form refer to "RSA private key token, 4096-bit chinese remainder Theorem external format" on page 1023. • For 4096-bit Modulus-Exponent form with AES OPK refer to "RSA private key, 4096-bit modulus-exponent format with AES encrypted OPK section (X'30') external form" on page 1024. • For 4096-bit Chinese Remainder Theorem form with AES OPK refer to "RSA private key, 4096-bit chinese remainder Theorem format with AES encrypted OPK section (X'31') external form" on page 1026.
		RSA Public Key Section (required)
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, "xxx".
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, which is zero for a private token. **Note:** In an RSA private key token, this field should be zero. The RSA private key section contains the modulus.
012	xxx	Public key exponent, e (this is generally a 1-, 3-, or 64- to 512-byte quantity). e must be odd and 1<e<n. (Frequently, the value of e is 2^{16}+1 (=65,537).
		Private Key Name (optional)
000	001	X'10', section identifier, private key name.
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
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.

RSA private key token, 1024-bit modulus-exponent external format

Table 409. RSA Private Key Token, 1024-bit Modulus-Exponent external format

Offset (Dec)	Number of Bytes	Description
000	001	X'02', section identifier, RSA private key, modulus-exponent format (RSA-PRIV)
001	001	X'00', version.
002	002	Length of the RSA private key section X'016C' (364 decimal).
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; set to binary zero.
028	001	Key format and security: **X'00'** Unencrypted RSA private key subsection identifier. **X'82'** Encrypted RSA private key subsection identifier.
029	001	Reserved, binary zero.
030	020	SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.
050	004	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. **6** The key is translatable. All other bits reserved, set to binary zero.
054	006	Reserved; set to binary zero.
060	024	Reserved; set to 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.
		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 $1<n<2^{1024}$.

RSA private key token, 4096-bit modulus-exponent external format

This RSA private key token and the external X'09' token is supported on a CCA Crypto Express coprocessor.

Table 410. RSA Private Key Token, 4096-bit Modulus-Exponent external format

Offset (Dec)	Number of Bytes	Description
000	001	X'09', section identifier, RSA private key, modulus-exponent format (RSAMEVAR).
001	001	X'00', version.

Table 410. RSA Private Key Token, 4096-bit Modulus-Exponent external format (continued)

Offset (Dec)	Number of Bytes	Description
002	002	Length of the RSA private key section 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; set to binary zero.
028	001	Key format and security: **X'00'** Unencrypted RSA private key subsection identifier. **X'82'** Encrypted RSA private key subsection identifier.
029	001	Reserved, set to binary zero.
030	020	SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.
050	001	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. **6** The key is translatable All other bits reserved, set to binary zero.
051	001	Reserved; set to binary zero.
052	048	Reserved; set to binary zero.
100	016	Reserved; set to 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; set to binary zero.
		Start of the optionally-encrypted secure 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 above to the end of the padding field is a multiple of eight bytes.
		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 $1<n<2^{4096}$.

RSA private key token, 4096-bit chinese remainder Theorem external format

This RSA private key token with up to 2048-bit modulus is supported on all coprocessors. The modulus size is increased to 4096-bit on the z9 EC, z9 BC, z10 EC, z10 BC, or later machines with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 411. RSA Private Key Token, 4096-bit Chinese Remainder Theorem external format

Offset (Dec)	Number of Bytes	Description
000	001	X'08', section identifier, RSA private key, CRT format (RSA-CRT)
001	001	X'00', version.
002	002	Length of the RSA private-key section, 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; set to binary zero.
028	001	Key format and security: **X'40'** Unencrypted RSA private-key subsection identifier, Chinese Remainder form. **X'42'** Encrypted RSA private-key subsection identifier, Chinese Remainder form.
029	001	Reserved; set to binary zero.
030	020	SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, then 20 bytes of X'00'.
050	004	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. **6** The key is translatable. All other bits reserved, set to binary 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 d_p, in bytes: rrr.
060	002	Length of d_q, in bytes: sss.
062	002	Length of U, in bytes: uuu.
064	002	Length of modulus, n, in bytes: nnn.
066	004	Reserved; set to binary zero.
070	002	Length of padding field, in bytes: xxx.
072	004	Reserved, set to binary zero.
076	016	Reserved, set to binary zero.
092	032	Reserved; set to binary zero.
		Start of the optionally-encrypted secure subsection.
124	008	Random number, confounder.
132	ppp	Prime number, p.

Table 411. RSA Private Key Token, 4096-bit Chinese Remainder Theorem external format (continued)

Offset (Dec)	Number of Bytes	Description
132 + ppp	qqq	Prime number, q
132 + ppp + qqq	rrr	d_p = d mod(p - 1)
132 + ppp + qqq + rrr	sss	d_q = d mod(q - 1)
132 + ppp + qqq + rrr + sss	uuu	U = q^{-1}mod(p).
132 + ppp + qqq + rrr + sss + uuu	xxx	X'00' padding of length xxx bytes such that the length from the start of the random number above to the end of the padding field is a multiple of eight bytes.
		End of the optionally-encrypted secure subsection; 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.
132 + ppp + qqq + rrr + sss + uuu + xxx	nnn	Modulus, n. n = pq where p and q are prime and $1<n<2^{4096}$.

RSA private key, 4096-bit modulus-exponent format with AES encrypted OPK section (X'30') external form

This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

Table 412. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'30'** RSA private key, ME format with AES encrypted OPK.
001	001	Section version number (X'00').
002	002	Section length: 122 + nnn + ppp
004	002	Length of “Associated Data” section
006	002	Length of payload data: ppp
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: **X'02'** Version 2
011	001	Key format and security flag: **X'00'** Unencrypted ME RSA private-key subsection identifier **X'82'** Encrypted ME RSA private-key subsection identifier
012	001	Key source flag: Reserved, binary zero.
013	001	Reserved, binary zeroes.

Table 412. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form (continued)

Offset (bytes)	Length (bytes)	Description
014	001	Hash type: **X'00'** Clear key **X'02'** SHA-256
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.
050	001	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) Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK) **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE)
051	001	Reserved, binary zero.
052	002	Length of modulus: nnn bytes
054	002	Length of private exponent: ddd bytes
		End of Associated Data
056	048	16 byte confounder + 32-byte Object Protection Key. OPK used as an AES key. encrypted with an AES KEK.
104	016	Key verification pattern • For an encrypted private key, KEK verification pattern (KVP) • For a clear private key, binary zeros • For a skeleton, binary zeros
120	002	Reserved, binary zeros.
122	nnn	Modulus
122+nnn	ppp	Payload starts here and includes: When this section is unencrypted: • Clear private exponent d. • Length ppp bytes : ddd + 0 When this section is encrypted: • Private exponent d within the AESKW-wrapped payload. • Length ppp bytes : ddd + AESKW format overhead

RSA private key, 4096-bit chinese remainder Theorem format with AES encrypted OPK section (X'31') external form

This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

Table 413. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form

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
004	002	Length of “Associated Data” section
006	002	Length of payload data: xxx
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: **X'03'** Version 3
011	001	Key format and security flag: **X'40'** Unencrypted RSA private-key subsection identifier **X'42'** Encrypted RSA private-key subsection identifier
012	001	Key source flag: Reserved, binary zero.
013	001	Reserved, binary zeroes.
014	001	Hash type: **X'00'** Clear key **X'01'** SHA-256
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.
050	001	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) Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK) **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE)

Table 413. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form (continued)

Offset (bytes)	Length (bytes)	Description
051	001	Reserved, binary zero.
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	002	Reserved, binary zero.
066	002	Reserved, binary zero.
		End of Associated Data
068	048	16 byte confounder + 32-byte Object Protection Key. OPK used as an AES key. External tokens: encrypted with an AES KEK. Internal tokens: encrypted with the ECC master key.
116	016	Key verification pattern • For an encrypted private key, KEK verification pattern (KVP) • For a clear private key, binary zeros • For a skeleton, binary zeros
132	002	Reserved, binary zeros
134	nnn	Modulus, n, n=pq, where p and q are prime.
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 + 0 When this section is encrypted: • prime number p • prime number q • dp • dq • U • within the AESKW-wrapped payload. Length xxx bytes : ppp + qqq + rrr + sss +uuu + AESKW format overhead

RSA private internal key token

An RSA private internal key token contains the following sections:

- A required PKA token header, starting with the token identifier X'1F'
- Basic record format of an RSA private internal key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first. All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 414. RSA Private Internal Key Token Basic Record Format

Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1F' indicates an internal token. The private key is enciphered with a PKA master key.
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
RSA Private Key Section and Secured Subsection (required) • For 1024-bit X'02' Modulus-Exponent form, refer to "RSA private key token, 1024-bit X'02' modulus-exponent internal form" on page 1029. • For 1024-bit X'06' Modulus-Exponent form, refer to "RSA private key token, 1024-bit X'06' modulus-exponent internal form" on page 1030. • For 4096-bit X'08' Chinese Remainder Theorem form, refer to "RSA private key token, 4096-bit chinese remainder Theorem internal form" on page 1035. • For 4096-bit Modulus-Exponent form with AES OPK, refer to "RSA private key, 4096-bit modulus-exponent format with AES encrypted OPK section internal form" on page 1031. • For 4096-bit Chinese Remainder Theorem form with AES OPK, refer to Table 418 on page 1033.		
RSA Public Key Section (required)		
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, "xxx".
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, which is zero for a private token.
012	xxx	Public key exponent (this is generally a 1, 3, or 64 to 512 byte quantity), e. e must be odd and 1<e<n. (Frequently, the value of e is $2^{16}+1$ (=65,537).
Private Key Name (optional)		
000	001	X'10', section identifier, private key name.
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
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.
Internal Information Section (required)		
000	004	Eye catcher 'PKTN'.

Table 414. RSA Private Internal Key Token Basic Record Format (continued)

Offset (Dec)	Number of Bytes	Description
004	004	PKA token type. **Bit** **Meaning When Set On** **0** RSA key. **1** DSS key. **2** Private key. **3** Public key. **4** Private key name section exists. **5** Private key unenciphered. **6** Blinding information present. **7** Retained private key.
008	004	Address of token header.
012	002	Total length of total structure including this information section.
014	002	Count of number of sections.
016	016	PKA master key hash pattern.
032	001	Domain of retained key.
033	008	Serial number of processor holding retained key.
041	007	Reserved.

RSA private key token, 1024-bit X'02' modulus-exponent internal form

Table 415. RSA Private Internal Key Token, 1024-bit X'02' ME Form

Offset (Dec)	Number of Bytes	Description
000	001	X'02', section identifier, RSA private key.
001	001	X'00', version.
002	002	Length of the RSA private key section X'016C' (364 decimal).
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; set to binary zero.
028	001	Key format and security: **X'02'** RSA private key.
029	001	Format of external key from which this token was derived: **X'21'** External private key was specified in the clear. **X'22'** External private key was encrypted.
030	020	SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.

Table 415. RSA Private Internal Key Token, 1024-bit X'02' ME Form (continued)

Offset (Dec)	Number of Bytes	Description
050	001	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. All other bits reserved, set to binary zero.
051	009	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the RSA-MK.
108	128	Secret key exponent d, encrypted under the OPK. $d=e^{-1}$ mod((p-1)(q-1))
236	128	Modulus, n. n=pq where p and q are prime and $1<n<2^{1024}$.

RSA private key token, 1024-bit X'06' modulus-exponent internal form

Table 416. RSA Private Internal Key Token, 1024-bit X'06' ME Form

Offset (Dec)	Number of Bytes	Description
000	001	X'06', section identifier, RSA private key modulus-exponent format (RSA-PRIV).
001	001	X'00', version.
002	002	Length of the RSA private key section X'0198' (408 decimal) + rrr + iii + xxx.
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; set to binary zero.
028	001	Key format and security: **X'02'** RSA private key.
029	001	Format of external key from which this token was derived: **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.
030	020	SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, this field is set to binary zeros.
050	004	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. All other bits reserved, set to binary zeros.
054	006	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the RSA-MK using the ede3 algorithm.

Table 416. RSA Private Internal Key Token, 1024-bit X'06' ME Form (continued)

Offset (Dec)	Number of Bytes	Description
108	128	Private key exponent d, encrypted under the OPK using the ede5 algorithm. $d=e^{-1}mod((p-1)(q-1))$, 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 blinding information subsection cleartext, offset 400 to the end of the section.
400	002	Length of the random number r, in bytes: rrr.
402	002	Length of the random number r^{-1}, in bytes: iii.
404	002	Length of the padding field, in bytes: xxx.
406	002	Reserved; set to binary zeros.
408	Start of the encrypted blinding subsection	
408	rrr	Random number r (used in blinding).
408 + rrr	iii	Random number r^{-1} (used in blinding).
408 + rrr + iii	xxx	X'00' padding of length xxx bytes such that the length from the start of the encrypted blinding subsection to the end of the padding field is a multiple of eight bytes.
	End of the encrypted blinding subsection; all of the fields starting with the random number r and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) algorithm.	

RSA private key, 4096-bit modulus-exponent format with AES encrypted OPK section internal form

This RSA private key token is supported on the Crypto Express3 and newer Coprocessor.

Table 417. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'30'** RSA private key, ME format with AES encrypted OPK.
001	001	Section version number (X'00').
002	002	Section length: 122 + nnn + ppp
004	002	Length of “Associated Data” section
006	002	Length of payload data: ppp
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: **X'02'** Version 2
011	001	Key format and security flag: **X'02'** Encrypted ME RSA private-key subsection identifier

Table 417. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form (continued)

Offset (bytes)	Length (bytes)	Description
012	001	Key source flag: Internal tokens: **X'21'** Imported from cleartext **X'22'** Imported from ciphertext **X'23'** Generated using regeneration data **X'24'** Randomly generated
013	001	Reserved, binary zeroes.
014	001	Hash type: **X'00'** Clear key **X'02'** SHA-256
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.
050	001	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) Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK) **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE)
051	001	Reserved, binary zero.
052	002	Length of modulus: nnn bytes
054	002	Length of private exponent: ddd bytes
		End of Associated Data
056	048	16 byte confounder + 32-byte Object Protection Key. OPK used as an AES key. encrypted with the ECC master key.
104	016	Key verification pattern • For an encrypted private key, ECC master-key verification pattern (MKVP) • For a skeleton, binary zeros
120	002	Reserved, binary zeros.
122	nnn	Modulus

Table 417. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form (continued)

Offset (bytes)	Length (bytes)	Description
122+nnn	ppp	Payload starts here and includes: When this section is unencrypted: • Clear private exponent d. • Length ppp bytes : ddd + 0 When this section is encrypted: • Private exponent d within the AESKW-wrapped payload. • Length ppp bytes : ddd + AESKW format overhead

RSA private key, 4096-bit chinese remainder Theorem format with AES encrypted OPK section internal form

This RSA private key token is supported on the Crypto Express3 and newer Coprocessor.

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

Table 418. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form

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
004	002	Length of “Associated Data” section
006	002	Length of payload data: xxx
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: **X'03'** Version 3
011	001	Key format and security flag: **X'08'** Unencrypted RSA private-key subsection identifier
012	001	Key source flag: **X'21'** Imported from cleartext **X'22'** Imported from ciphertext **X'23'** Generated using regeneration data **X'24'** Randomly generated
013	001	Reserved, binary zeroes.
014	001	Hash type: **X'00'** Clear key **X'01'** SHA-256

Table 418. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.
050	001	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) Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK) **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE)
051	001	Reserved, binary zero.
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	002	Reserved, binary zero.
066	002	Reserved, binary zero.
		End of Associated Data
068	048	16 byte confounder + 32-byte Object Protection Key. OPK used as an AES key. encrypted with the ECC-MK.
116	016	Key verification pattern • For an encrypted private key, ECC master-key verification pattern (MKVP) • For a skeleton, binary zeros
132	002	Reserved, binary zeros
134	nnn	Modulus, n, n=pq, where p and q are prime.

Table 418. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
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 + 0 When this section is encrypted: • prime number p • prime number q • dp • dq • U • within the AESKW-wrapped payload. Length xxx bytes : ppp + qqq + rrr + sss +uuu + AESKW format overhead

RSA private key token, 4096-bit chinese remainder Theorem internal form

This RSA private key token (up to 2048-bit modulus) is supported on all cryptographic coprocessors. The 4096-bit modulus private key token is supported on the z9 EC, z9 BC, z10 EC, z10 BC, or IBM zEnterprise 196 with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 419. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format

Offset (Dec)	Number of Bytes	Description
000	001	X'08', section identifier, RSA private key, CRT format (RSA-CRT)
001	001	X'00', version.
002	002	Length of the RSA private-key section, 132 + ppp + qqq + rrr + sss + uuu + ttt + iii + 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; set to binary zero.
028	001	Key format and security: **X'08'** Encrypted RSA private-key subsection identifier, Chinese Remainder form.
029	001	Key derivation method: **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.

Table 419. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)

Offset (Dec)	Number of Bytes	Description
030	020	SHA-1 hash of the optional key-name section and any following sections. If there are no optional sections, then 20 bytes of X'00'.
050	004	Key use flag bits: **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. All other bits reserved, set to binary 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 d_p, in bytes: rrr.
060	002	Length of d_q, in bytes: sss.
062	002	Length of U, in bytes: uuu.
064	002	Length of modulus, n, in bytes: nnn.
066	002	Length of the random number r, in bytes: ttt.
068	002	Length of the random number r^{-1}, in bytes: iii.
070	002	Length of padding field, in bytes: xxx.
072	004	Reserved, set to binary zero.
076	016	RSA master key verification pattern.
092	032	Object Protection Key (OPK) encrypted under the Asymmetric-Keys Master Key using the TDES (CBC outer chaining) algorithm.
124	Start of the encrypted secure subsection, encrypted under the OPK using TDES (CBC outer chaining).	
124	008	Random number, confounder.
132	ppp	Prime number, p.
132 + ppp	qqq	Prime number, q
132 + ppp + qqq	rrr	$d_p = d \bmod(p - 1)$
132 + ppp + qqq + rrr	sss	$d_q = d \bmod(q - 1)$
132 + ppp + qqq + rrr + sss	uuu	$U = q^{-1} \bmod(p)$.
132 + ppp + qqq + rrr + sss + uuu	ttt	Random number r (used in blinding).
132 + ppp + qqq + rrr + sss + uuu + ttt	iii	Random number r^{-1} (used in blinding).
132 + ppp + qqq + rrr + sss + uuu + ttt + iii	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 encrypted secure subsection; 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 + ttt + iii + xxx	nnn	Modulus, n. $n = pq$ where p and q are prime and $1<n<2^{4096}$.

RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form:

Table 420. RSA Private Internal Key Token, 1024-bit ME Form

Offset (Dec)	Number of Bytes	Description
000	001	X'02', section identifier, RSA private key.
001	001	X'00', version.
002	002	Length of the RSA private key section X'016C' (364 decimal).
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; set to binary zero.
028	001	Key format and security: **X'02'** RSA private key.
029	001	Format of external key from which this token was derived: **X'21'** External private key was specified in the clear. **X'22'** External private key was encrypted.
030	020	SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.
050	001	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. All other bits reserved, set to binary zero.
051	009	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the RSAmaster key.
108	128	Secret key exponent d, encrypted under the OPK. $d=e^{-1} \bmod((p-1)(q-1))$
236	128	Modulus, n. $n=pq$ where p and q are prime and $1<n<2^{1024}$.

RSA Private Key Token, 1024-bit Modulus-Exponent internal form with encrypted blinding:

Table 421. RSA Private Internal Key Token, 1024-bit ME internal form with encrypted blinding

Offset (Dec)	Number of Bytes	Description
000	001	X'06', section identifier, RSA private key modulus-exponent format (RSA-PRIV).
001	001	X'00', version.
002	002	Length of the RSA private key section X'0198' (408 decimal) + rrr + iii + xxx.
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; set to binary zero.
028	001	Key format and security: **X'02'** RSA private key.

Table 421. RSA Private Internal Key Token, 1024-bit ME internal form with encrypted blinding (continued)

Offset (Dec)	Number of Bytes	Description
029	001	Format of external key from which this token was derived: **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.
030	020	SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, this field is set to binary zeros.
050	004	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. All other bits reserved, set to binary zeros.
054	006	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the RSA Master Key using the ede3 algorithm.
108	128	Private key exponent d, encrypted under the OPK using the ede5 algorithm. $d=e^{-1}mod((p-1)(q-1))$, 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 hash pattern.
380	020	SHA-1 hash value of the blinding information subsection cleartext, offset 400 to the end of the section.
400	002	Length of the random number r, in bytes: rrr.
402	002	Length of the random number r^{-1}, in bytes: iii.
404	002	Length of the padding field, in bytes: xxx.
406	002	Reserved; set to binary zeros.
408	Start of the encrypted blinding subsection	
408	rrr	Random number r (used in blinding).
408 + rrr	iii	Random number r^{-1} (used in blinding).
408 + rrr + iii	xxx	X'00' padding of length xxx bytes such that the length from the start of the encrypted blinding subsection to the end of the padding field is a multiple of eight bytes.
	End of the encrypted blinding subsection; all of the fields starting with the random number r and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) algorithm.	

RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section internal form: This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

Table 422. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'30'** RSA private key, ME format with AES encrypted OPK.
001	001	Section version number (X'00').
002	002	Section length: 122 + nnn + ppp
004	002	Length of "Associated Data" section
006	002	Length of payload data: ppp
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: **X'02'** Version 2
011	001	Key format and security flag: **X'02'** Encrypted ME RSA private-key subsection identifier
012	001	Key source flag: Internal tokens: **X'21'** Imported from cleartext **X'22'** Imported from ciphertext **X'23'** Generated using regeneration data **X'24'** Randomly generated
013	001	Reserved, binary zeroes.
014	001	Hash type: **X'00'** Clear key **X'02'** SHA-256
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.
050	001	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) Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK) **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE)
051	001	Reserved, binary zero.

Table 422. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form (continued)

Offset (bytes)	Length (bytes)	Description
052	002	Length of modulus: nnn bytes
054	002	Length of private exponent: ddd bytes
		End of Associated Data
056	048	16 byte confounder + 32-byte Object Protection Key. OPK used as an AES key. encrypted with the ECC master key.
104	016	Key verification pattern • For an encrypted private key, ECC master-key verification pattern (MKVP) • For a skeleton, binary zeros
120	002	Reserved, binary zeros.
122	nnn	Modulus
122+nnn	ppp	Payload starts here and includes: When this section is unencrypted: • Clear private exponent d. • Length ppp bytes : ddd + 0 When this section is encrypted: • Private exponent d within the AESKW-wrapped payload. • Length ppp bytes : ddd + AESKW format overhead

RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section internal form: This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

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

Table 423. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form

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
004	002	Length of “Associated Data” section
006	002	Length of payload data: xxx
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: **X'03'** Version 3
011	001	Key format and security flag: **X'08'** Unencrypted RSA private-key subsection identifier

Table 423. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
012	001	Key source flag: **X'21'** Imported from cleartext **X'22'** Imported from ciphertext **X'23'** Generated using regeneration data **X'24'** Randomly generated
013	001	Reserved, binary zeroes.
014	001	Hash type: **X'00'** Clear key **X'01'** SHA-256
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.
050	001	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) Translation control: **B'xxxx xx1x'** Private key translation is allowed (XLATE-OK) **B'xxxx xx0x'** Private key translation is not allowed (NO-XLATE)
051	001	Reserved, binary zero.
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	002	Reserved, binary zero.
066	002	Reserved, binary zero.
		End of Associated Data
068	048	16 byte confounder + 32-byte Object Protection Key. OPK used as an AES key. encrypted with the ECC master key.

Table 423. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
116	016	Key verification pattern • For an encrypted private key, ECC master-key verification pattern (MKVP) • For a skeleton, binary zeros
132	002	Reserved, binary zeros
134	nnn	Modulus, n, n=pq, where p and q are prime.
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 + 0 When this section is encrypted: • prime number p • prime number q • dp • dq • U • within the AESKW-wrapped payload. Length xxx bytes : ppp + qqq + rrr + sss +uuu + AESKW format overhead

RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form: This RSA private key token with up to 2048-bit modulus is supported on all coprocessors. The modulus size is increased to 4096-bit on the z9 EC, z9 BC, z10 EC, z10 BC, or later machines with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 424. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format

Offset (Dec)	Number of Bytes	Description
000	001	X'08', section identifier, RSA private key, CRT format (RSA-CRT)
001	001	X'00', version.
002	002	Length of the RSA private-key section, 132 + ppp + qqq + rrr + sss + uuu + ttt + iii + 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; set to binary zero.
028	001	Key format and security: **X'08'** Encrypted RSA private-key subsection identifier, Chinese Remainder form.

Table 424. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)

Offset (Dec)	Number of Bytes	Description
029	001	Key derivation method: **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.
030	020	SHA-1 hash of the optional key-name section and any following sections. If there are no optional sections, then 20 bytes of X'00'.
050	004	Key use flag bits: **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. All other bits reserved, set to binary 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 d_p, in bytes: rrr.
060	002	Length of d_q, in bytes: sss.
062	002	Length of U, in bytes: uuu.
064	002	Length of modulus, n, in bytes: nnn.
066	002	Length of the random number r, in bytes: ttt.
068	002	Length of the random number r^{-1}, in bytes: iii.
070	002	Length of padding field, in bytes: xxx.
072	004	Reserved, set to binary zero.
076	016	RSA Master Key hash pattern.
092	032	Object Protection Key (OPK) encrypted under the RSA Master Key using the TDES (CBC outer chaining) algorithm.
124	Start of the encrypted secure subsection, encrypted under the OPK using TDES (CBC outer chaining).	
124	008	Random number, confounder.
132	ppp	Prime number, p.
132 + ppp	qqq	Prime number, q
132 + ppp + qqq	rrr	$d_p = d \bmod(p - 1)$
132 + ppp + qqq + rrr	sss	$d_q = d \bmod(q - 1)$
132 + ppp + qqq + rrr + sss	uuu	$U = q^{-1} \bmod(p)$.
132 + ppp + qqq + rrr + sss + uuu	ttt	Random number r (used in blinding).
132 + ppp + qqq + rrr + sss + uuu + ttt	iii	Random number r^{-1} (used in blinding).

Table 424. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)

Offset (Dec)	Number of Bytes	Description
132 + ppp + qqq + rrr + sss + uuu + ttt + iii	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 encrypted secure subsection; 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 + ttt + iii + xxx	nnn	Modulus, n. n = pq where p and q are prime and $1<n<2^{4096}$.

ECC key token format

The following table presents the format of the ECC Key Token.

Table 425. ECC Key Token Format

Offset (Dec)	Number of Bytes	Description
Token Header		
000	001	Token identifier. **X'00'** Null token **X'1E'** External token **X'1F'** Internal token; the private key is protected by the master key
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
ECC Token Private section		
000	001	X'20', section identifier, ECC private key
001	001	X'00', version.
002	002	Section length.
004	001	Wrapping Method: This value indicates the wrapping method used to protect the data in the encrypted section. It is not the method used to protect the Object Protection Key (OPK). **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 **X'04'** Reserved. **X'08 '** Reserved
006	002	Reserved Binary Zero

Table 425. ECC Key Token Format (continued)

Offset (Dec)	Number of Bytes	Description
008	001	Key Usage: **X'C0'** Key Agreement **X'80'** Both signature generation and key agreement **X'00'** Signature generation only **X'02'** Translate allowed The two high-order bits indicate permitted key usage in the decryption of symmetric keys and in the generation of digital signatures. The bit in the second nibble indicates if the key is translatable. A key is translatable if it can be re-encrypted from one key encrypting key to another.
009	001	Curve type: **X'00'** Prime curve **X'01'** Brainpool curve
010	001	Key Format and Security Flag. External Token: **X'40'** Unencrypted ECC private key identifier **X'42'** Encrypted ECC private key identifier Internal Token: **X'08'** Encrypted ECC private key identifier
011	001	Reserved Binary Zero
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)
014	002	IBM Associated Data length. The length of this field must be greater than or equal to 16
016	008	External Token: • Unencrypted – Reserved Binary 0x'00' • Encrypted – KVP of the AESKEK Internal Token: MKVP of the ECC-MK

Table 425. ECC Key Token Format (continued)

Offset (Dec)	Number of Bytes	Description
024	048	External Token: reserved binary zeros. Internal Token: Object Protection Key (OPK), ICV (Integrity Check value), 8 byte confounder and a 256-bit AES key used with the AESKW algorithm to encrypt the ECC private key. The OPK is encrypted by the AES master key using AESKW as well. Example format for OPK data passed to AESKW: • 8 bytes = A6A6A6A6A6A60000 • 40 bytes = Confounder(8)/Key(32)
072	002	Associated data length, aa
074	002	Length of formatted section in bytes, bb
076	aa	Associated data See "Associated data format for ECC token" on page 1047.
076 + aa	Start of formatted section	If this section is in the clear it contains private key d. If it is encrypted it contains the AESKW wrapped payload.
76 + aa	bb	Formatted section which includes Private key d See "AESKW wrapped payload format for ECC private key token" on page 1048.
76 + aa + bb	End of formatted section	
ECC Token Public Section		
000	001	X'21', section identifier
001	001	X'00', version.
002	002	Section length
004	004	Reserved field, binary zero
008	001	Curve type **X'00'** Prime curve **X'01'** Brainpool curve
009	001	Reserved field, binary zero

Table 425. ECC Key Token Format (continued)

Offset (Dec)	Number of 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
012	002	This field is the length of the public key q value in bytes, the maximum value could be up to 133 bytes, cc. The value includes the key material length and one byte to indicate if the key material is compressed or uncompressed.
014	cc	Public Key , q field

Associated data format for ECC token

The table below defines the associated data as it is stored in the ECC token in the clear. Associated data is data whose integrity but not confidentiality is protected by a key wrap mechanism.

Table 426. Associated Data Format for ECC Private Key Token

Offset (Dec)	Number of Bytes	Description
000	001	Associated Data Version. 0 for ECC
001	001	Length of Key Label, kl
002	002	IBM Associated Data length, 16 + kl + xxx
004	002	IBM Extended Associated Data length, xxx
006	001	User Definable Associated Data length, yyy. User definable lengths are from 0 bytes to 100 bytes.
007	001	Curve Type
008	002	Length of p in bits
010	001	Usage flag
011	001	Format and Security flag
012	004	reserved
016	kl	Key Label (optional)
016 + kl	xxx	IBM Extended Associated Data
016 + kl + xxx	yyy	User-definable Associated Data

AESKW wrapped payload format for ECC private key token

This table defines the contents of the AESKW payload: data will be copied into this format, then encrypted with the OPK according to the AESKW specification, and the result will be stored in the encrypted data section.

Table 427. AESKW Wrapped Payload Format for ECC Private Key Token

Offset (Dec)	Number of Bytes	Description
000	006	ICV (‘A6’....)
006	001	Length of padding in bits
007	001	Length of the hash of the associated data in bytes, ii
008	004	Hash options
012	ii	Hash of Associated Data
12+ii	mm	Key data
12+ii+mm	0-7	Padding to a multiple of 8 bytes

Trusted block key token

A trusted block key-token (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 verb. This verb 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 verb 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 verb.

  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 428. Trusted block sections

Section	Reference	Usage
Header	Table 429 on page 1050	Trusted block token header

Table 428. Trusted block sections (continued)

Section	Reference	Usage
X'11'	Table 430 on page 1051	Trusted block public key
X'12'	Table 431 on page 1052	Trusted block rule
X'13'	Table 438 on page 1060	Trusted block name (key label)
X'14'	Table 439 on page 1060	Trusted block information
X'15'	Table 443 on page 1063	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 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 429. 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."

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 1050.

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 430. 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 $1 \leq e < n$. (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 $2^{512} \leq n < 2^{4096}$. The field length is 64 - 512 bytes.

Table 430. Trusted block trusted RSA public-key section (X'11') (continued)

Offset (bytes)	Length (bytes)	Description
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 1050.

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 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 431. 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').

Table 431. Trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
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.
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 is the only key format permitted when the flags value (offset 012) is set to generate a new key. **X'01'** Return a CCA DES key-token encrypted under a transport key. **Note:** This is the only key format permitted when the flags value (offset 012) is set to export 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.
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 1050.

Section X'12' has five rule subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 432. 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 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.
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 1050.

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 433. 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 prior to use.

Table 433. Transport key variant subsection (X'0001' of trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
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 1050.

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 434. 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.
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 desired. 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 435. 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.
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 prior to 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.
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 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 a CV value is present. 3. If the source key is 8 bytes in length, do not replicate the key to 16 bytes.

Table 435. Common export key parameters subsection (X'0003') of trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
012+*xxx*	*yyy*	CV. Place this CV into the output exported key-token, provided that 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 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 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 1050.

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 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 436. 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 436. 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 1050.

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 437. 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 437. 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 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 1050.

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 438. 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 1050.

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 439. 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 1050.

Section X'14' has two information subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 440. 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).
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 1050.

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 441. 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 1050.

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 442. Activation and 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 1050.

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 443. 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*)
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. Control Vectors and Changing Control Vectors with the CVT Callable Service

This section 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.

Control Vector Table

Note: The Control Vectors used in ICSF are exactly the same as documented in CCA and the TSS documents.

The master key enciphers all 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 444 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 444. 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	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
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

Table 444. 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)
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	
CVARXCVL	00 3F 44 00 03 00 00 00	
CVARXCVR	00 3F 47 00 03 00 00 00	
DATA (external)	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
DATA (internal)	00 00 7D 00 03 41 00 00	00 00 7D 00 03 21 00 00
DATA	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 generation key (external)	00 00 4D 00 03 41 00 00	00 00 4D 00 03 21 00 00
DATAM key (internal)	00 05 4D 00 03 00 00 00	00 05 4D 00 03 00 00 00
DATAMV MAC verification key (external)	00 00 44 00 03 41 00 00	00 00 44 00 03 21 00 00
DATAMV MAC verification key (internal)	00 05 44 00 03 00 00 00	00 05 44 00 03 00 00 00
DECIPHER	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
DKYGENKY	00 71 44 00 03 41 00 00	00 71 44 00 03 21 00 00
ENCIPHER	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
EXPORTER	00 41 7D 00 03 41 00 00	00 41 7D 00 03 21 00 00
IKEYXLAT	00 42 42 00 03 41 00 00	00 42 42 00 03 21 00 00
IMP-PKA	00 42 05 00 03 41 00 00	00 42 05 00 03 21 00 00
IMPORTER	00 42 7D 00 03 41 00 00	00 42 7D 00 03 21 00 00
IPINENC	00 21 5F 00 03 41 00 00	00 21 5F 00 03 21 00 00
MAC	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
MACVER	00 05 44 00 03 00 00 00	
MACVER (double-length)	00 05 44 00 03 41 00 00	00 05 44 00 03 21 00 00
OKEYXLAT	00 41 42 00 03 41 00 00	00 41 42 00 03 21 00 00
OPINENC	00 24 77 00 03 41 00 00	00 24 77 00 03 21 00 00
PINGEN	00 22 7E 00 03 41 00 00	00 22 7E 00 03 21 00 00
PINVER	00 22 42 00 03 41 00 00	00 22 42 00 03 21 00 00

Note: The external control vectors for DATAC, DATAM MAC generation and DATAMV MAC verification keys are also referred to as data compatibility control vectors.

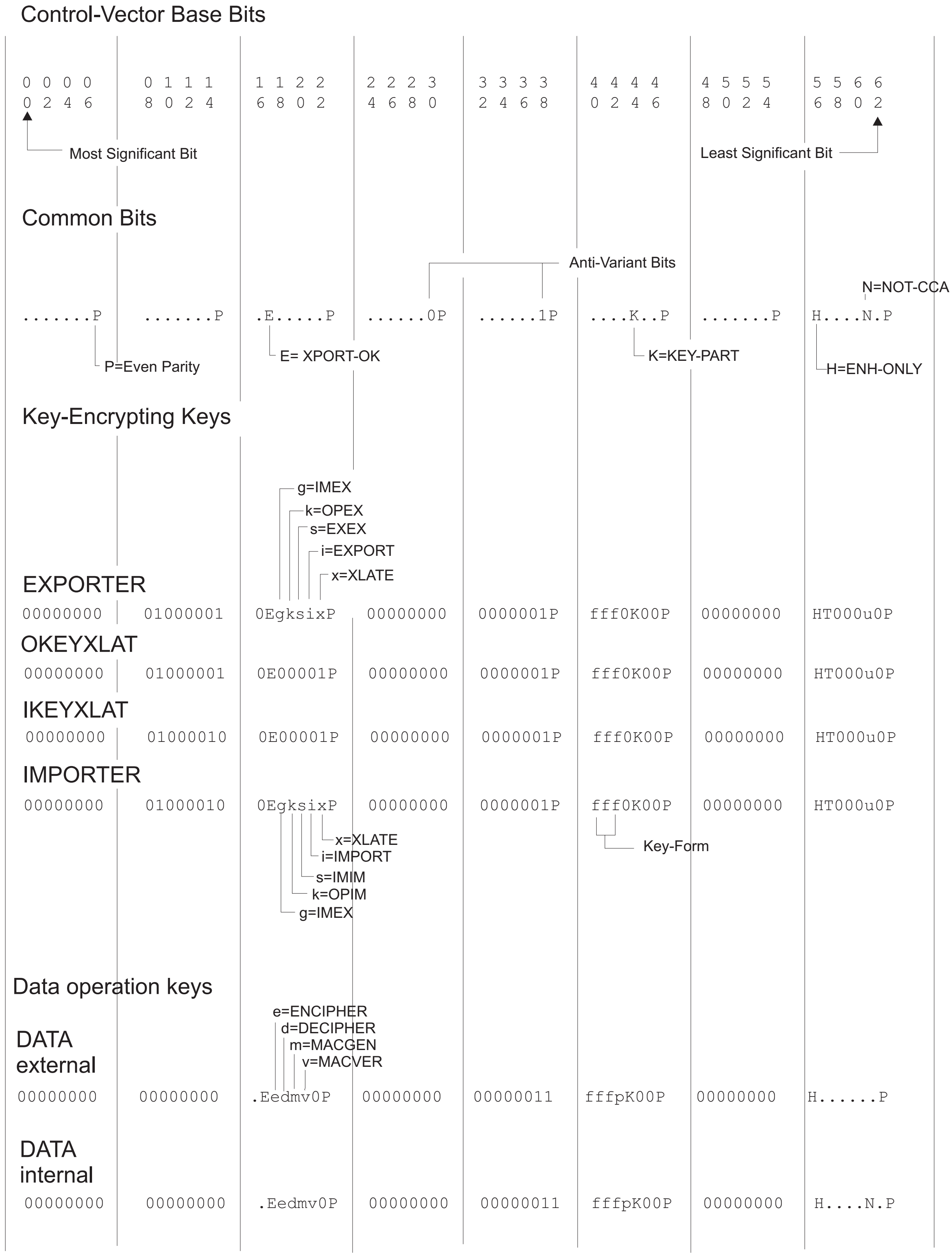

Figure 17. Control Vector Base Bit Map (Common Bits and Key-Encrypting Keys)

Control-Vector Base Bits

	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

Data Operation Keys

e=ENCIPHER
d=DECIPHER
m=MACGEN
v=MACVER

Key								
DATA	00000000	00000000	0Eedmv0P	00000000	00000011	fff0K00P	00000000	HT000u0P
DATAC	00000000	00000000	0E11000P	00000000	00000011	fff0K00P	00000000	HT000u0P
DATAM	00000000	00000000	0E00110P	00000000	00000011	fff0K00P	00000000	HT000u0P
DATAMV	00000000	00000000	0E00010P	00000000	00000011	fff0K00P	00000000	HT000u0P
CIPHER	00000000	00000011	0E11000P	00000000	00000011	fff0K00P	00000000	HT000u0P
CIPHERXI	0000uu00	00001100	0E010001	00000000	00000011	fff0K00P	00000000	HT000u0P
CIPHERXO	0000uu00	00001100	0E100001	00000000	00000011	fff0K00P	00000000	HT000u0P
CIPHERXI	0000uu00	00001100	0E110001	00000000	00000011	fff0K00P	00000000	HT000u0P
DECIPHER	00000000	00000011	0E01000P	00000000	00000011	fff0K00P	00000000	HT000u0P
ENCIPHER	00000000	00000011	0E10000P	00000000	00000011	fff0K00P	00000000	HT000u0P
SECMSG	00000000	00001010	0E..000P	00000000	00000011	fff0K00P	00000000	HT000u0P
MAC	cccc0000	00000101	0E00110P	00000000	00000011	fff0K00P	00000000	HT000u0P
MACVER	cccc0000	00000101	0E00010P	00000000	00000011	fff0K00P	00000000	HT000u0P

CIPHERXI (0E110001): XDout, Xdin

SECMSG (0E..000P): 01 PIN encryption, 10 Key encryption

cccc: 0000 ANY, 0001 ANSI X9.9, 0010 CVV KEY-A, 0011 CVV KEY-B, 0100 AMEX-CSC

fff: Key-Form

Figure 18. Control Vector Base Bit Map (Data Operation Keys)

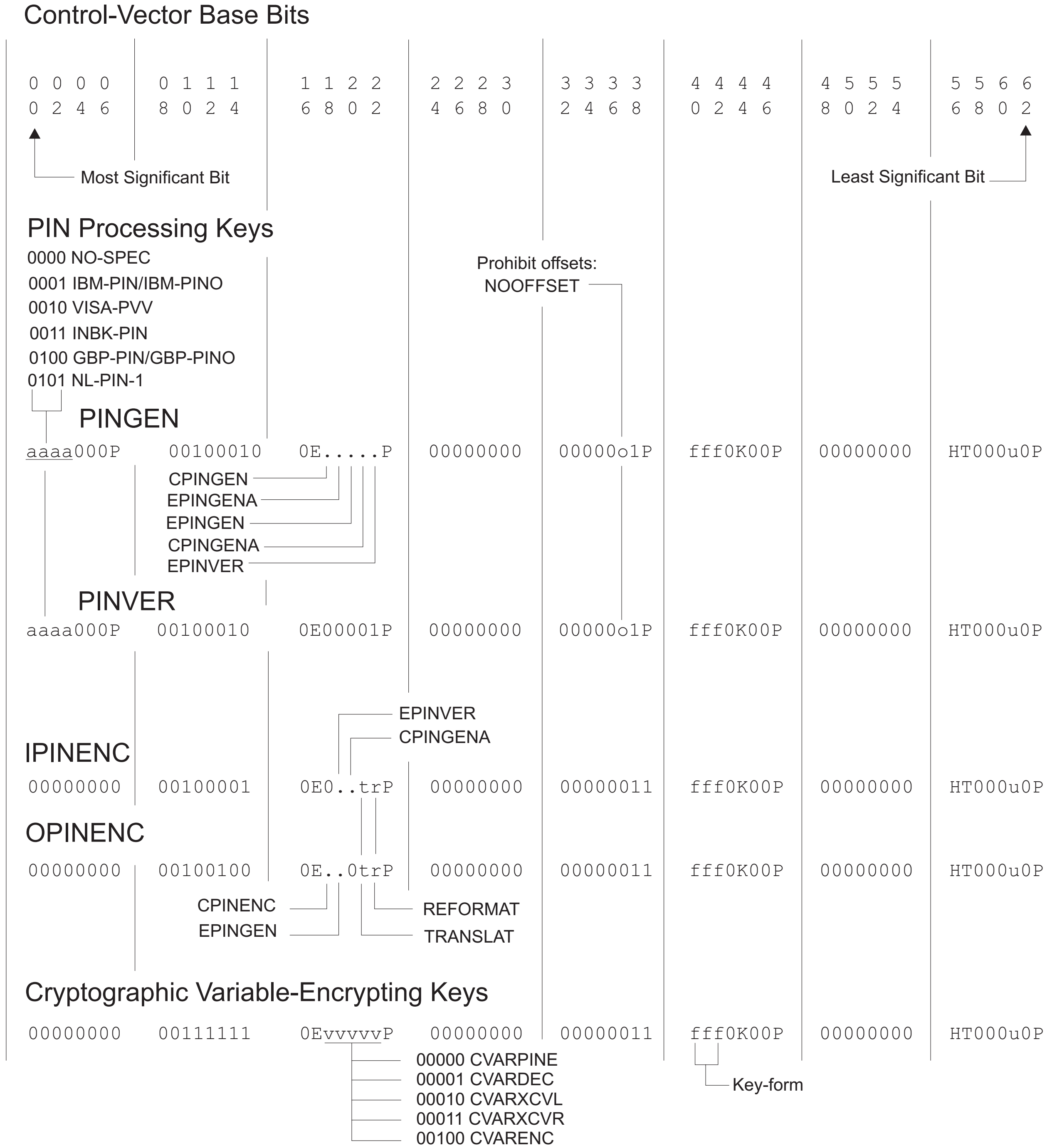

Figure 19. Control Vector Base Bit Map (PIN Processing Keys and Cryptographic Variable-Encrypting Keys)

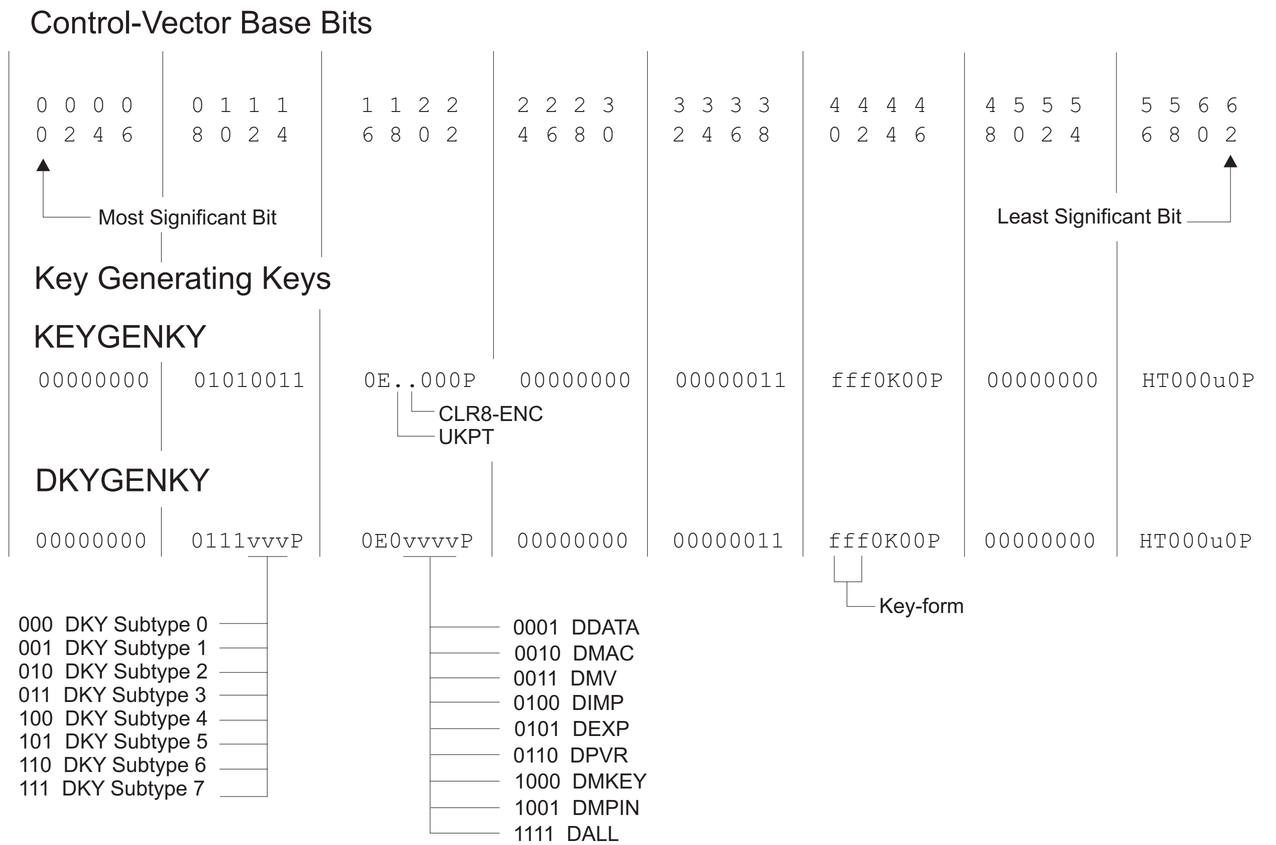

Figure 20. Control Vector Base Bit Map (Key Generating Keys)

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

010 Double length key, left half

001 Double length key. right half

110 Double-length key, left half, halves guaranteed unique

101 Double-length key, right half, halves guaranteed unique

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 445. 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

Table 445. Main Key Type for Bits 8 to 11 (continued)

Bits 8 to 11	Main Key Type
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 446. Key Subtype for Diversified Key Generating Keys

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 the Key-Encrypting Keys section of Figure 17 on page 1067). 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
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 18-22 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
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 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. 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.

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 21 on page 1077. 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 21 on page 1077.) Set the **A** bits to the value that you require for the corresponding control vector bits. In expressions 1 through 3 on page 1076, set the **B** bits to select the control vector bits to be evaluated. In expression 4 on page 1076, 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 **BOTH** 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.

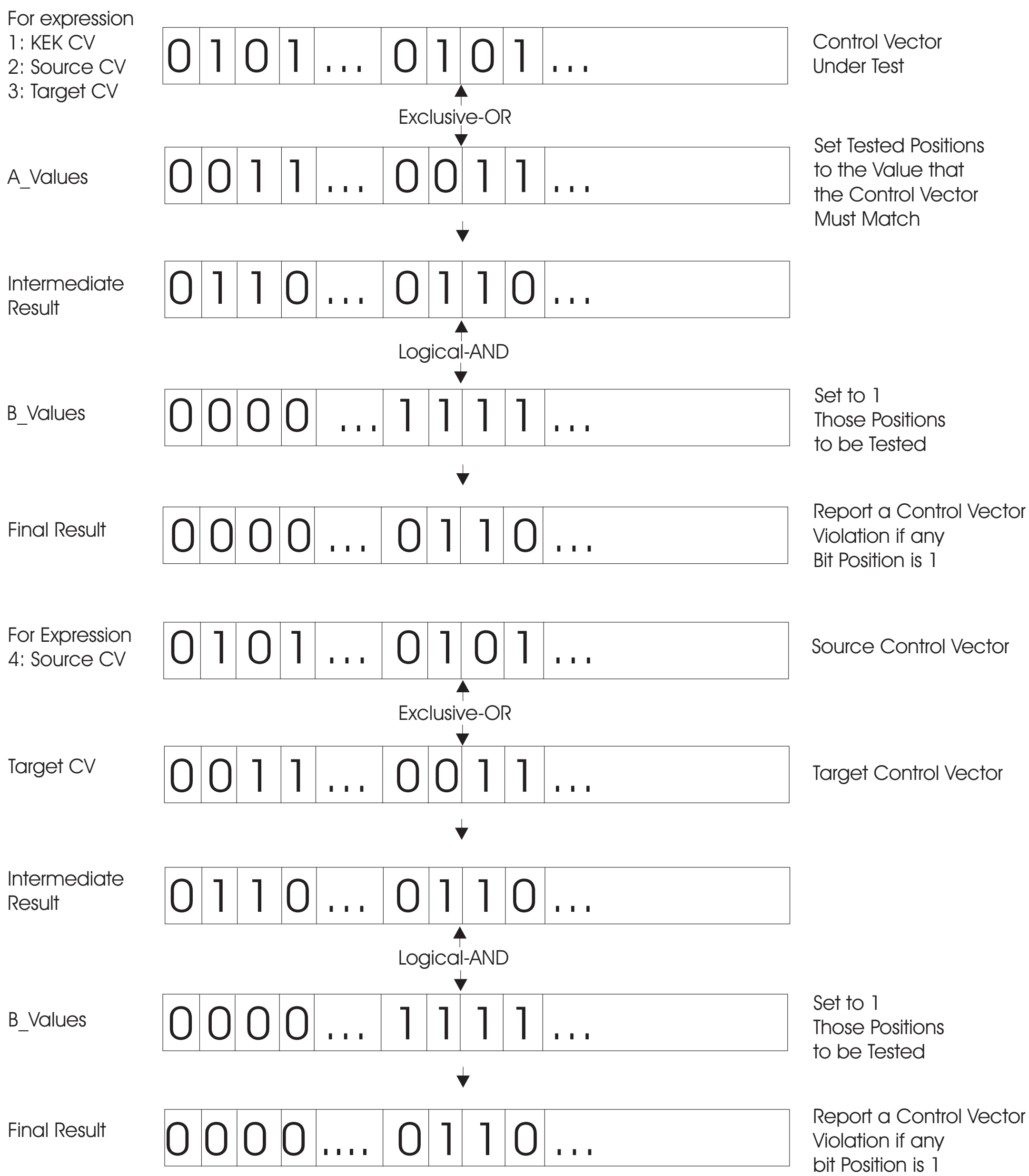

Figure 21. Control Vector Translate Callable Service Mask_Array Processing

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 22 on page 1078.

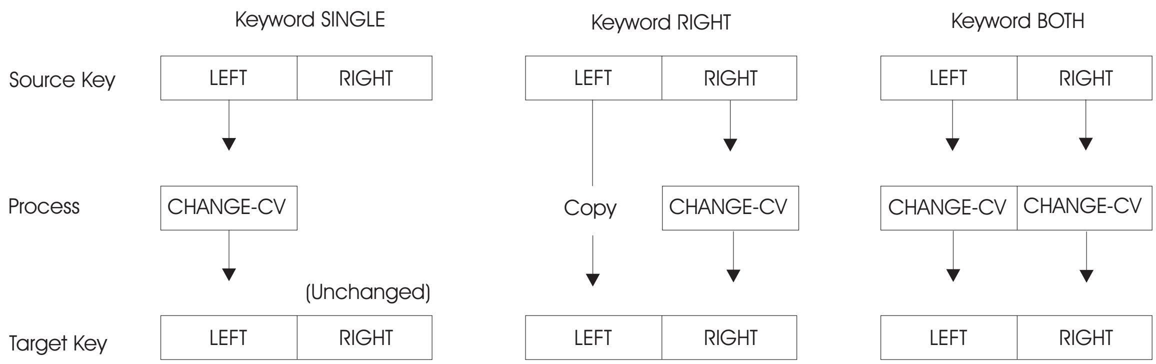

Figure 22. Control Vector Translate Callable Service. In this figure, CHANGE-CV means the requested control vector translation change; LEFT and RIGHT mean the left and right halves of a key and its control vector.

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 appendix provides sample routines using the ICSF callable services for these languages:
- C
- COBOL
- Assembler
- PL/1

The C, COBOL and Assembler H 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 HFS directory /usr/include and is copied to SYS1.SIEAHDR.H(CSFBEXT).

Information on creating C applications that call ICSF PKCS #11 services is available in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

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.
```

Assembler H

```
         TITLE 'SAMPLE ENCIPHER/DECIPHER S/370 PROGRAM.'
*======================================================================*
*       SYSTEM/370 ASSEMBLER H 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/1

```
/*********************************************************************/
/*                                                                   */
/* 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(( '02000040000300408000000000000000'X ||
                                '01AE28DA4606D885EB7E0340D6BAAC51'X ||
                                '991C0CD0EAE835AFD9CFF3CD7E7EA741'X ||
                                '41DADD24A6331BEDF41A6626522CCF15'X ||
                                '767D167D01A16F970100010252BDAD42'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;
```

Appendix E. Cryptographic Algorithms and Processes

This appendix 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', and X'2'.

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.

A = A 4-bit decimal digit that constitutes one digit of a user-specified constant.

PIN Block Formats

This section 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 = ZZZZAAAAAAAAAAAA
```

```
   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

```
   PIN Block =  CLPPPPrrrrrrrrRR

where C = X'3'
      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 = LPPPPffffffffFSS

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 section 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, 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 518 for an explanation of PIN extraction method keywords.

Encrypted PIN Translate 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, 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 section 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 23 illustrates the 3624 PIN generation algorithm.

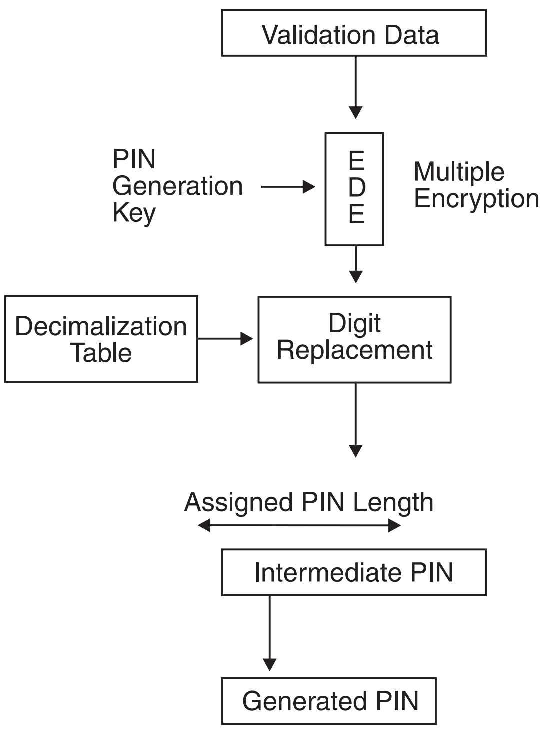

Figure 23. 3624 PIN Generation Algorithm

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 24 illustrates the German Banking Pool (GBP) PIN generation algorithm.

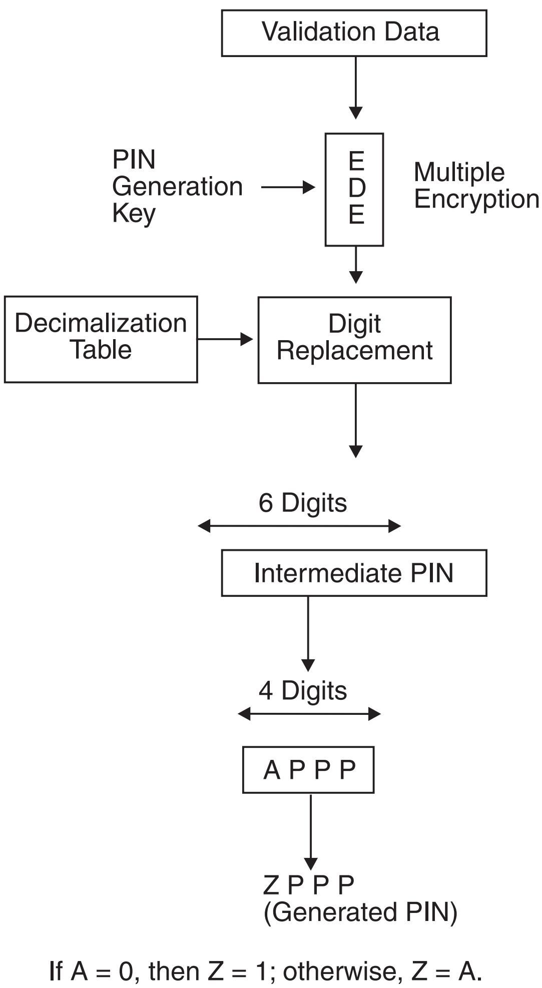

Figure 24. GBP PIN Generation Algorithm

PIN Offset Generation Algorithm

To allow the customer to select his own PIN, a PIN offsets is used by the IBM 3624 PIN generationN algorithms 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 25 illustrates the PIN offset generation algorithm.

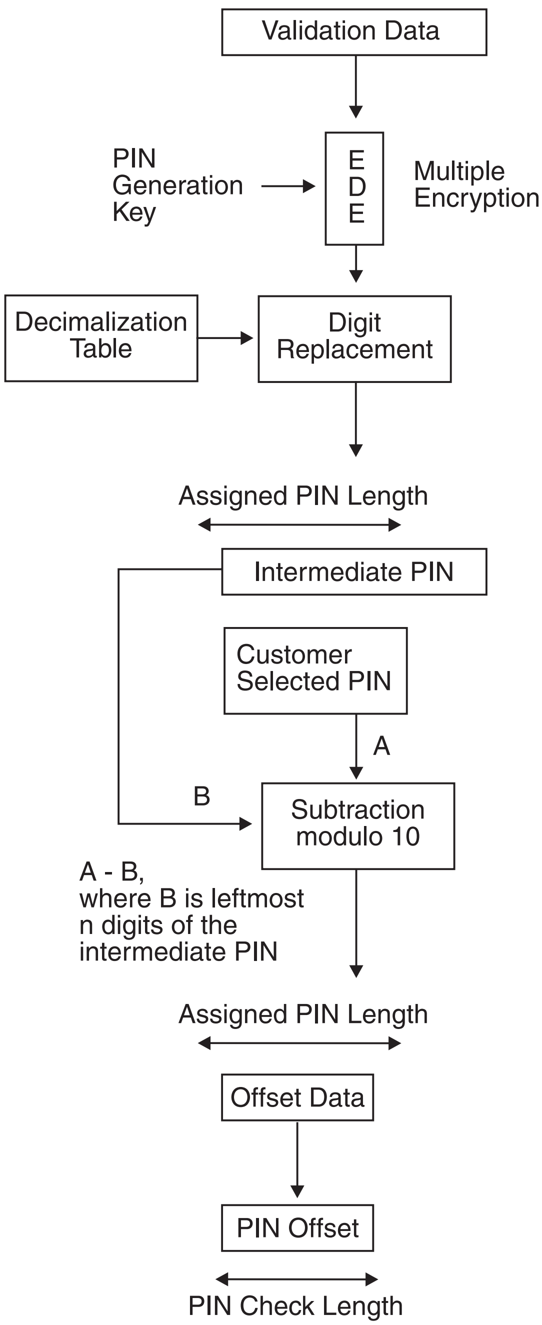

Figure 25. PIN-Offset Generation Algorithm

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 26 on page 1101 illustrates the PIN verification algorithm.

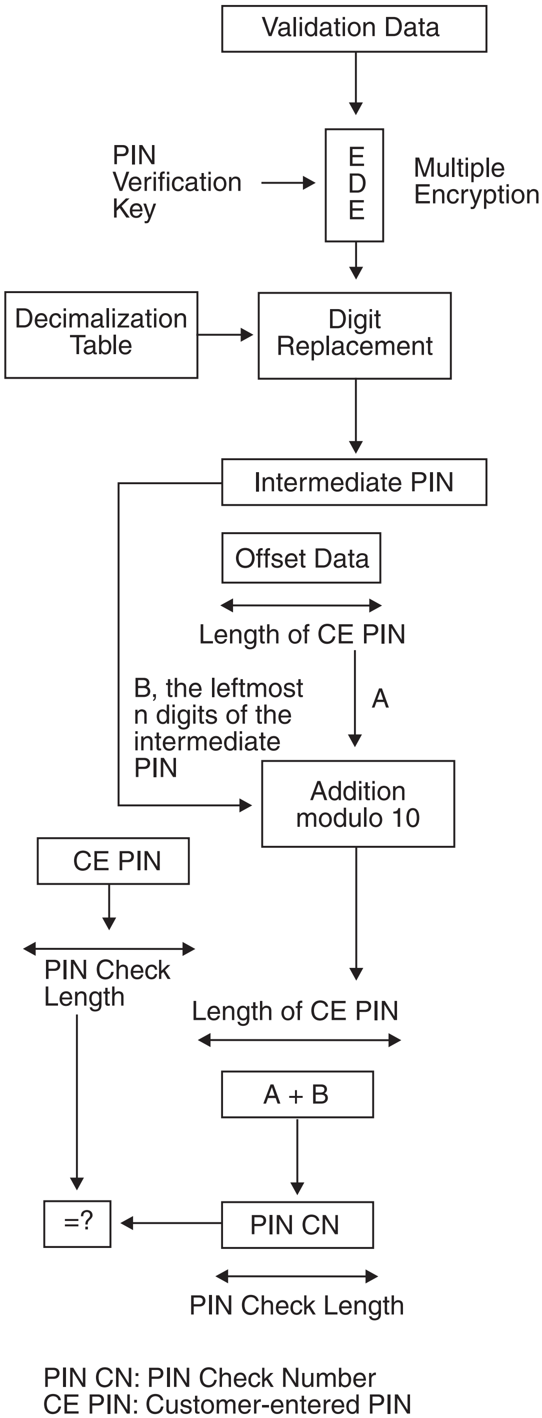

Figure 26. PIN Verification Algorithm

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 27 illustrates the GBP PIN verification algorithm.

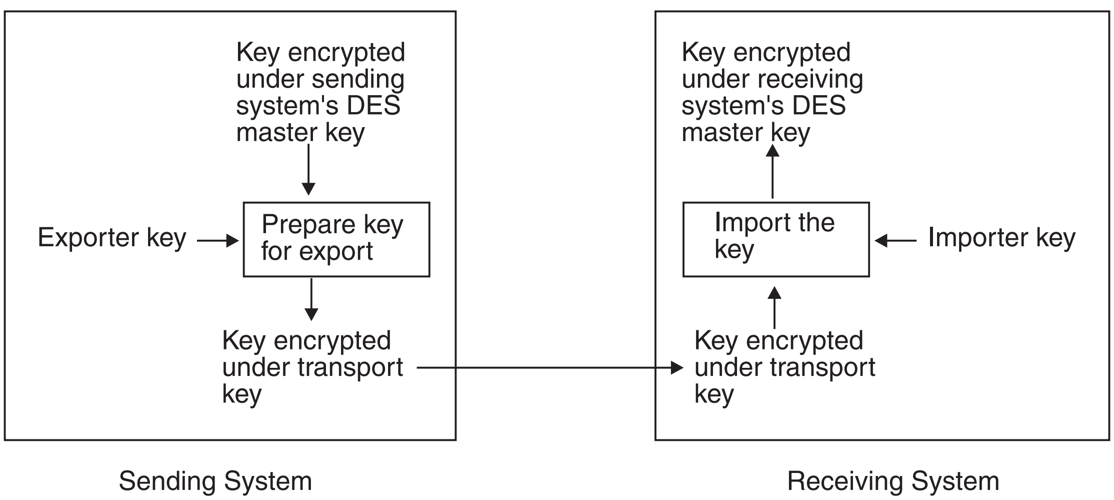

Figure 27. GBP PIN Verification Algorithm

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 28 illustrates the PVV generation algorithm.

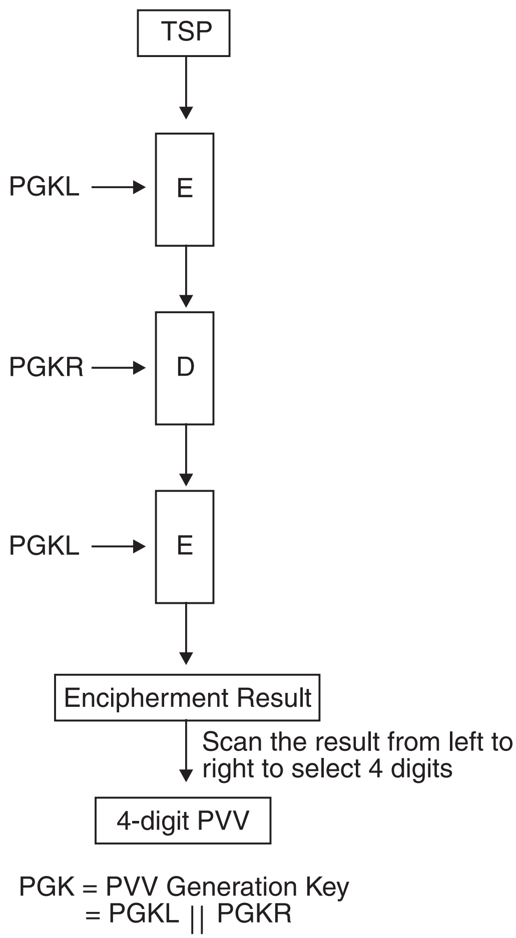

Figure 28. PVV Generation Algorithm

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 as that above, 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 as that above, 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 section 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. The table below 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.

Value of clear text length (mod 16)	Number of padding bytes added	Value of each padding byte
0	16	0x10
1	15	0x0F

Value of clear text length (mod 16)	Number of padding bytes added	Value of each padding byte
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 section explains how symmetric keys are wrapped with master and key-encrypting keys. For DES and AES keys, two methods are detailed. These use the 64-byte token. HMAC keys will use a variable length token with associated data and the payload wrapping method. In the future, all symmetric keys will be able to use the variable length token and the payload wrapping method.

ECB Wrapping of DES Keys (Original Method)

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

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 (Enhanced Method)

The enhanced CBC wrapping method uses triple DES encryption, an internal chaining of the key value and CBC mode.

The enhanced wrapping of a double-length key (*K) using a double-length *KEK is defined as follows:

```
e*KEK(*KL) = ecbcKEKL(dcbcKEKR(ecbcKEKL(KLPRIME || KR)))

KLPRIME = KL XOR SHA1(KR)
```

Where:

- KL is the left 64 bits of *K.
- KR is the right 64 bits of *K.
- KLPRIME is the 64 bit modified value of KL
- KEKL is the left 64 bits of *KEK.
- KEKR is the right 64 bits of *KEK.
- SHA1(X) is the 160-bit SHA-1 hash of X
- || means concatenation.
- XOR means bitwise exclusive OR

- ecbc means encryption using cipher block chaining mode
- dcbc means decryption using cipher block chaining mode

Wrapping key derivation for enhanced wrapping of DES keys

The wrapping key is exactly the same key that is used by CCA today, with one exception. Instead of using the base key itself (master key or key-encrypting key), ICSF will use a key that is derived from that base key. The derived key will have the control vector applied to it in the standard CCA manner, and then use the resulting key to wrap the new-format target key token. The reason for using a derived key is to ensure that no attacks against this wrapping scheme are possible using the existing CCA functions. For example, it was observed that an attack was possible by copying the wrapped key into an ECB CCA key token, if the wrapping key was used instead of a derivative of that key.

The key will be derived using a method defined in the NIST standard SP 800-108, "Recommendation for Key Derivation Using Pseudorandom Functions" (October, 2009). Derivation will use the method "KDF in Counter Mode" using pseudorandom function (PRF) HMAC-SHA256. This method provides sufficient strength for deriving keys for any algorithm used.

The HMAC algorithm is defined as follows:

```
HMAC(K, text) = H((K0 XOR opad ) || H((K0 XOR ipad) || text))
```

where opad is the constant 0x5C repeated to form a string the same length as K0, and ipad is the constant 0x36 repeated to form a string the same length as K0. If the key K is equal in length to the input block size of the hash function (512 bits for SHA-256), then K0 is set to the value of K. Otherwise, K0 is formed from K by hashing and/or padding.

The KDF specification calls for inputs optionally including two byte strings, Label and Context. The context will not be used. The label will contain information on the usage of this key, to distinguish it from other derivations that CCA may use in the future for different purposes. Since the security of the derivation process is rooted in the security of the derivation key and in the HMAC and KDF functions themselves, it is not necessary for this label string to be of any particular minimum size. The separation indicator byte of 0x00 specified in the NIST document will follow the label.

The label value will be defined so that it will be unique to derivation for this key wrapping process. This means that in any future designs which use the same KDF, ICSF must use a different value for the label. The label will be the 16 byte value consisting of the following ASCII characters:

```
ENHANCEDWRAP2010  (X'454E4841 4E434544 57524150 32303130')
```

The parameters for the counter mode KDF defined in SP 800-108 are as follows:

- Fixed values:
  - h (length of output of PRF) = 256 bits
  - r (length of the counter, in bits) = 32 - the counter will be an unsigned 4-byte value
- Inputs:
  - KI (input key) will be the key we are deriving from
  - Label will be the value shown above (ASCII ENHANCEDWRAP2010)
  - Separator byte of 0x00 will follow the label value

– Context will be a null string (no context is used)
– L will be the length of the derived key to be produced, rounded up to the next multiple of 256
– PRF (pseudorandom function) will be HMAC-SHA256

The KDF function will produce a pseudorandom bit string that is a multiple of 256 and will use as many bits of that as are required for the key to be produced. Bits for the key will be taken starting from the leftmost bit of the pseudorandom string, and any unused bits at the right will be discarded.

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 of 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 447 on page 1112 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 447. 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 447 on page 1112.

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

- A final byte of X'CC'.

PKCS #1 Formats

Version 2.0 of the PKCS #1 standard[4] defines methods for formatting keys and hashes prior to RSA encryption of the resulting data structures. The lower versions of the PKCS #1 standard 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 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. Keyword PKCSOAEP is 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.

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.

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 and EMV-related smart card formats and processes

The VISA and EMV specifications for performing secure messaging with an EMV compliant smart card are covered in these documents:

4. PKCS standards can be retrieved from http://www.rsasecurity.com/rsalabs/pkcs.

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.

- *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*

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 above-mentioned documentation, the relevant processes are described in these sections:
- "Deriving the smart-card-specific authentication code"
- "Constructing the PIN-block for transporting an EMV smart-card PIN"
- "Deriving the CCA TDES-XOR session key" on page 1116
- "Deriving the EMV TDESEMVn tree-based session key" on page 1116
- "PIN-block self-encryption" on page 1117

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 ECB-mode triple-DES encrypting:
   - The second 8 bytes of card-specific derivation data when 16 bytes have been supplied
   - The exclusive-OR of the supplied 8 bytes of derivation data with X'FFFFFFFF FFFFFFFF'
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 above 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 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 verb appends 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.

DES Algorithm (single- 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 RN
```

where:

- eK(x) - x is encrypted by key K using the DES algorithm
- KL is the left 128-bit clear key value of the key
- KR is the right 128-bit clear key value of the key (will be hex zero for a single length key)
- C is X''4545454545454545'
- KK is a 128-bit intermediate value
- RN is a 128-bit pseudo-random number
- VP is the 128-bit verification pattern

SHAVP1 Algorithm

This algorithm is used by the Key Test2 callable service to generate and verify the verification pattern.

```
VP = Trunc128( SHA256( KA || KT || KL || K ))
```

Where:

- VP is the 128-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 section presents tables showing EBCDIC to ASCII and ASCII to EBCDIC conversion tables. In the table headers, EBC refers to EBCDIC and ASC refers to ASCII.

Table 448 shows the EBCDIC to ASCII default conversion table.

Table 448. 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
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 449 shows the ASCII to EBCDIC default conversion table.

Table 449. 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
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 have require special consideration when enabling the access control point.

Table 450. Access control points affecting multiple services or requiring special consideration

Name	Callable Services	Notes	Usage
Allow weak DES wrap of RSA	CSNDPKG / CSNFPKG	When enabled, a weaker DES key-encrypting key is allowed to wrap an RSA private key token. The Prohibit weak wrap – Transport keys access control point must be enabled and this access control point will override the restriction. See "Key Strength and Wrapping of Key" on page 82 for more information.	DD, SC

Table 450. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable Services	Notes	Usage
ANSI X9.8 PIN - Allow modification of PAN	CSNBPTR / CSNEPTR	See "ANSI X9.8 PIN Restrictions" on page 495 for a description of this control.	DD, SC
ANSI X9.8 PIN - Allow only ANSI PIN blocks	CSNBPTR / CSNEPTR	See "ANSI X9.8 PIN Restrictions" on page 495 for a description of this control.	DD, SC
ANSI X9.8 PIN - Enforce PIN block restrictions	CSNBCPA / CSNECPA, CSNBPTR / CSNEPTR, and CSNBSPN / CSNESPN	See "ANSI X9.8 PIN Restrictions" on page 495 for a description of this control.	DD, SC
ANSI X9.8 PIN – Use stored decimalization tables only	CSNBPGN / CSNEPGN, CSNBCPA / CSNECPA, CSNBEPG / CSNEEPG and CSNBPVR / CSNEPVR	See "ANSI X9.8 PIN Restrictions" on page 495 for a description of this control.	DD, SC
DATAM Key Management Control	CSNBKGN / CSNEKGN, CSNBKIM / CSNEKIM, CSNBKEX / CSNEKEX and CSNBDKG / CSNEDKG	When enabled, the DATAM and DATAMV key types can be used. When disabled, the key types are not allowed.	ED
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 27 for more information.	DD, SC
Enhanced PIN Security	CSNBCPE / CSNECPE, CSNBCPA / CSNECPA, CSNBEPG / CSNEEPG, CSNBPTR / CSNEPTR, CSNBPVR / CSNEPVR, and CSNBPCU / CSNEPCU	"Enhanced PIN Security Mode" on page 500 for a description of this control	DD, SC
NOCV KEK usage for export-related functions	CSNBKEX / CSNEKEX, CSNBSKM / CSNESKM, and CSNBKGN / CSNEKGN	When enabled, NOCV key-encrypting keys can be used by the listed services.	ED, SC

Table 450. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable Services	Notes	Usage
NOCV KEK usage for import-related functions	CSNBKIM / CSNEKIM, CSNBSKI / CSNESKI, CSNBSKM / CSNESKM, and CSNBKGN / CSNEKGN	When enabled, NOCV key-encrypting keys can be used by the listed services.	ED, SC
Prohibit weak wrap – 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 27 and "Key Strength and Wrapping of Key" on page 82 for more information.	DD, SC
Prohibit weak wrap – 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 27 for more information.	DD, SC
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.	AE
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.	AE
Symmetric token wrapping - internal enhanced method	Services that wrap internal symmetric key tokens	When enabled, this control will cause all generated or imported keys to be wrapped with the enhanced method. This control can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 27 for more information.	DD, SC
Symmetric token wrapping - internal original method	Services that wrap internal symmetric key tokens	When enabled, this control will cause all generated or imported keys to be wrapped with the original method. This control can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 27 for more information.	ED
Symmetric token wrapping - external enhanced method	Services that wrap external symmetric key tokens	When enabled, this control will cause all generated or exported keys to be wrapped with the enhanced method. This control can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 27 for more information.	DD, SC

Table 450. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable Services	Notes	Usage
Symmetric token wrapping - external original method	Services that wrap external symmetric key tokens	When enabled, this control will cause all generated or exported keys to be wrapped with the original method. This control can be overridden by rule array keywords for certain services. See "Key strength and wrapping of key" on page 27 for more information.	ED
UKPT - PIN Verify, PIN Translate	CSNBPVR / CSNEPVR and CSNBPTR / CSNEPTR	When enabled, the listed services can use UKPT key derivation.	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 27 and "Key Strength and Wrapping of Key" on page 82 for more information.	DD. SC
Warn when weak wrap – Key-encrypting 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 27 and "Key Strength and Wrapping of Key" on page 82 for more information.	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 wrap – 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 451. Access control points – Callable Services

Name	Callable Service	Usage
Authentication Parameter Generate	CSNBAPG / CSNEAPG	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Authentication Parameter Generate - Clear	CSNBAPG / CSNEAPG	DD
Cipher Text translate2	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	ED
Cipher Text translate2 – Allow translate from AES to TDES	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	ED
Cipher Text translate2 – Allow translate to weaker AES	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	ED
Cipher Text translate2 – Allow translate to weaker DES	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	ED
Cipher Text translate2 – Allow only cipher text translate types	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	DD
Clear Key Import / Multiple Clear Key Import - DES	CSNBCKI / CSNECKI and CSNBCKM / CSNECKM	ED
Clear PIN Encrypt	CSNBCPE / CSNECPE	ED
Clear PIN Generate - 3624	CSNBPGN / CSNEPGN	ED
Clear PIN Generate - GBP	CSNBPGN / CSNEPGN	ED
Clear PIN Generate - VISA PVV	CSNBPGN / CSNEPGN	ED
Clear PIN Generate - Interbank	CSNBPGN / CSNEPGN	ED
Clear Pin Generate Alternate - 3624 Offset	CSNBCPA / CSNECPA	ED
Clear PIN Generate Alternate - VISA PVV	CSNBCPA / CSNECPA	ED
Control Vector Translate	CSNBCVT / CSNECVT	ED
Cryptographic Variable Encipher	CSNBCVE / CSNECVE	ED
CVV Key Combine	CSNBCKC / CSNECKC	ED
CVV Key Combine - Allow wrapping override keywords	CSNBCKC / CSNECKC	ED
CVV Key Combine - Permit mixed key types	CSNBCKC / CSNECKC	ED
Data Key Export	CSNBDKX / CSNEDKX	ED
Data Key Export - Unrestricted	CSNBDKX / CSNEDKX	ED
Data Key Import	CSNBDKM / CSNEDKM	ED
Data Key Import - Unrestricted	CSNBDKM / CSNEDKM	ED
Decipher - DES	CSNBDEC / CSNEDEC	ED
Digital Signature Generate	CSNDDSG / CSNFDSG	ED
DSG - ZERO-PAD restriction lifted	CSNDDSG / CSNFDSG	ED
Digital Signature Verify	CSNDDSV / CSNFDSV	ED
Diversified Key Generate - CLR8–ENC	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - SESS-XOR	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - TDES-ENC	CSNBDKG / CSNEDKG	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Diversified Key Generate - TDES-CBC	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - TDES-DEC	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - TDES-XOR	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - TDESEMV2/TDESEMV4	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - Allow wrapping override keywords	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - single length or same halves	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - DKYGENKY - DALL	CSNBDKG / CSNEDKG	DD, SC
Diversified Key Generate2 – AES EMV1 SESS	CSNBDKG2 / CSNEDKG2	ED
Diversified Key Generate2 - DALL	CSNBDKG2 / CSNEDKG2	DD, SC
DK Deterministic PIN Generate	CSNBDDPG / CSNEDDPG	DD
DK Migrate PIN	CSNBDMP / CSNEDMP	DD
DK PAN Modify in Transaction	CSNBDPMT / CSNEDPMT	DD
DK PAN Translate	CSNBDPT / CSNEDPT	DD
DK PIN Verify	CSNBDPV / CSNEDPV	DD
DK PIN Change	CSNBDPC / CSNEDPC	DD
DK PRW Card Number Update	CSNBPNU / CSNEPNU	DD
DK PRW CMAC Generate	CSNBDPCG / CSNBPCG	DD
DK Random PIN Generate	CSNBDRPG / CSNEDRPG	DD
DK Regenerate PRW	CSNBDRP / CSNEDRP	DD
ECC Diffie-Hellman	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 192	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 224	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 256	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 384	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 521	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 160	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 192	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 224	CSNDEDH / CSNFEDH	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
ECC Diffie-Hellman – Allow BP Curve 256	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 320	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 384	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 512	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow PASSTHRU	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow key wrap override	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Prohibit weak key generate	CSNDEDH / CSNFEDH	DD, SC
Encipher - DES	CSNBENC / CSNEENC	ED
Encrypted PIN Generate - 3624	CSNBEPG / CSNEEPG	ED
Encrypted PIN Generate - GBP	CSNBEPG / CSNEEPG	ED
Encrypted PIN Generate - Interbank	CSNBEPG / CSNEEPG	ED
Encrypted PIN Translate - Translate	CSNBPTR / CSNEPTR	ED
Encrypted PIN Translate - Reformat	CSNBPTR / CSNEPTR	ED
Encrypted PIN Verify - 3624	CSNBPVR / CSNEPVR	ED
Encrypted PIN Verify - GPB	CSNBPVR / CSNEPVR	ED
Encrypted PIN Verify - VISA PVV	CSNBPVR / CSNEPVR	ED
Encrypted PIN Verify - Interbank	CSNBPVR / CSNEPVR	ED
FPE Decipher	CSNBFPED / CSNEFPED	ED
FPE Encipher	CSNBFPEE / CSNEFPEE	ED
FPE Translate	CSNBFPET / CSNEFPET	ED
HMAC Generate – SHA-1	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Generate – SHA-224	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Generate – SHA-256	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Generate – SHA-384	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Generate – SHA-512	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Verify – SHA-1	CSNBHMV / CSNBHMV1 and CSNEHMV / CSNEHMV1	ED
HMAC Verify – SHA-224	CSNBHMV / CSNBHMV1 and CSNEHMV / CSNEHMV1	ED
HMAC Verify – SHA-256	CSNBHMV / CSNBHMV1 and CSNEHMV / CSNEHMV1	ED
HMAC Verify – SHA-384	CSNBHMV / CSNBHMV1 and CSNEHMV / CSNEHMV1	ED
HMAC Verify – SHA-512	CSNBHMV / CSNBHMV1 and CSNEHMV / CSNEHMV1	ED
Key Export	CSNBKEX / CSNEKEX	ED
Key Export - Unrestricted	CSNBKEX / CSNEKEX	ED
Key Generate – OP	CSNBKGN / CSNEKGN	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Key Generate – Key set	CSNBKGN / CSNEKGN	ED
Key Generate – Key set extended	CSNBKGN / CSNEKGN	ED
Key Generate - SINGLE-R	CSNBKGN / CSNEKGN	ED
Key Generate2 – DK PIN admin1 key MAC	CSNBKGN2 / CSNEKGN2	DD
Key Generate2 – DK PIN admin1 key PINPROT	CSNBKGN2 / CSNEKGN2	DD
Key Generate2 – DK PIN admin2 key MAC	CSNBKGN2 / CSNEKGN2	DD
Key Generate2 – DK PIN key set	CSNBKGN2 / CSNEKGN2	DD
Key Generate2 – DK PIN print key	CSNBKGN2 / CSNEKGN2	DD
Key Generate2 – Key set	CSNBKGN2 / CSNEKGN2	ED
Key Generate2 – Key set extended	CSNBKGN2 / CSNEKGN2	ED
Key Generate2 – OP	CSNBKGN2 / CSNEKGN2	ED
Key Import	CSNBKIM / CSNEKIM	ED
Key Import - Unrestricted	CSNBKIM / CSNEKIM	ED
Key Part Import - First key part	CSNBKPI / CSNEKPI	ED
Key Part Import - Middle and final	CSNBKPI / CSNEKPI	ED
Key Part Import - ADD-PART	CSNBKPI / CSNEKPI	ED
Key Part Import - COMPLETE	CSNBKPI / CSNEKPI	ED
Key Part Import - Allow wrapping override keywords	CSNBKPI / CSNEKPI	ED
Key Part Import - Unrestricted	CSNBKPI / CSNEKPI	ED
Key Part Import2 – Load first key part, require 3 key parts	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 – Load first key part, require 2 key parts	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 - Load first key part, require 1 key parts	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 - Add second of 3 or more key parts	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 - Add last required key part	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 - Add optional key part	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 – Complete key	CSNBKPI2 / CSNEKPI2	ED
Key Test and Key Test2	CSNBKYT / CSNEKYT and CSNBKYT2 / CSNEKYT2	AE
Key Test2 – AES, ENC-ZERO	CSNBKYT2 / CSNEKYT2	AE
Key Test - Warn when keyword inconsistent with key length	CSNBKYTX / CSNFKYTX	DD
Key Translate	CSNBKTR / CSNEKTR	ED
Key Translate2	CSNBKTR2 / CSNEKTR2	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Key Translate2 - Allow use of REFORMAT	CSNBKTR2 / CSNEKTR2	ED
Key Translate2 - Allow wrapping override keywords	CSNBKTR2 / CSNEKTR2	ED
Key Translate2 - Disallow AES ver 5 to ver 4 conversion	CSNBKTR2 / CSNEKTR2	DD
Key Translate2 – Translate fixed to variable payload	CSNBKTR2 / CSNEKTR2	DD, SC
MAC Generate	CSNBMGN / CSNEMGN	ED
MAC Generate2 – AES CMAC	CSNBMGN2 / CSNEMGN2 / CSNBMGN3 / CSNEMGN3	ED
MAC Verify	CSNBMVR / CSNEMVR	ED
MAC Verify2 – AES CMAC	CSNBMVR2 / CSNEMVR2 / CSNBMVR3 / CSNEMVR3	ED
Multiple Clear Key Import / Multiple Secure Key Import - AES	CSNBCKM / CSNECKM and CSNBSKM / CSNESKM	ED
Multiple Clear Key Import - Allow wrapping override keywords	CSNBCKM / CSNECKM	ED
Multiple Secure Key Import - Allow wrapping override keywords	CSNBSKM / CSNESKM	ED
Operational Key Load	CSNBOKL / CSNEOKL	ED
Operational Key Load - Variable-Length Tokens	CSNBOKL / CSNEOKL	ED
PIN Change/Unblock - change EMV PIN with OPINENC	CSNBPCU / CSNEPCU	ED
PIN Change/Unblock - change EMV PIN with IPINENC	CSNBPCU / CSNEPCU	ED
PKA Decrypt	CSNDPKD / CSNFPKD	ED
PKA Encrypt	CSNDPKE / CSNFPKE	ED
PKA Key Generate	CSNDPKG / CSNFPKG	ED
PKA Key Generate – Clear RSA keys	CSNDPKG / CSNFPKG	ED
PKA Key Generate – Clear ECC keys	CSNDPKG / CSNFPKG	ED
PKA Key Generate - Clone	CSNDPKG / CSNFPKG	ED
PKA Key Generate - Permit Regeneration Data	CSNDPKG / CSNFPKG	ED
PKA Key Generate - Permit Regeneration Data Retain	CSNDPKG / CSNFPKG	ED
PKA Key Import	CSNDPKI / CSNFPKI	ED
PKA Key Import - Import an External Trusted Key Block to internal form	CSNDPKI / CSNFPKI	ED
PKA Key Token Change RTCMK	CSNDKTC / CSNFKTC	ED
PKA Key Translate - from CCA RSA to SC Visa format	CSNDPKT / CSNFPKT	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
PKA Key Translate - from CCA RSA to SC ME format	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from CCA RSA to SC CRT format	CSNDPKT / CSNFPKT	ED
PKA Key Translate – Translate internal key token	CSNDPKT / CSNFPKT	ED
PKA Key Translate – Translate external key token	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from source EXP KEK to target EXP KEK	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from source IMP KEK to target EXP KEK	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from source IMP KEK to target IMP KEK	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from CCA RSA CRT to EMVDDA format	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from CCA RSA CRT to EMVDDAE format	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from CCA RSA CRT to EMVCRT format	CSNDPKT / CSNFPKT	ED
Prohibit Export	CSNBPEX / CSNEPEX	ED
Prohibit Export Extended	CSNBPEXX /CSNEPEXX	ED
Recover PIN From Offset	CSNBPFO / CSNEPFO	ED
Remote Key Export - Generate or export a key for use by a non-CCA node	CSNDRKX / CSNFRKX	ED
Remote Key Export – Include RKX in Default Key-Wrapping Configuration	CSNDRKX / CSNFRKX	DD
Remote Key Export - Allow wrapping override keywords	CSNDRKX / CSNFRKX	DD
RKX/TBC – Disallow triple-length MAC key	CSNDRKX / CSNFRKX and CSNDTBC / CSNFTBC	DD, SC
Restrict Key Attribute – Export Control	CSNBRKA / CSNERKA	ED
Restrict Key Attribute - Permit setting the TR-31 export bit	CSNBRKA / CSNERKA	ED
Retained Key Delete	CSNDRKD / CSNFRKD	ED
Retained Key List	CSNDRKL / CSNFRKL	ED
Secure Key Import – DES, IM	CSNBSKI / CSNESKI and CSNBSKM / CSNESKM	ED
Secure Key Import – DES, OP	CSNBSKI / CSNESKI and CSNBSKM / CSNESKM	ED
Secure Key Import2 - OP	CSNBSKI2 / CSNESKI2	ED
Secure Key Import2 - IM	CSNBSKI2 / CSNESKI2	ED
Secure Messaging for Keys	CSNBSKY / CSNESKY	ED
Secure Messaging for PINs	CSNBSPN / CSNESPN	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
SET Block Compose	CSNDSBC / CSNFSBC	ED
SET Block Decompose	CSNDSBD / CSNFSBD	ED
SET Block Decompose - PIN ext IPINENC	CSNDSBD / CSNFSBD	ED
SET Block Decompose - PIN ext OPINENC	CSNDSBD / CSNFSBD	ED
Symmetric Algorithm Decipher - Secure AES	CSNBSAD / CSNESAD and CSNBSAD1 / CSNESAD1	ED
Symmetric Algorithm Encipher - Secure AES	CSNBSAE / CSNESAE and CSNBSAE1 / CSNESAE1	ED
Symmetric Key Encipher/Decipher - Encrypted DES keys	CSNBSYD / CSNBSYE and CSNBSYD1 / CSNESYD1	ED
Symmetric Key Encipher/Decipher - Encrypted AES keys	CSNBSYD / CSNBSYE and CSNBSYD1 / CSNESYD1	ED
Symmetric Key Export with Data	CSNDSXD / CSNFSXD	DD
Symmetric Key Export with Data - Special	CSNDSXD / CSNFSXD	DD
Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - AES, PKOAEP2	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - AES, ZERO-PAD	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - AESKW	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - AESKWCV	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - DES, PKCS-1.2	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - DES, ZERO-PAD	CSNDSYX / CSNFSYX	ED
Symmetric Key Export – HMAC,PKOAEP2	CSNDSYX / CSNFSYX	ED
Symmetric Key Generate - AES, PKCSOAEP, PKCS-1.2	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - AES, ZERO-PAD	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - DES, PKCS-1.2	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - DES, ZERO-PAD	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - DES, PKA92	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - Allow wrapping override keywords	CSNDSYG / CSNFSYG	ED
Symmetric Key Import - AES, PKCSOAEP, PKCS-1.2	CSNDSYI / CSNFSYI	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Symmetric Key Import - AES, ZERO-PAD	CSNDSYI / CSNFSYI	ED
Symmetric Key Import - DES, PKCS-1.2	CSNDSYI / CSNFSYI	ED
Symmetric Key Import - DES, ZERO-PAD	CSNDSYI / CSNFSYI	ED
Symmetric Key Import - DES, PKA92 KEK	CSNDSYI / CSNFSYI	ED
Symmetric Key Import - Allow wrapping override keywords	CSNDSYI / CSNFSYI	ED
Symmetric Key Import2 – AES,PKOAEP2	CSNDSYI2 / CSNFSYI2	ED
Symmetric Key Import2 - AESKW	CSNDSYI2 / CSNFSYI2	ED
Symmetric Key Import2 - AESKWCV	CSNDSYI2 / CSNFSYI2	ED
Symmetric Key Import2 - Allow wrapping override keywords	CSNDSYI2 / CSNFSYI2	ED
Symmetric Key Import2 - disallow weak import	CSNDSYI2 / CSNFSYI2	DD, SC
Symmetric Key Import2 – HMAC,PKOAEP2	CSNDSYI2 / CSNFSYI2	ED
TR31 Export – Permit version A TR-31 key blocks	CSNBT31X / CSNET31X	ED
TR31 Export – Permit version B TR-31 key blocks	CSNBT31X / CSNET31X	ED
TR31 Export – Permit version C TR-31 key blocks	CSNBT31X / CSNET31X	ED
TR31 Export – Permit any CCA key if INCL-CV is specified	CSNBT31X / CSNET31X	ED
TR31 Export – Permit KEYGENKY:UKPT to B0	CSNBT31X / CSNET31X	ED
TR31 Export – Permit MAC/MACVER:AMEXCSC to C0:G/C/V	CSNBT31X / CSNET31X	DD
TR31 Export – Permit MAC/MACVER:CVVKEYA to C0:G/C/V	CSNBT31X / CSNET31X	DD
TR31 Export – Permit MAC/MACVER:ANYMAC to C0:G/C/V	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DATA to C0:G/C	CSNBT31X / CSNET31X	ED
TR31 Export – Permit ENCIPHER/DECIPHER/CIPHER to D0:E/D/B	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DATA to D0:B	CSNBT31X / CSNET31X	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
TR31 Export – Permit EXPORTER/OKEYXLAT to K0:E	CSNBT31X / CSNET31X	DD
TR31 Export – Permit IMPORTER/IKEYXLAT to K0:D	CSNBT31X / CSNET31X	DD
TR31 Export – Permit EXPORTER/OKEYXLAT to K1:E	CSNBT31X / CSNET31X	DD
TR31 Export – Permit IMPORTER/IKEYXLAT to K1:D	CSNBT31X / CSNET31X	DD
TR31 Export – Permit MAC/DATA/DATAM to M0:G/C	CSNBT31X / CSNET31X	DD
TR31 Export – Permit MACVER/DATAMV to M0:V	CSNBT31X / CSNET31X	ED
TR31 Export – Permit MAC/DATA/DATAM to M1:G/C	CSNBT31X / CSNET31X	ED
TR31 Export – Permit MACVER/DATAMV to M1:V	CSNBT31X / CSNET31X	ED
TR31 Export – Permit MAC/DATA/DATAM to M3:G/C	CSNBT31X / CSNET31X	ED
TR31 Export – Permit MACVER/DATAMV to M3:V	CSNBT31X / CSNET31X	ED
TR31 Export – Permit OPINENC to P0/E	CSNBT31X / CSNET31X	ED
TR31 Export – Permit IPINENC to P0/D	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINVER:NO-SPEC to V0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit PINGEN:NO-SPEC to V0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit PINVER:NO-SPEC/IBM-PIN/IBM-PINO to V1	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINGEN:NO-SPEC/IBM-PIN/ IBM-PINO to V1	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINVER:NO-SPEC/VISA-PVV to V2	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINGEN:NO-SPEC/VISA-PVV to V2	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DKYGENKY:DKYL0+DMAC to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DMV to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DMAC to E0	CSNBT31X / CSNET31X	DD

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
TR31 Export – Permit DKYGENKY:DKYL1+DMV to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DALL to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DDATA to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DMPIN to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DDATA to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DMPIN to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DALL to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DMAC to E2	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E2	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DMAC to E2	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DALL to E2	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DATA/MAC/CIPHER/ENCIPHER to E3	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DDATA to E4	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E4	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DKYGENKY:DKYL0+DEXP to E5	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DMAC to E5	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DDATA to E5	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E5	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINGEN/PINVER to V0/V1/V2:N	CSNBT31X / CSNET31X	DD
TR31 Import – Permit version A TR-31 key blocks	CSNBT31I / CSNET31I	ED
TR31 Import – Permit version B TR-31 key blocks	CSNBT31I / CSNET31I	ED
TR31 Import – Permit version C TR-31 key blocks	CSNBT31I / CSNET31I	ED

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
TR31 Import – Permit override of default wrapping method	CSNBT31I / CSNET31I	ED
TR31 Import – Permit C0 to MAC/MACVER:CVVKEY-A	CSNBT31I / CSNET31I	DD
TR31 Import – Permit C0 to MAC/MACVER:AMEX-CSC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K0:E to EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K0:D to IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K0:B to EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K0:B to IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K1:E to EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K1:D to IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K1:B to EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K1:B to IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit M0/M1/M3 to MAC/MACVER:ANY-MAC	CSNBT31I / CSNET31I	ED
TR31 Import – Permit P0:E to OPINENC	CSNBT31I / CSNET31I	ED
TR31 Import – Permit P0:D to IPINENC	CSNBT31I / CSNET31I	ED
TR31 Import – Permit V0 to PINGEN:NO-SPEC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit V0 to PINVER:NO-SPEC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit V1 to PINGEN:IBM-PIN/IBM-PINO	CSNBT31I / CSNET31I	ED
TR31 Import – Permit V1 to PINVER:IBM-PIN/IBM-PINO	CSNBT31I / CSNET31I	ED
TR31 Import – Permit V2 to PINGEN:VISA-PVV	CSNBT31I / CSNET31I	ED
TR31 Import – Permit V2 to PINVER:VISA-PVV	CSNBT31I / CSNET31I	ED
TR31 Import – Permit E0 to DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E0 to DKYGENKY:DKYL0+DMV	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E0 to DKYGENKY:DKYL1+DMAC	CSNBT31I / CSNET31I	DD

Table 451. Access control points – Callable Services (continued)

Name	Callable Service	Usage
TR31 Import – Permit E0 to DKYGENKY:DKYL1+DMV	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E1 to DKYGENKY:DKYL0+DMPIN	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E1 to DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E1 to DKYGENKY:DKYL1+DMPIN	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E1 to DKYGENKY:DKYL1+DDATA	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E2 to DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E2 to DKYGENKY:DKYL1+DMAC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E3 to ENCIPHER	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E4 to DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	ED
TR31 Import – Permit E5 to DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E5 to DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E5 to DKYGENKY:DKYL0+DEXP	CSNBT31I / CSNET31I	DD
TR31 Import – Permit V0/V1/V2:N to PINGEN/PINVER	CSNBT31I / CSNET31I	DD
Transaction Validation – Generate	CSNBTRV / CSNETRV	ED
Transaction Validation - Verify CSC-3	CSNBTRV / CSNETRV	ED
Transaction Validation - Verify CSC-4	CSNBTRV / CSNETRV	ED
Transaction Validation - Verify CSC-5	CSNBTRV / CSNETRV	ED
Trusted Block Create - Activate an Inactive Trusted Key Block	CSNDTBC / CSNFTBC	ED
Trusted Block Create - Create Trusted Key Block in Inactive Form	CSNDTBC / CSNFTBC	ED
Unique Key Derive	CSNBUKD / CSNEUKD	ED
Unique Key Derive - Allow PIN-DATA processing	CSNBUKD / CSNEUKD	DD
Unique Key Derive - K3IPEK	CSNBUKD / CSNEUKD	DD
Unique Key Derive - Override default wrapping	CSNBUKD / CSNEUKD	ED
VISA CVV Generate	CSNBCSG / CSNECSG	ED
VISA CVV Verify	CSNBCSV / CSNECSV	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 TR31 Export - 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. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS® information, send a detailed message to the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html) 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

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM® Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, `3` or `3.1` or `3.1.1`. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the

default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included
The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. 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 give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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:

Site Counsel
IBM Corporation
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 information 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.

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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, 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 (http://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" at www.ibm.com/legal/copytrade.shtml.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Glossary

This glossary defines terms and abbreviations used in Integrated Cryptographic Service Facility (ICSF). If you do not find the term you are looking for, refer to the index of the appropriate Integrated Cryptographic Service Facility document or view IBM Terminology located at: IBM Glossary of Computing Terms (http://www.ibm.com/software/globalization/terminology/)

This glossary includes terms and definitions from:

- IBM Terminology definitions are identified by the symbol (D) after the definition.
- 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). (D)

Advanced Encryption Standard (AES)
In computer security, the National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) algorithm. The AES algorithm is documented in a draft Federal Information Processing Standard.

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. (D)

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. (D)

In ICSF, a callable service.

asymmetric cryptography
Synonym for public key cryptography. (D)

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. (D)

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'.

CEX2A
Crypto Express2 Accelerator

CEX2C
Crypto Express2 Coprocessor

CEX3A
Crypto Express3 Accelerator

CEX3C
Crypto Express3 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. (D)

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. (D)

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. (D)

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. (D)

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 z890, z990, z9 EC, z9 BC, z10 EC and z10 BC processors to provide 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 Express2 Coprocessor
An asynchronous cryptographic coprocessor available on the z890, z990, z9 EC, z9 BC, z10 EC and z10 BC.

Crypto Express3 Coprocessor
An asynchronous cryptographic coprocessor available on z10 EC and z10 BC.

cryptographic adapter (4755 or 4758)
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. (D)

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. (D)

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. (D)

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. (D)

data key or data-encrypting key
A key used to encipher, decipher, or authenticate data. (D)

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. (D)

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. (D)

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).

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.

double-length key
A key that is 128 bits long. A key can be either double- or single-length. A single-length key is 64 bits long.

DSA Digital Signature Algorithm.

DSS Digital Signature Standard.

ECB Electronic codebook.

ECI Euroch⌂que 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. (D)

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. (D)

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. (D)

enciphered data
Data whose meaning is concealed from unauthorized users or observers. (D)

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. (D)

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. (D)

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. (D)

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. (D)

key-encrypting key (KEK)
: In computer security, a key used for encryption and decryption of other keys. (D)

: 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.

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.

MD4 Message Digest 4. A hash algorithm.

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. (D)

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. (D)

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. (D)

PCICA
PCI Cryptographic Accelerator.

PCI X Cryptographic Coprocessor
An asynchronous cryptographic coprocessor available on the IBM eServer zSeries 990 and IBM eServer zSeries 800.

PCIXCC
PCI X Cryptographic Coprocessor.

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. (D)

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. (D)

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.

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. (D)

retained key
A private key that is generated and retained within the secure boundary of the PCI Cryptographic Coprocessor.

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. (D)

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.

SAF Security 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. (D)

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. (D)

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-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.

single-length key
A key that is 64 bits long. A key can be single- or double-length. A double-length key is 128 bits long.

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. (D)

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. (D)

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 128-bit key used to protect keys distributed from one system to another. A transport key can either be an exporter key-encrypting key, an importer key-encrypting key, or an ANSI key-encrypting key.

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. (D)

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.

4753 The Network Security processor. The IBM 4753 is a processor that uses the Data Encryption Algorithm and the RSA algorithm to provide cryptograpic support for systems requiring secure transaction processing (and other cryptographic services) at the host computer. The NSP includes a 4755 cryptographic adapter in a workstation which is channel attached to a S/390 host computer.

4758 The IBM PCI Cryptographic processor provides a secure programming and hardware environment where DES and RSA processes are performed.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

W

X

Printed in USA

SC14-7508-03