z/OS
Cryptographic Services
Integrated Cryptographic Service Facility

PKA Key Management Extensions – APAR OA28855

Contents

Chapter 1. Overview

This document update describes PKA Key Management Extensions, and contains alterations to information previously presented in the following books:

- *z/OS Cryptographic Services ICSF Overview*, SA22-7519-12
- *z/OS Cryptographic Services ICSF Administrator's Guide*, SA22-7521-13
- *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522-12
- *z/OS Cryptographic Services ICSF System Programmer's Guide*, SA22-7520-13
- *z/OS Cryptographic Services ICSF Messages*, SA22-7523-12

The preceding books document capabilities provided by FMID HCR7751, and support z/OS Version 1 Release 10.

Technical changes or additions related to PKA Key Management Extensions in this document update are indicated by a vertical line to the left of the change.

These updates relate to the enhancements made to the ICSF product by the application of APAR OA28855. For full functionality on z/OS V1R8, V1R9 and V1R10, you must also apply APAR OA28437 (for SAF) and APAR OA28439 (for RACF). For full functionality on z/OS V1R11, you must also apply APAR OA28581 (for SAF) and APAR OA28580 (for RACF).

Chapter 2. Update of z/OS Cryptographic Services ICSF Overview, SA22-7519-12, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF Overview*, SA22-7519-12, for the PKA Key Management Extensions enhancements provided by this APAR. Refer to this source document if background information is needed.

Security

In reviewing your installation security plan before installing ICSF, consider these points:

- **Controlling Access to Disk Copies of the CKDS**

  You should determine which users and applications should have access to each copy of the CKDS on your system.

  Note: The in-storage copy of the CKDS can be accessed only through ICSF functions such as callable services, KGUP, or the ICSF panels. To protect the in-storage copy of the CKDS, control who can use these services.

- **Controlling Access to the PKDS**

  You should determine which users and applications should have access to the PKDS on your system.

  Note: The in-storage copy of the PKDS can be accessed only through ICSF functions such as callable services or the ICSF panels. To protect the in-storage copy of the PKDS, control who can use these services.

- **Controlling Access to the Key Generator Utility Program (KGUP)**

  Anyone who is running the KGUP can read and change an unprotected CKDS. To prevent unauthorized persons from using the KGUP, store the program in an APF-authorized library that is protected by the Security Server (RACF). KGUP is also protected by CSFSERV(CSFKGUP).

- **Controlling Access to Services and Keys**

  Users of the Security Server (RACF) can use the CSFSERV and CSFKEYS classes to perform access checking and auditing of services and keys, respectively. The CSFSERV class also protects some critical administrative TSO panels, such as changing the master key and refreshing the CKDS. The audit records that are produced by these routines are SMF type 80 records.

  You can also define profiles in the XFACILIT class to establish a Key Store Policy. Each profile you define in the XFACILIT class is a separate Key Store Policy control. Together, these profiles define your overall Key Store Policy. By establishing a Key Store Policy, you can control access to secure symmetric keys in the CKDS and asymmetric keys in the PKDS. A Key Store Policy can also specify how keys in a PKDS or CKDS can be used. By enabling Key Store Policy controls, you can:

  – have ICSF verify, when an application passes a callable service a key token instead of a key label, that the user has authority to the secure token. ICSF does this be identifying key labels associated with the key token, so that a SAF authorization check (which depends on key labels) can be carried out against profiles in the CSFKEYS class.

  – prevent applications from storing duplicate tokens in a CKDS or PKDS.

– raise the level of access authority required to create, write to, or delete a key label.
– raise the level of access authority required to export a symmetric key (transfer it from encryption under a master key to encryption under an application-supplied RSA public key) when an application calls the Symmetric Key Export callable service (CSNDSYX or CSNFSYX). In this case, a SAF authorization check is performed against profiles in the XCSFKEY class rather than the CSFKEYS class.
– Set additional restrictions on how keys can be used. These additional restrictions are specified in the ICSF segment of CSFKEYS (or XCSFKEY) profiles. Using the ICSF segment of profiles in these classes, you can:
  - specify that asymmetric keys covered by the profile can not be used for secure export or import operations.
  - specify that asymmetric keys covered by the profile can not be used in the handshake operations performed by the following callable services:
    • Digital Signature Generate (CSNDDSG and CSNFDSG)
    • Digital Signature Verify (CSNDDSV and CSNFDSV)
    • PKA Encrypt (CSNDPKE and CSNFPKE)
    • PKA Decrypt (CSNDPKD and CSNFPKD)
  - specify whether symmetric keys covered by the profile can be exported using the Symmetric Key Export callable service (CSNDSYX or CSNFSYX). If allowing the symmetric keys covered by the profile to be exported, you can specify which asymmetric keys can be used to perform the export operation. You can specify this by supplying a list of labels of RSA keys in the PKDS, or a list of certificates in either a PKCS #11 token, or a SAF key ring.

You should familiarize yourself with the controls you can enable and decide on the Key Store Policy that is best for your installation.

• **Scheduling Changes for Cryptographic Keys**

To reduce the possibility of exposing a key value, you may want to change the value of cryptographic keys, including master keys, from time to time:

– You can use the ICSF panels to change the DES, AES and PKA master keys.
– If you have an optional Trusted Key Entry (TKE) workstation installed, you can use it to change DES, AES and PKA master keys on all cryptographic coprocessors.
– You can use KGUP or the ICSF panel to change the CKDS.
– You can develop applications that use the dynamic CKDS update callable services to change both the in-storage and DASD copies of the CKDS.

You can perform all of these operations without interrupting cryptographic functions.

• **Allowing or Preventing Clear Cryptographic Keys**

With ICSF, keys exist in the clear only in these cases:

– if you specifically allow special secure mode *and* actually set special secure mode during operations, applications can use the secure key import callable service, the clear PIN generate callable service, the clear PIN generate alternate callable service and the multiple secure key import and symmetric key generate with the IM keyword. On a z990, z890, z9 EC, z9 BC, z10 EC and z10 BC with a PCIXCC/CEX2C, access control points for these services must be enabled.

- If ICSF is not in special secure mode, most keys in the system are encrypted except DATA keys that a user may enter through the use of the clear key import callable service. CLRAES and CLRDES keys are also not encrypted.

  Note: The clear key import callable service is equivalent to the PCF EMK macro.
- The encode callable service can use a clear key to encipher data.
- If you use the Master Key Entry panels to enter the key parts of a master key, the key parts appear briefly in the clear in host storage.
- When an application calls the symmetric key generate callable service to generate a DATA key, the DATA key appears briefly in the clear in host storage when executed on a CCF. The DATA key is then quickly encrypted under the DES master key and the RSA public key.
- When an application calls the symmetric key export callable service to transfer a DATA key from encryption under the host DES master key to encryption under an RSA public key, the DATA key appears briefly in the clear in host storage when executed on a CCF.
- When an application calls the SET block compose and SET block decompose callable services, the DATA key exists briefly in the clear in host storage when executed on a CCF.
- On the z890, z990, z9 EC, z9 BC, z10 EC and z10 BC clear keys are used to provide improved performance for the DES, TDES and AES algorithms. Symmetric key encrypt and decrypt services (CSNBSYD and CSNBSYE) are available on all CP's.

On a CCF system, these services will be routed to the PCI Cryptographic Coprocessor if one is available: CSNDSYG, CSNDSYX, CSNDSBC, CSNDSYI, CSNDSBD, CSNBSKY, and CSNBSPN. If no PCI Cryptographic Coprocessor is available, then keys will appear briefly in the clear as stated previously.

- **Sending Cryptographic Keys to Other Installations**

  To eliminate the need to have a courier deliver clear keys between installations, you can use either or both of these options:
  - DES transport keys to encrypt keys for network distribution
  - The receiving installation's RSA public key to encrypt a DES or AES DATA key prior to electronic distribution

  Both of these methods make key distribution more secure.
- **Controlling access to the Disk Copies**

  You should determine which users and applications should have access to each DASD copy of the CKDS, PKDS and TKDS on your system.
- **SMF Records Generated by ICSF**

  ICSF generates type 82 records in the SMF data set when these conditions occur:
  - ICSF starts
  - ICSF status changes on a processor
  - When you enable or disable special secure mode
  - When you enter a clear master key part to the Cryptographic Coprocessor Feature through the use of the ICSF panels
  - You enter a master key part
  - When the in-storage CKDS is refreshed
  - When the in-storage PKDS is refreshed
  - You use the ICSF panels to process an operational key part or key part register loaded using the TKE workstation
  - When an application uses any of the dynamic services that write to the CKDS

- When ICSF handles error conditions or tampering
- When you issue a command from the TKE workstation to the Cryptographic Coprocessor Feature
- When an application uses any of the dynamic services that write to the PKDS
- When you use the Master Key Entry panels to enter a master key in the PCICC, PCIXCC or CEX2C
- When you create or delete a retained key on a PCICC, PCIXCC or CEX2C
- When you use the TKE workstation to communicate with the PCICC, PCIXCC or CEX2C
- To capture measurements of timing and configuration for the PCICC, PCIXCC, CEX2C, CEX2A or PCICA
- When ICSF issues IXCJOIN or IXCLEAVE to join or leave the ICSF sysplex group.
- When you use the Trusted Block Create callable service to create or activate a trusted block.
- When you use the PKCS #11 token management callable services to create or delete a token or object or to modify an attribute value of an object
- When the security administrator has indicated that duplicate key tokens must be identified
- When a callable service checks the key store policy

You can also use the Security Server (RACF) or an equivalent product to record attempts to use protected cryptographic keys or functions.

- **Recording and Formatting type 82 SMF Records in a Report**

  Sample jobs are available (in SYS1.SAMPLIB) to assist in the recording and formatting of type 82 SMF data:

  - **CSFSMFJ** - JCL that executes the code to dump and format SMF type 82 records for ICSF. Before executing the JOB step, you need to make modifications to the JCL (see the prologue in the sample for specific instructions). After the JCL has been modified, terminate SMF recording of the currently active dump dataset (by issuing I SMF) to allow for the unloading of SMF records. After SMF recording has been terminated, execute the JCL. The output goes into the held queue.
  - **CSFSMFR** - An EXEC that formats the SMF records into a readable report.

- **Recording and Formatting type 80 SMF Records in a Report**

  RACF provides support to log type 80 SMF records when a user attempts to access an ICSF service, utility, or key label when a profile is defined for the service, utility or label. See the *z/OS Security Server RACF Auditor's Guide* for guidance on how to activate this logging and to format the type 80 SMF records.

Chapter 3. Update of z/OS Cryptographic Services ICSF Administrator's Guide, SA22-7521-13, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF Administrator's Guide*, SA22-7521-13, for the PKA Key Management Extensions enhancements provided by this APAR. Refer to this source document if background information is needed.

Controlling Who Can Use Cryptographic Keys and Services

You can use z/OS Security Server RACF to control which applications can use specific keys and services. This can help you ensure that keys and services are used only by authorized users and jobs. You can also use RACF to audit the use of keys and services. In addition, you can establish a Key Store Policy that defines rules for the use of encrypted key tokens that are stored in a CKDS or PKDS. To use RACF to control access to keys and services, you create and maintain general resource profiles in the CSFKEYS class, the CSFSERV class, and the XFACILIT class.

- The CSFKEYS class controls access to cryptographic keys. You create profiles in this class (based on the label by which the key is defined in the CKDS or PKDS) to set access authority for the keys. For the exclusive purpose of requiring UPDATE instead of READ authority when transferring a secure AES or DES key from encryption under the master key to encryption under an RSA key, you can define profiles in the XCSFKEY class. Profiles in the XCSFKEY class are used in authorization checks only when the Symmetric Key Export service (CSNDSYX or CSNFSYX) is called. For all other callable services, the CSFKEYS class is used.
- The CSFSERV class controls access to ICSF services and ICSF TSO panel utilities.
- One or more resource profiles in the XFACILIT class define your Key Store Policy. A Key Store Policy consists of a number of controls that collectively determine how encrypted key tokens defined in a CKDS or PKDS can be accessed and used.

If you are not the RACF security administrator, you may need to ask assistance from that person. To use the auditing capabilities of RACF, you may need to ask for reports from a RACF auditor. Your installation's security plan should show who is responsible for maintaining these RACF profiles and auditing their use.

Steps for RACF-protecting keys and services

This procedure describes one approach for RACF-protecting keys and services:

1. Decide whether you will protect keys, services, or both. You can select which keys and services to protect.
2. You may want to organize the users who need access to ICSF keys and services into groups. To do this, obtain a list of the user IDs of users who need to use ICSF keys and services. If batch jobs or started tasks need to use ICSF, obtain the user IDs under which they will run.

   Group any of the user IDs together if they require access to the same keys and services. For example, you might want to set up groups as follows:
   - Users who work with MAC-related callable services
   - Users who work with PIN-related callable services
   - Users who work with a particular MAC, or a particular PIN
   - Users who call applications to dynamically update the CKDS
   - Users who perform functions available on the User Control Functions panel

Usually, all users of ICSF should have access to keys and services by virtue of their membership in one of these RACF groups, rather than specific users. This is because RACF maintains the access lists in in-storage profiles. When the in-storage profiles are created or changed, the in-storage profiles must be refreshed. (Merely changing them in the RACF data base is not sufficient. This is analogous to the in-storage CKDS maintained by ICSF.) To refresh the in-storage RACF profiles, the RACF security administrator must use the SETROPTS command:

```
SETROPTS RACLIST(CSFKEYS) REFRESH

SETROPTS RACLIST(CSFSERV) REFRESH
```

If you place *RACF groups* in the access lists of the RACF profiles, you can change a user's access to the protected services and keys by adding or removing the user from the groups. Ask your RACF security administrator to create the RACF groups.

You should also ask your RACF security administrator to connect you to these groups with CONNECT group authority. This permits you to connect and remove users from the groups.

For example, the security administrator could issue these commands:

```
ADDGROUP groupid

CONNECT  your-userid  GROUP(groupid)  AUTHORITY(CONNECT)
```

With CONNECT group authority, you are able to connect other users to the groups:

```
CONNECT  other-userid  GROUP(groupid)
```

With CONNECT group authority, you are also able to remove users from the groups:

```
REMOVE  other-userid  GROUP(groupid)
```

3. Ask your RACF security administrator for the authority to create and maintain profiles in the CSFKEYS and CSFSERV general resource classes. Usually, this is done by assigning a user the CLAUTH (class authority) attribute in the specified classes. For example, the security administrator can issue this command:

   ```
   ALTUSER  your-userid  CLAUTH(CSFKEYS CSFSERV)
   ```

4. If you want to use generic profiles that contain characters such as * and %, ask your RACF security administrator to activate generic profile checking in the CSFKEYS and CSFSERV classes:

   ```
   SETROPTS  GENERIC(CSFKEYS CSFSERV)
   ```

   Note: Using generic profiles has several advantages. Using generic profiles you can reduce the number of profiles that you need to maintain. You can also create a "top" generic profile that can be used to protect all keys and services that are not protected by a more specific profile.

5. Define profiles in the CSFKEYS and CSFSERV classes. For further instructions, see "Setting up profiles in the CSFKEYS general resource class" on page 9 and "Setting up profiles in the CSFSERV general resource class" on page 10.

6. Activate logging for CSFSERV using these commands:
   - ALTUSER userid UAUDIT - audits a userid
   - RALTER class-name profile-name AUDIT(*audit-attempt*[(*audit-access-level*)]) - used by the profile owner

     RALTER class-name profile-name GLOBALAUDIT(*access-attempt*[(*audit-access-level*)]) - used by a user with AUDITOR authority to set up profiles

- SETROPTS CLASSACT(CSFSERV) RACLIST(CSFSERV)
  SETR LOGOPTIONS(CSFSERV(....))

For more information on RDEFINE, RALTER, and SETR, see the *z/OS Security Server RACF Command Language Reference*.

7. Determine if you need to establish a Key Store Policy for a CKDS and/or a PKDS. A Key Store Policy is made up of a number of controls. Each Key Store Policy control is a resource in the XFACILIT class. The existence of a profile for a particular resource in the XFACILIT class enables that control. A Key Store Policy applies only to encrypted keys in a CKDS or PKDS. No key store policy controls are available or needed for a TKDS, because none of the keys in a TKDS are enciphered under a key-encrypting key. By enabling Key Store Policy controls, you can:
   - verify, when an application passes a callable service a key token instead of a key label, that the user has authority to the secure token. Profiles in the CSFKEYS class are named based on key labels (a discrete profile will exactly match the key label, while a generic profile will contain generic characters to match a number of key labels). Because the profiles are based on the key label, a SAF authorization check needs to know the key label of a CKDS or PKDS key record in order to perform the authorization check. A Key Store Policy control is available that will, if an application passes a callable service a key token instead of a key label, locate the associated key label(s) for the token so that a SAF authorization check can be carried out. By default, if ICSF cannot find an associated key label for the key token, the callable service will fail. However, another Key Store Policy control lets you use a default profile to specify access authority to tokens that are not stored in the CKDS or PKDS.
   - prevent applications from storing duplicate tokens in a CKDS or PKDS.
   - raise the level of access authority required to create, write to, or delete a key label.
   - raise the level of access authority required to export a token using the Symmetric Key Export callable service (CSNDSYX or CSNFSYX).
   - set additional restrictions on how keys covered by the profile can be used.

   You should familiarize yourself with the controls you can enable and decide on the Key Store Policy that is best for your installation. Refer to “Defining a key store policy” on page 14 for more information.

Setting up profiles in the CSFKEYS general resource class

To set up profiles in the CSFKEYS general resource class, take these steps:

1. Define appropriate profiles in the CSFKEYS class:

```
RDEFINE  CSFKEYS label  UACC(NONE)
         other-optional-operands
```

   where *label* is the label by which the key is defined in the CKDS or PKDS (this is not the transport key label). Note that if an application uses a token instead of a key label, no authorization checking is done on the use of the key.

   Notes:

   a. If you have ICSF/MVS Version 1 Release 1 profiles that specify *key-type.label*, you need to change them to specify only *label*.

   b. As with any RACF profile, if you want to change the profile later, use the RALTER command. To change the access list, use the PERMIT command as described in the next step.

c. If you have already started ICSF, you need to refresh the in-storage profiles. See Step 3.

d. You can specify other operands, such as auditing (AUDIT operand), on the RDEFINE or RALTER commands.

e. If the RACF security administrator has activated generic profile checking for the CSFKEYS class, you can create generic profiles using the generic characters * and %. This is the same as any RACF general resource class.

2. Give appropriate users (preferably groups) access to the profiles:

```
PERMIT  profile-name  CLASS(CSFKEYS)
        ID(groupid)  ACCESS(READ)
```

3. When the profiles are ready to be used, ask the RACF security administrator to activate the CSFKEYS class and refresh the in-storage RACF profiles:

```
SETROPTS  CLASSACT(CSFKEYS)

SETROPTS RACLIST(CSFKEYS) REFRESH
```

Setting up profiles in the CSFSERV general resource class

To set up profiles in the CSFSERV general resource class, take these steps:

1. Define appropriate profiles in the CSFSERV class:

```
RDEFINE  CSFSERV service-name  UACC(NONE)
         other-optional-operands
```

Where *service-name* is one of these:

CSFAEGN ANSI X9.17 EDC generate callable service

CSFAKEX ANSI X9.17 key export callable service

CSFAKIM ANSI X9.17 key import callable service

CSFAKTR ANSI X9.17 key translate callable service

CSFATKN ANSI X9.17 key transport key partial notarize callable service

CSFCKI Clear key import callable service

CSFCKM Multiple clear key import callable service

CSFCMK Change master key (TSO panel) utility

CSFCONV PCF CKSD to ICSF CKDS conversion utility

CSFCPA Clear PIN generate alternate callable service

CSFCPE Clear PIN encrypt callable service

CSFCSG VISA CVV service generate callable service

CSFCSV VISA CVV service verify callable service

CSFCTT Cipher text translate callable service

CSFCTT1 Cipher text translate (with ALET) callable service

CSFCVE Cryptographic variable encipher callable service

CSFCVT Control vector translate callable service

CSFDCO Decode callable service

CSFDEC Decipher callable service

CSFDEC1 Decipher (with ALET) callable service

CSFDKCS Clear master key entry (TSO panel) utility (PCICC and PCIXCC/CEX2C)

CSFDKEF	Clear master key entry (TSO panel) utility (CCF)
CSFDKG	Diversified key generate callable service
CSFDKM	Data key import callable service
CSFDKX	Data key export callable service
CSFDSG	Digital signature generate callable service
CSFDSV	Digital signature verify callable service
CSFECO	Encode callable service
CSFEDC	Compatibility service for the PCF CIPHER macro
CSFEMK	Compatibility service for the PCF EMK macro
CSFENC	Encipher callable service
CSFENC1	Encipher (with ALET) callable service
CSFEPG	Encrypted PIN generate callable service
CSFGKC	Compatibility service for the PCF GENKEY macro
CSFIQA	ICSF Query Algorithm callable service
CSFIQF	ICSF Query Facility callable service
CSFKEX	Key export callable service
CSFKGN	Key generate callable service
CSFKGUP	Key generation utility program
CSFKIM	Key import callable service
CSFKPI	Key part import callable service
CSFKRC	Key record create callable service
CSFKRD	Key record delete callable service
CSFKRR	Key record read callable service
CSFKRW	Key record write callable service
CSFKTR	Key translate callable service
CSFKYT	Key test callable service
CSFKYTX	Key test extended callable service
CSFMDG	MDC generate callable service
CSFMDG1	MDC generate (with ALET) callable service
CSFMGN	MAC generate callable service
CSFMGN1	MAC generate (with ALET) callable service
CSFMVR	MAC verify callable service
CSFMVR1	MAC verify (with ALET) callable service
CSFOWH	One-way hash generate callable service
CSFOWH1	One-way hash generate (with ALET) callable service
CSFPCAD	PCICC and PCIXCC/CEX2C management (TSO panel) utility (activate/deactivate)
CSFPCI	PCI interface callable service

CSFPCU	PIN Change/Unblock callable service
CSFPEX	Prohibit export callable service
CSFPEXX	Prohibit export extended callable service
CSFPGN	Clear PIN generate callable service
CSFPKD	PKA decrypt callable service
CSFPKDR	PKDS reencipher and PKDS activate (TSO panel) utilities
CSFPKE	PKA encrypt callable service
CSFPKG	PKA key generate callable service
CSFPKI	PKA key import callable service
CSFPKRC	PKDS record create callable service
CSFPKRD	PKDS record delete callable service
CSFPKRR	PKDS record read callable service
CSFPKRW	PKDS record write callable service
CSFPKSC	PKSC interface callable service
CSFPKTC	PKA key token change callable service
CSFPKX	PKA public key extract callable service
CSFPMCI	Pass phrase master key/KDS initialization (TSO panel) utility
CSFPTR	Encrypted PIN translate callable service
CSFPVR	Encrypted PIN verify callable service
CSFREFR	Refresh CKDS (TSO panel) utility
CSFRENC	Reencipher CKDS (TSO panel) utility
CSFRKD	Retained key delete callable service
CSFRKL	Retained key list callable service
CSFRKX	Remote key export callable service
CSFRNG	Random number generate callable service
CSFRNGL	Random number generate long callable service
CSFRSWS	Administrative control functions (TSO panel) utility (ENABLE)
CSFRTC	Compatibility service for the CUSP or PCF RETKEY macro
CSFSAD	Symmetric Algorithm Decipher
CSFSAD1	Symmetric Algorithm Decipher
CSFSAE	Symmetric Algorithm Encipher
CSFSAE1	Symmetric Algorithm Encipher
CSFSBC	SET block compose callable service
CSFSBD	SET block decompose callable service
CSFSKI	Secure key import callable service
CSFSKM	Multiple secure key import callable service
CSFSKY	Secure messaging for keys callable service

CSFSMK Set master key (TSO panel) utility

CSFSPN Secure messaging for PINs callable service

CSFSSWS Administrative control functions (TSO panel) utility (DISABLE)

CSFSYG Symmetric key generate callable service

CSFSYI Symmetric key import callable service

CSFSYX Symmetric key export callable service

CSFTBC Trusted block create callable service

CSFTCK Transform CDMF key callable service

CSFTRV Transaction validation callable service

CSFUDK User derived key callable service

CSF1GAV PKCS11 get attribute value callable service

CSF1SAV PKCS11 set attribute value callable service

CSF1TRC PKCS11 token record create callable service

CSF1TRD PKCS11 token record delete callable service

CSF1TRL PKCS11 token record list callable service

These service names are PKCS11 callable services that are not published in the ICSF application programmer's guide, but can be protected using the CSFSERV resource.

- CSF1GKP PKCS11 generate key pair callable service
- CSF1GSK PKCS11 generate secret key callable service
- CSF1PKS PKCS11 private key sign callable service
- CSF1PKV PKCS11 public key verify callable service
- CSF1SKD PKCS11 secret key decrypt callable service
- CSF1SKE PKCS11 secret key encrypt callable service
- CSF1UWK PKCS11 unwrap key callable service
- CSF1WPK PKCS11 wrap key callable service

Notes:

a. As with any RACF general resource profile, if you want to change the profile later, use the RALTER command. To change the access list, use the PERMIT command as described in the next step.

b. If you have already started ICSF, you need to refresh the in-storage profiles. See Step 3 on page 14.

c. You can specify other operands, such as auditing (AUDIT operand), on the RDEFINE or RALTER commands.

d. If the RACF security administrator has activated generic profile checking for the CSFSERV class, you can create generic profiles using the generic characters * and %. This is the same as with any RACF general resource class.

Example

If generic profile checking is in effect, these profiles enable you to specify which users and jobs can use the ciphertext translate callable services. No other services can be used by any job on the system.

```
RDEFINE  CSFSERV  CSFCTT   UACC(NONE)

RDEFINE  CSFSERV  CSFCTT1  UACC(NONE)

RDEFINE  CSFSERV  *        UACC(NONE)
```

2. Give appropriate users (preferably groups) access to the profiles:

```
PERMIT  profile-name  CLASS(CSFSERV)
        ID(groupid)  ACCESS(READ)
```

3. When the profiles are ready to be used, ask the RACF security administrator to activate the CSFSERV class and refresh the in-storage RACF profiles:

```
SETROPTS  CLASSACT(CSFSERV)

SETROPTS RACLIST(CSFSERV) REFRESH
```

Defining a key store policy

A Key Store Policy defines rules for how encrypted key tokens stored in a CKDS or PKDS can be accessed and used. A Key Store Policy is collectively defined by a number of separate controls that each specify a particular rule. Most of the Key Store Policy controls work in conjunction with profiles in the CSFKEYS class, and enable you to:

- Specify how ICSF should respond when a key token is passed to a callable service instead of a key label (which is needed to perform a SAF authorization check).
- Determine if applications should be prevented from creating a new key record (with a new key label) for a token that is already stored in the CKDS or PKDS (in a key record with a different key label).
- Specify if READ access authority is sufficient to create, write to, or delete a key label, or if a higher level of access authority should be required for these actions.
- Specify if READ access authority to an AES or DES key is sufficient to export the key (move it from encryption under a master key to encryption under an RSA key), or if UPDATE authority should be required for this action.
- Place restrictions on how keys can be used. You can:
  - restrict a particular AES or DES key from being exported, or allow it to be exported only by certain RSA keys (or only by RSA keys bound to identities in certain key certificates).
  - restrict certain RSA keys from being used in secure export and import operations, or from being used in handshake operations.

Each Key Store Policy control is a resource in the XFACILIT class, and can be enabled by creating a profile for the resource using the RDEFINE command. Similarly, you can disable a control by deleting its profile using the RDELETE command. ICSF detects changes to the XFACILIT class, so the class does not need to be active or RACLISTed.

Certain controls, when enabled, will *activate* Key Store Policy for either the CKDS or PKDS. When Key Store Policy is *activated*, ICSF will identify the key label(s) associated with each key token in the key store. This information is needed, for example, in order to carry out SAF authorization checks against RACF profiles (which are based on key labels) when a key token is passed to a callable service, or to ensure an application doesn't store a duplicate token (a token that is already stored, but associated with a different key label) in the key store. In addition to the controls that activate Key Store Policy, other controls that do not themselves activate Key Store Policy may still require, or to a lesser degree rely upon, an active Key Store Policy and its key token/label associations. The following table outlines the Key Store Policy controls that are available. This table also highlights

the controls that activate Key Store Policy for a CKDS or PKDS, as well as the dependencies the other controls have on Key Store Policy being active. Be aware that Key Store Policy is activated separately for a CKDS and a PKDS.

Table 1. Key Store Policy controls

The following Key Store Policy controls:	Consist of the following XFACILIT class resources:	Description:
Key Token Authorization Checking controls Verifies, when an application passes a callable service a key token instead of a key label, that the user has authority to the key token in the CKDS or PKDS. It does this by identifying the key label associated with the passed token.	CSF.CKDS.TOKEN.CHECK.LABEL.WARN	**Activates Key Store Policy for CKDS.** Enables Key Token Authorization Checking for the CKDS in warning mode. In this mode, a failing authorization check will result in a warning, but the operation will be allowed to continue.
	CSF.CKDS.TOKEN.CHECK.LABEL.FAIL	**Activates Key Store Policy for CKDS.** Enables Key Token Authorization Checking for the CKDS in fail mode. In this mode, ICSF does not allow the operation to continue when the authorization check fails. The service returns with an error.
	CSF.PKDS.TOKEN.CHECK.LABEL.WARN	**Activates Key Store Policy for PKDS.** Enables Key Token Authorization Checking for the PKDS in warning mode. In this mode, a failing authorization check will result in a warning, but the operation will be allowed to continue.
	CSF.PKDS.TOKEN.CHECK.LABEL.FAIL	**Activates Key Store Policy for PKDS.** Enables Key Token Authorization Checking for the PKDS in fail mode. In this mode, ICSF does not allow the operation to continue when the authorization check fails. The service returns with an error.
Default Key Label Checking controls Specifies that ICSF should use a default profile to determine application access to tokens that are not stored in the CKDS or PKDS. Can be enabled only if the Key Token Authorization Checking control for the appropriate key store is also enabled.	CSF.CKDS.TOKEN.CHECK.DEFAULT.LABEL	**Requires an active Key Store Policy for CKDS.** Specifically, this control can be enabled only if the CSF.CKDS.TOKEN.CHECK.LABEL.WARN or CSF.CKDS.TOKEN.CHECK.LABEL.FAIL control is also enabled. Specifies that ICSF should use the default profile CSF-CKDS-DEFAULT in the CSFKEYS class to determine user access to tokens that are not stored in the CKDS.
	CSF.PKDS.TOKEN.CHECK.DEFAULT.LABEL	**Requires an active Key Store Policy for PKDS.** Specifically, this control can be enabled only if the CSF.PKDS.TOKEN.CHECK.LABEL.WARN or CSF.PKDS.TOKEN.CHECK.LABEL.FAIL control is also enabled. Specifies that ICSF should use the default profile CSF-PKDS-DEFAULT in the CSFKEYS class to determine user access to tokens that are not stored in the PKDS.
Duplicate Key Token Checking controls Prevents applications from storing duplicate tokens in the CKDS or PKDS.	CSF.CKDS.TOKEN.NODUPLICATES	**Activates Key Store Policy for CKDS.** Enables Duplicate Key Token Checking for the CKDS. ICSF will prevent an application from creating a new key record (with a new key label) for a token that is already stored in the CKDS.
	CSF.PKDS.TOKEN.NODUPLICATES	**Activates Key Store Policy for PKDS.** Enables Duplicate Key Token Checking for the PKDS. ICSF will prevent an application from creating a new key record (with a new key label) for a token that is already stored in the PKDS.

Table 1. Key Store Policy controls (continued)

The following Key Store Policy controls:	Consist of the following XFACILIT class resources:	Description:
Granular Key Label Access controls Increases the level of access authority required to create, write to, or delete a key label.	CSF.CSFKEYS.AUTHORITY.LEVELS.WARN	Enables Granular Key Label Access in warning mode. In this mode, a warning will be issued if the user does not have UPDATE authority (if creating a label), or CONTROL authority (if writing to or deleting a label). As long as the user has READ authority, however, ICSF will allow the operation to continue. **Does not require an active Key Store Policy for CKDS or PKDS. However, if a key token is passed to a callable service instead of a key label, ICSF will, in order to initiate a SAF authorization check, rely on an active Key Store Policy for the appropriate key store.**
	CSF.CSFKEYS.AUTHORITY.LEVELS.FAIL	Enables Granular Key Label Access in fail mode. In this mode, ICSF will not allow a key label to be modified if the user does not have UPDATE authority (if creating a label), or CONTROL authority (if writing to or deleting a label). The service returns with an error. **Does not require an active Key Store Policy for CKDS or PKDS. However, if a key token is passed to a callable service instead of a key label, ICSF will, in order to initiate a SAF authorization check, rely on an active Key Store Policy for the appropriate key store.**
Symmetric Key Label Export controls Specifies that profiles in the XCSFKEY class (instead of profiles in the CSFKEYS class) should be used to determine access to AES or DES keys that an application is attempting to export using the Symmetric Key Export (CSNDSYX or CSNFSYX) callable service. This allows you to control access to AES and DES keys for the purpose of key export separately from the access allowed to the keys for other purposes.	CSF.XCSFKEY.ENABLE.AES	Enables Symmetric Key Label Export for AES keys. Specifies that profiles in the XCSFKEY class should determine access to an AES key when an application is attempting to export it using the Symmetric Key Export (CSNDSYX or CSNFSYX) callable service. **Does not require an active Key Store Policy for CKDS or PKDS. However, if a key token is passed to the callable service instead of a key label, ICSF will, in order to initiate the SAF authorization check, rely on an active Key Store Policy for CKDS.**
	CSF.XCSFKEY.ENABLE.DES	Enables Symmetric Key Label Export for DES keys. Specifies that profiles in the XCSFKEY class should determine access to a DES key when an application is attempting to export it using the Symmetric Key Export (CSNDSYX or CSNFSYX) callable service. **Does not require an active Key Store Policy for CKDS or PKDS. However, if a key token is passed to the callable service instead of a key label, ICSF will, in order to initiate the SAF authorization check, rely on an active Key Store Policy for CKDS.**

Table 1. Key Store Policy controls (continued)

The following Key Store Policy controls:	Consist of the following XFACILIT class resources:	Description:
PKA Key Management Extensions control Specifies that the ICSF segment of profiles in the CSFKEYS class (and the XCSFKEY class when a Symmetric Key Label Export control is enabled) will be checked to determine additional restrictions on how keys covered by the profile can be used.	CSF.PKAEXTNS.ENABLE.WARNONLY	**Requires an active Key Store Policy for CKDS and PKDS.** Enables PKA Key Management Extensions in warning mode. The ICSF segment of CSFKEYS or XCSFKEY profiles will be checked to: • determine if a symmetric key can be exported, and, if so, which asymmetric keys can be used in the operation to re-encrypt the symmetric key. • determine if an asymmetric key can be used in secure export and import operations, or in handshake operations. However, because this is warning mode, ICSF will allow the operation to continue even if the ICSF segment indicates that the operation is not allowed.
	CSF.PKAEXTNS.ENABLE	**Requires an active Key Store Policy for CKDS and PKDS.** Enables PKA Key Management Extensions in fail mode. The ICSF segment of CSFKEYS or XCSFKEY profiles will be checked to: • determine if a symmetric key can be exported, and, if so, which asymmetric keys can be used in the operation to re-encrypt the symmetric key. • determine if an asymmetric key can be used in secure export and import operations, or in handshake operations. If the ICSF segment indicates that the operation is not allowed, the service returns with an error.

For more information on the:

- Key Token Authorization Checking controls, refer to “Enabling access authority checking for key tokens”
- Default Key Label Checking controls, refer to “Determining access to tokens not stored in the CKDS or PKDS” on page 19
- Duplicate Key Token Checking controls, refer to “Enabling duplicate key label checking” on page 20
- Granular Key Label Access controls, refer to “Increasing the level of authority needed to modify key labels” on page 21
- Symmetric Key Label Export controls, refer to “Increasing the level of authority required to export symmetric keys” on page 23
- PKA Key Management Extension control, refer to “Controlling how cryptographic keys can be used” on page 25

Enabling access authority checking for key tokens

Profiles in the CSFKEYS class determine access authority to cryptographic keys. However, CSFKEYS profiles protect keys by their key label (discrete or generic CSFKEYS profiles are named to match one or more key labels), and ICSF callable services accept either a key label or key token. By default, if an application passes a callable service a key token instead of a key label, no authorization checking is done on the use of the key. By enabling Key Token Authorization Checking controls, you can have ICSF identify a key token's associated key label so that a SAF authorization check can be performed. This lets you implement a consistent security policy for keys regardless of how they are identified (by key label or key token) to callable services.

Separate Key Token Authorization Checking controls are provided for activating the checking for either a CKDS or a PKDS in either warning or fail mode. In warning mode, authorization checking is performed, but an application will not be prevented from using a token even when the user lacks the necessary authority. Instead, ICSF will merely log an SMF type 82 subtype 25 record in the SMF dataset. Warning mode allows you to identify users who will need access permission to a key prior to moving to a stricter implementation of the Key Token Authorization Checking policy.

This stricter implementation of the policy is called fail mode. In fail mode, an application will be denied access to a token when the user does not have authority to access it. The operation will be unsuccessful, and a return code 8, reason code BF7 (3063) will be returned to the calling application. As with warning mode, ICSF will log an SMF type 82 subtype 25 record in the SMF dataset. In addition, RACF will log an SMF type 80 record (with event code qualifier of ACCESS). The resource name in the SMF type 80 record will be the first label associated with the key token that failed the check.

Because the same token could be associated with multiple key records in the key store, when an application passes an encrypted key token to an ICSF callable service, ICSF locates all the labels associated with the passed token. If the user has permission to any of the key labels, then the application is granted authority to use the token. Because access authority to any label associated with a token will give a user access to the token, you may want to ensure that the key store does not contain multiple key records for the same key token. ICSF provides a utility program, CSFDUTIL, that generates a report of all duplicate keys for either a CKDS or PKDS. To prevent duplicate keys from being added to a key store, you can enable the Default Key Label Checking control for either the CKDS or PKDS as described in “Enabling duplicate key label checking” on page 20.

If ICSF can not find an associated key label for the passed token in the key store, no authorization checking will be performed on the use of the key unless the Default Key Label Checking control is enabled for the key store. If the Default Key Label Checking control is enabled (as described in “Determining access to tokens not stored in the CKDS or PKDS” on page 19), a default profile will determine user access when ICSF cannot identify an associated label for the passed token.

The following table shows the controls for enabling Key Token Authorization Checking for the CKDS and PKDS in either warning or fail mode. To enable one of the Key Token Authorization Checking controls, create the appropriate profile in the XFACILIT class. The XFACILIT class does not need to be active or RACLISTed in order for the control to be enabled.

Table 2. Key Store Policy controls: The Key Token Authorization Checking controls

The existence of this resource profile in the XFACILIT class:	**Does this:**
CSF.CKDS.TOKEN.CHECK.LABEL.WARN	**Activates Key Store Policy for CKDS.** Enables Key Token Authorization Checking for the CKDS in warning mode. In this mode, a failing authorization check will result in a warning, but the operation will be allowed to continue.
CSF.CKDS.TOKEN.CHECK.LABEL.FAIL	**Activates Key Store Policy for CKDS.** Enables Key Token Authorization Checking for the CKDS in fail mode. In this mode, ICSF does not allow the operation to continue when the authorization check fails. The service returns with an error.
CSF.PKDS.TOKEN.CHECK.LABEL.WARN	**Activates Key Store Policy for PKDS.** Enables Key Token Authorization Checking for the PKDS in warning mode. In this mode, a failing authorization check will result in a warning, but the operation will be allowed to continue.

Table 2. Key Store Policy controls: The Key Token Authorization Checking controls (continued)

The existence of this resource profile in the XFACILIT class:	Does this:
CSF.PKDS.TOKEN.CHECK.LABEL.FAIL	**Activates Key Store Policy for PKDS.** Enables Key Token Authorization Checking for the PKDS in fail mode. In this mode, ICSF does not allow the operation to continue when the authorization check fails. The service returns with an error.

For example, say you want to enable Key Token Authorization Checking for both a CKDS and a PKDS. You're not certain all the users currently accessing key tokens in these key stores will have the necessary access authority, and do not want to disrupt current work patterns at your installation. For this reason, you decide to allow a warning period during which you can identify users who will need permission to access certain key tokens. The following command will enable Key Token Authorization Checking for the CKDS and the PKDS in warning mode.

```
RDEFINE XFACILIT CSF.CKDS.TOKEN.CHECK.LABEL.WARN
RDEFINE XFACILIT CSF.PKDS.TOKEN.CHECK.LABEL.WARN
```

During the warning period, you can, by examining the SMF type 82 subtype 25 records logged in the SMF dataset, identify the users who need permission to access keys. You can then create or modify the necessary profiles in the CSFKEYS class. When you are ready to move to a stricter implementation of this policy, you enable the controls for fail mode and disable the ones for warning mode.

```
RDEFINE XFACILIT CSF.CKDS.TOKEN.CHECK.LABEL.FAIL
RDEFINE XFACILIT CSF.PKDS.TOKEN.CHECK.LABEL.FAIL
RDELETE XFACILIT CSF.CKDS.TOKEN.CHECK.LABEL.WARN
RDELETE XFACILIT CSF.PKDS.TOKEN.CHECK.LABEL.WARN
```

If you accidentally enable the Key Token Authorization Checking controls for both warning and fail mode, the control for fail mode will take precedence.

Determining access to tokens not stored in the CKDS or PKDS: When the Key Token Authorization Checking control for a key store has been enabled, and a token is passed to a callable service, ICSF will find the key label(s) associated with the passed token so that a SAF authority check can be performed. If, however, the token passed to the callable service is not in the key store, there will be no associated key label to find. By default, no authorization checking is performed on the use of the key, and the operation is allowed. If you enable the Default Key Label Checking control for the CKDS or PKDS, however, ICSF will use a default profile to determine user access to tokens that are not in the key store.

Separate controls are provided for enabling Default Key Label Checking for a CKDS or a PKDS, The Default Key Label Checking control will be enabled only if the Key Token Authorization Checking control for the appropriate key store is also enabled. Refer to “Enabling access authority checking for key tokens” on page 17 for more information. To enable one the Default Key Label Checking controls, create the appropriate profile in the XFACILIT class. The XFACILIT class does not need to be active or RACLISTed in order for the control to be enabled.

Table 3. Key Store Policy controls: The Default Key Label Checking controls

The existence of this resource profile in the XFACILIT class:	Does this:
CSF.CKDS.TOKEN.CHECK.DEFAULT.LABEL	Specifies that ICSF should use the default profile CSF-CKDS-DEFAULT in the CSFKEYS class to determine user access to tokens that are not stored in the CKDS. This control is enabled only if the CSF.CKDS.TOKEN.CHECK.LABEL.WARN or CSF.CKDS.TOKEN.CHECK.LABEL.FAIL control is also enabled.

Table 3. Key Store Policy controls: The Default Key Label Checking controls (continued)

The existence of this resource profile in the XFACILIT class:	Does this:
CSF.PKDS.TOKEN.CHECK.DEFAULT.LABEL	Specifies that ICSF should use the default profile CSF-PKDS-DEFAULT in the CSFKEYS class to determine user access to tokens that are not stored in the PKDS. This control is enabled only if the CSF.PKDS.TOKEN.CHECK.LABEL.WARN or CSF.PKDS.TOKEN.CHECK.LABEL.FAIL control is also enabled.

For example, to enable the Default Key Label Checking control for a CKDS, you would:

1. Create the default profile CSF-CKDS-DEFAULT in the CSFKEYS class.

   ```
   RDEFINE CSFKEYS CSF-CKDS-DEFAULT UACC(NONE)
   ```

2. By defining the universal access authority (UACC) as NONE in the preceding step, the use of key tokens that do not reside in the key store has been prohibited. If necessary, however, you can give appropriate users (preferably groups) access in the CSF-CKDS-DEFAULT profile and refresh the CSFKEYS class in storage:

   ```
   PERMIT CSF-CKDS-DEFAULT CLASS(CSFKEYS) ID(group-id) ACCESS(READ)
   SETROPTS RACLIST(CSFKEYS) REFRESH
   ```

3. Create a profile for the CSF.CKDS.TOKEN.CHECK.DEFAULT.LABEL resource in the XFACILIT class.

   ```
   RDEFINE XFACILIT CSF.CKDS.TOKEN.CHECK.DEFAULT.LABEL
   ```

Enabling duplicate key label checking

A key token could be stored in a key store within multiple key records, and so could be associated with multiple key labels. When the Key Token Authorization Checking control is enabled for the key store, duplicate tokens can cause problems because all labels that are associated with a key token passed to an ICSF callable service will be used to determine user access to that token. Although you may deliberately restrict access to a token by one of the labels associated with it, a user might still have access to the token through another label. You can enable the Duplicate Key Token Checking control for the CKDS or PKDS to prevent applications from storing duplicate tokens in the key store. When enabled, ICSF services that update the key store will check for duplicate tokens. ICSF will not allow a key token to be written to the key store if it matches a token that is already stored. The Duplicate Key Token Checking controls do not rely on SAF authorization checks against CSFKEYS class profiles. Instead, the callable services that update the key store will verify that a duplicate token does not already exist within the key store.

Note: Enabling the Duplicate Key Token Checking control for the CKDS or PKDS ensures only that no duplicate keys are added to the key store. To identify any duplicate key tokens that may already exist in a CKDS or PKDS, use the CSFDUTIL utility program. The CSFDUTIL utility program generates a report of all duplicate keys in either a CKDS or a PKDS.

Separate controls are provided for enabling Duplicate Key Token Checking for a CKDS or a PKDS. To enable either of the Duplicate Key Token Checking controls, create the appropriate profile in the XFACILIT class. The XFACILIT class does not need to be active or RACLISTed in order for the control to be enabled.

Table 4. Key Store Policy controls: The Duplicate Key Token Checking controls

The existence of this resource profile in the XFACILIT class:	Does this:
CSF.CKDS.TOKEN.NODUPLICATES	**Activates Key Store Policy for CKDS.** Enables Duplicate Key Token Checking for the CKDS. ICSF will prevent an application from creating a new key record (with a new key label) for a token that is already stored in the CKDS.
CSF.PKDS.TOKEN.NODUPLICATES	**Activates Key Store Policy for PKDS.** Enables Duplicate Key Token Checking for the PKDS. ICSF will prevent an application from creating a new key record (with a new key label) for a token that is already stored in the PKDS.

For example, to ensure that duplicate tokens are not stored in either the CKDS or PKDS, you would enter the following commands:

```
RDEFINE XFACILIT CSF.CKDS.TOKEN.NODUPLICATES
RDEFINE XFACILIT CSF.PKDS.TOKEN.NODUPLICATES
```

Increasing the level of authority needed to modify key labels

A number of ICSF callable services enable an application to create, write to, or delete a key label. By default, the user needs only READ authority to read from, create, write to, or delete a label. In some cases, however, you might want to require a higher level of authority for modifying a label than is required to merely read a label. By enabling the Granular Key Label Access control, you increase the level of access authority required to create, write to, or delete a label, while still requiring only READ authority for cryptographic functions. This way, you can give a user permission to access a key for encryption or decryption operations, while preventing that same user from changing or deleting the key record.

The following table outlines the increased access authority required when the Granular Key Label Access control is enabled.

Table 5. Increased access authority required to modify key labels when Granular Key Label Access control is enabled

To do this:	The level of access authority required is increased from READ to:	This impacts the following callable services:
Create a label	UPDATE	Key Record Create (CSNBKRC) PKDS Record Create (CSNDKRC and CSNFKRC)
Write to a label	CONTROL	Key Part Import (CSNBKPI) Key Record Write (CSNBKRW) PKDS Record Create (CSNDKRC and CSNFKRC) PKDS Record Write (CSNDKRW) PKA Key Generate (CSNDPKG and CSNFPKG) PKA Key Import (CSNDPKI and CSNFPKI) Trusted Block Create (CSNDTBC)
Delete a label	CONTROL	Key Record Delete (CSNBKRD) PKDS Record Delete (CSNDKRD and CSNFKRD) Retained Key Delete (CSNDRKD and CSNFRKD)

You can enable the Granular Key Label Access control in warning or fail mode. In warning mode, the user's access authority will be checked, but only READ authority will be required. However, if a user does not have UPDATE authority when creating a label, or CONTROL authority when writing to or deleting a label, a warning will be

issued and the access will be logged. Warning mode allows you to identify any users who will need to be granted increased access authority prior to moving to a stricter implementation of the policy. The stricter implementation of the policy is called fail mode. In fail mode, users who lack the increased access authority required will not be able to modify key labels. The operation will be unsuccessful, and a return code of 8 (reason code 16004) will be returned to the calling application.

It is recommended that you activate Key Store Policy for both the CKDS and the PKDS before enabling the Granular Key Label Access control. If Key Store Policy is not activated and the Granular Key Label Access control is enabled, the increased access authority checks will work only when the application passes a callable service a key label. However, if the application were to pass the callable service a key token instead of a key label, then no authorization checking will be performed. When a token is passed, ICSF will, in order to initiate a SAF authorization check, rely on an active Key Store Policy for the appropriate key store.

Enabling any one of the following controls will activate Key Store Policy for a CKDS:

- CSF.CKDS.TOKEN.CHECK.LABEL.WARN
- CSF.CKDS.TOKEN.CHECK.LABEL.FAIL
- CSF.CKDS.TOKEN.NODUPLICATES

Enabling any one of the following controls will activate Key Store Policy for a PKDS:

- CSF.PKDS.TOKEN.CHECK.LABEL.WARN
- CSF.PKDS.TOKEN.CHECK.LABEL.FAIL
- CSF.PKDS.TOKEN.NODUPLICATES

The following table shows the controls for enabling Granular Key Label Access in warning or fail mode. To enable one of the controls, create the appropriate profile in the XFACILIT class. The XFACILIT class does not need to be active or RACLISTed in order for the control to be enabled.

Table 6. Key Store Policy controls: The Granular Key Label Access controls

The existence of this resource profile in the XFACILIT class:	**Does this:**
CSF.CSFKEYS.AUTHORITY.LEVELS.WARN	Enables Granular Key Label Access in warning mode. In this mode, a warning will be issued if the user does not have UPDATE authority if creating a label, or CONTROL authority if writing to or deleting a label. As long as the user has READ authority, however, ICSF will allow the operation to continue.
CSF.CSFKEYS.AUTHORITY.LEVELS.FAIL	Enables Granular Key Label Access in fail mode. In this mode, ICSF will not allow a key label to be modified if the user does not have UPDATE authority if creating a label, or CONTROL authority if writing to or deleting a label. The service returns with an error.

For example, you want to require UPDATE authority to create a label, and CONTROL authority to write to or delete a label. You're not certain all the users currently modifying key labels will have the necessary access authority, and do not want to disrupt current work patterns at your installation. For this reason, you decide to allow a warning period during which you can identify which users will need to be granted increased authority. To do this, you would:

1. Enable the Granular Key Label Access control in warning mode.

   ```
   RDEFINE XFACILIT CSF.CSFKEYS.AUTHORITY.LEVELS.WARN
   ```

2. Because you have enabled the control in warning mode, a failing access check will still allow a user to modify the key record (as long as the user has READ authority), but will issue a warning and log the access. Using this information, you can update the appropriate profiles in the CSFKEYS class to grant increased access authority to the appropriate users. For example, if user RITA needs to be able to generate RSA key tokens (by way of the CSNDKRC and CSNDPKG callable services), she will need CONTROL access to the label:

   ```
   PERMIT RITA.RSA.TEST.* CLASS(CSFKEYS) ID(RITA) ACCESS(CONTROL)
   ```

3. When you are ready to move to a stricter implementation of the policy, you would enable the control for fail mode and disable the one for warning mode.

   ```
   RDEFINE XFACILIT CSF.CKDS.TOKEN.CHECK.LABEL.FAIL
   RDELETE XFACILIT CSF.CKDS.TOKEN.CHECK.LABEL.WARN
   ```

If you accidentally enable the Granular Key Label Access controls for both warning and fail mode, the control for fail mode will take precedence.

Increasing the level of authority required to export symmetric keys

Using the Symmetric Key Export (CSNDSYX or CSNFSYX) callable service, an application can transfer a symmetric (AES or DES) key from encryption under a master key to encryption under an application-supplied RSA public key. This callable service is used because a secure key (which is encrypted under a master key in the ICSF environment) might need to be shared with a partner, and to transfer it to that partner securely, it will need to be encrypted under an RSA key provided by the partner. The partner will then be able to decrypt it using a corresponding private key.

The export operation performed by the Symmetric Key Export callable service does not fit into a traditional access control hierarchy. Due to the nature of the export operation, you might want to restrict users from accessing a symmetric key for the purpose of exporting it, while still allowing users to access the key for other purposes. By enabling the Symmetric Key Label Export control for AES or DES keys, and creating profiles in the XCSFKEY resource class, you can increase the level of access authority needed to export AES or DES keys without increasing the level of authority needed to access the keys for other operations.

By default, the CSFKEYS class determines access authority to cryptographic keys passed to callable services (including the CSNDSYX/CSNFSYX callable service). When the Symmetric Key Label Export control for AES or DES keys is **not** enabled and the CSNDSYX or CSNFSYX service is called, a user needs only READ authority for the key (as specified in a CSFKEYS class profile). If, however, the Symmetric Key Label Export control for AES or DES keys **is** enabled and the CSNDSYX or CSNFSYX service is called, then a user needs UPDATE authority for the key (as specified in an XCSFKEY class profile). The Symmetric Key Label Export controls affect only the CSNDSYX/CSNFSYX callable service; for all other callable services, access to cryptographic keys is checked against profiles in the CSFKEYS class. What's more, the Symmetric Key Label Export controls affect access only to the symmetric key the application is attempting to export, and do not affect access to the RSA key that is being used to re-encrypt the symmetric key. Access authority to the AES or DES key will be checked against XCSFKEY class profiles, while access to the RSA key will still be checked against CSFKEYS class profiles.

It is recommended that you activate Key Store Policy for the CKDS before enabling the Symmetric Key Label Export control for AES or DES keys. If Key Store Policy is not activated for the CKDS and the Symmetric Key Label Export control for AES or

DES keys is enabled, the access authority check for the symmetric key will be performed only when it is identified to the CSNDSYX or CSNFSYX callable service by its key label. If the application were to pass the callable service a key token instead of a key label, then no authorization checking will be performed. When a token is passed, ICSF will, in order to initiate a SAF authorization check, rely on an active Key Store Policy for CKDS. Enabling any one of the following controls will activate Key Store Policy for a CKDS:

- CSF.CKDS.TOKEN.CHECK.LABEL.WARN
- CSF.CKDS.TOKEN.CHECK.LABEL.FAIL
- CSF.CKDS.TOKEN.NODUPLICATES

The following table shows the controls for enabling Symmetric Key Label Export for AES or DES keys. To enable the controls, create the appropriate profile in the XFACILIT class. The XFACILIT class does not need to be active or RACLISTed in order for the control to be enabled. There are separate Symmetric Key Label Export controls for AES and DES keys, so you can require UPDATE authority (which will be checked against XCSFKEY profiles) for export of one type of key, while still requiring only READ authority (which will still be checked against CSFKEY profiles) for export of the other type of key. There are no Symmetric Key Label Export controls that enable the policy in a warning mode. However, you can use the WARNING operand on XCSFKEY profiles to achieve the same results.

Table 7. Key Store Policy controls: The Symmetric Key Label Export controls

The existence of this resource profile in the XFACILIT class:	**Does this:**
CSF.XCSFKEY.ENABLE.AES	Enables Symmetric Key Label Export for AES keys. Specifies that profiles in the XCSFKEY class should determine access to an AES key when an application is attempting to export it using the Symmetric Key Export (CSNDSYX or CSNFSYX) callable service.
CSF.XCSFKEY.ENABLE.DES	Enables Symmetric Key Label Export for DES keys. Specifies that profiles in the XCSFKEY class should determine access to a DES key when an application is attempting to export it using the Symmetric Key Export (CSNDSYX or CSNFSYX) callable service.

For example, you want to require UPDATE authority to export any symmetric key (AES or DES) using the Symmetric Key Export callable service. You're not certain all the users currently exporting symmetric keys will have the necessary access authority, and do not want to disrupt current work patterns at your installation. For this reason, you decide to allow a warning period during which you can identify which users will need to be granted increased authority. To do this, you would:

1. Create profiles in the XCSFKEY class to cover the symmetric keys. In this example, your installation has a consistent naming policy for AES and DES key labels, so the following two generic profiles will cover all symmetric keys. The WARNING operand is specified to initiate the warning period.

   ```
   RDEFINE XCSFKEY AES* UACC(NONE) WARNING
   RDEFINE XCSFKEY DES* UACC(NONE) WARNING
   ```

   The XCSFKEY class will need to be activated and placed in common storage:

   ```
   SETROPTS CLASSACT(XCSFKEY)
   SETROPTS RACLIST(XCSFKEY)
   ```

2. Enable the Symmetric Key Label Export control for AES and DES. In this example, we enable both controls so that UPDATE authority is required when exporting any symmetric key.

```
RDEFINE CSF.XCSFKEY.ENABLE.AES
RDEFINE CSF.XCSFKEY.ENABLE.DES
```

3. Because the WARNING operand was specified on the generic profiles AES* and DES*, any failing access check will still allow access to the symmetric key, but will issue a warning message and log the access. Using this information, you can grant UPDATE access to users or groups as needed. Since the generic profiles in our example cover all AES and all DES keys, you may need to create other generic profiles or discrete profiles to limit access for certain users. Here, user BOBADMIN is given UPDATE access to all symmetric keys, while user GWEN is given UPDATE access to the key labeled DES.BURDA.MEDINC.

   ```
   PERMIT AES* CLASS(XCSFKEY) ID(BOBADMIN) ACCESS(UPDATE)
   PERMIT DES* CLASS(XCSFKEY) ID(BOBADMIN) ACCESS(UPDATE)
   RDEFINE XCSFKEY DES.BURDA.MEDINC UACC(NONE)
   PERMIT DES.BURDA.MEDINC CLASS(XCSFKEY) ID(GWEN) ACCESS(UPDATE)
   ```

   The XCSFKEY class will need to be refreshed in common storage:

   ```
   SETROPTS RACLIST(XCSFKEY) REFRESH
   ```

4. When you are ready to move to a stricter implementation of the policy, you can end the warning period. To do this, update the necessary profiles in the XCSFKEY class using the RALTER command with its NOWARNING operand.

   ```
   RALTER XCSFKEY AES* UACC(NONE) NOWARNING
   RALTER XCSFKEY DES* UACC(NONE) NOWARNING
   ```

   The XCSFKEY class will need to be refreshed in common storage:

   ```
   SETROPTS RACLIST(XCSFKEY) REFRESH
   ```

Controlling how cryptographic keys can be used

In addition to using profiles in the CSFKEYS class (and, when Symmetric Key Label Export is enabled, the XCSFKEY class) to identify which users have permission to certain cryptographic keys, you can also enable the PKA Key Management Extensions control so that CSFKEYS and XCSFKEY profiles can place restrictions on how keys are used. For example, you can:

- restrict an asymmetric key from being used in secure export and import operations.
- restrict an asymmetric key from being used in handshake operations.
- Restrict a symmetric key from being exported (transferred from encryption under a master key to encryption under an application-supplied RSA public key). Alternatively, you can allow the symmetric key to be exported, but only by certain public keys (as indicated by a list of key labels), or only by public keys bound to certain identities (as indicated by a list of certificates in either a PKCS #11 token, or a SAF key ring).

Setting restrictions such as these can help ensure that keys are used only for intended purposes, regardless of who has access to the keys. For example, if you have an RSA key pair intended only for generating and verifying digital signatures, you can set a restriction to ensure that the public key of this key pair is never used to export a symmetric key.

You place restrictions on cryptographic keys using the ICSF segment of the CSFKEYS or XCSFKEY class profiles that cover the keys. After you have modified the profiles with the restrictions you want to place on the keys, you can enable the PKA Key Management Extensions control by creating a CSF.PKAEXTNS.ENABLE profile in class XFACILIT. You can also enable PKA Key Management Extensions in warning mode by creating a CSF.PKAEXTNS.ENABLE.WARNONLY profile in class XFACILIT. In order to enable PKA Key Management Extensions, Key Store Policy

must be active for both the CKDS and the PKDS. For more information, refer to "Enabling PKA Key Management Extensions" on page 32.

Restricting asymmetric keys from being used in secure import and export operations: Using the ASYMUSAGE field in the ICSF segment of CSFKEYS profiles enables you to restrict asymmetric keys covered by the profile from being used in secure import and export operations. In secure export operations, a symmetric key (AES or DES) is moved from encryption under a master key to encryption under an asymmetric key (RSA public key). In a secure import operation, the private key of an RSA key pair is used to move a symmetric key from encryption under the RSA public key to encryption under a master key. The following callable services all identify an asymmetric key (either the public or private key of an RSA key pair) to encrypt or decrypt a symmetric key. The callable services that perform secure import and export operations are:

- Symmetric Key Generate (CSNDSYG)
- Symmetric Key Export (CSNDSYX and CSNFSYX)
- Symmetric Key Import (CSNDSYI and CSNFSYI)

For each of these services, a profile in the CSFKEYS class will control access to the asymmetric key. In addition to specifying user access to the key, the CSFKEYS profile can also specify information (in the ICSF segment of the profile) on how the key can be used. The ASYMUSAGE field of the ICSF segment enables you to specify whether an asymmetric key covered by the profile can participate in secure import or export operations. By specifying the NOSECUREEXPORT keyword in the ASYMUSAGE field, you restrict any asymmetric key covered by the profile from being used to encrypt or decrypt the symmetric key in these operations.

For example, the profile RSA.SAMMY.DIGSIG in class CSFKEYS covers an RSA key pair that should be used only for generating and verifying digital signatures and performing TLS/SSL handshakes. The following RALTER command modifies the profile to ensure that the public key of the RSA key pair is never used to export keys. The SETROPTS RACLIST command is used to refresh the profile in common storage.

```
RALTER CSFKEYS RSA.SAMMY.DIGSIG ICSF(ASYMUSAGE(NOSECUREEXPORT))
SETROPTS RACLIST(CSFKEYS) REFRESH
```

In order for the secure import/export restriction to take effect, you will need to enable the PKA Key Management Extensions control by creating a CSF.PKAEXTNS.ENABLE profile in class XFACILIT. In order to enable the PKA Key Management Extensions control, the Key Store Policy for both the CKDS and the PKDS must also be active. Refer to "Enabling PKA Key Management Extensions" on page 32 for more information.

When the PKA Key Management Extensions control is enabled, the default is to allow keys to participate in secure import and export operations. You can also explicitly specify this using the SECUREEXPORT keyword in the ASYMUSAGE field of a CSFKEYS profile. For example:

```
RALTER CSFKEYS RSA.SAMMY.EXPORT ICSF(ASYMUSAGE(SECUREEXPORT))
SETROPTS RACLIST(CSFKEYS) REFRESH
```

The ASYMUSAGE field can also contain the NOHANDSHAKE or HANDSHAKE keywords to specify whether keys covered by the profile can participate in handshake operations (as described in "Restricting asymmetric keys from being

used in handshake operations"). These keywords can be specified along with the NOSECUREEXPORT or SECUREEXPORT keywords when entering the RDEFINE or RALTER command.

```
RALTER CSFKEYS RSA.SAMMY.EXPORT ICSF(ASYMUSAGE(SECUREEXPORT NOHANDSHAKE))
SETROPTS RACLIST(CSFKEYS) REFRESH
```

Restricting asymmetric keys from being used in handshake operations:
Using the ASYMUSAGE field in the ICSF segment of CSFKEYS profiles enables you to restrict asymmetric keys covered by the profile from being used in handshake operations. The following callable services all identify an asymmetric key to be used in a handshake operation. The callable services that perform handshake operations are:

- Digital Signature Generate (CSNDDSG and CSNFDSG)
- Digital Signature Verify (CSNDDSV and CSNFDSV)
- PKA Encrypt (CSNDPKE and CSNFPKE)
- PKA Decrypt (CSNDPKD and CSNFPKD)

For each of these services, a profile in the CSFKEYS class will control access to the asymmetric key used to generate/verify a digital signature, or encrypt/decrypt a clear key value. In addition to specifying user access to the key, the CSFKEYS profile can also specify information (in the ICSF segment of the profile) on how the key can be used. The ASYMUSAGE field of the ICSF segment enables you to specify whether an asymmetric key covered by the profile can participate in handshake operations. By specifying the NOHANDSHAKE keyword in the ASYMUSAGE field, you restrict any key covered by the profile from being used in handshake operations. For example, the profile RSA.SAMMY.EXPORT in class CSFKEYS covers an RSA key pair intended for exporting and importing symmetric keys. The following RALTER command modifies the profile to ensure that the RSA keys are not used in handshake operations. The SETROPTS RACLIST command is used to refresh the profile in common storage.

```
RALTER CSFKEYS RSA.SAMMY.EXPORT ICSF(ASYMUSAGE(NOHANDSHAKE))
SETROPTS RACLIST(CSFKEYS) REFRESH
```

In order for the restriction on handshake operations to take effect, you will need to enable the PKA Key Management Extensions control by creating a CSF.PKAEXTNS.ENABLE profile in class XFACILIT. In order to enable the PKA Key Management Extensions control, the Key Store Policy for both the CKDS and the PKDS must also be active. Refer to "Enabling PKA Key Management Extensions" on page 32 for more information.

When the PKA Key Management Extensions control is enabled, the default is to allow keys to participate in handshake operations. You can also explicitly specify this using the HANDSHAKE keyword in the ASYMUSAGE field of profiles in the CSFKEYS class. For example:

```
RALTER CSFKEYS RSA.SAMMY.EXPORT ICSF(ASYMUSAGE(HANDSHAKE))
SETROPTS RACLIST(CSFKEYS) REFRESH
```

The ASYMUSAGE field can also contain the NOSECUREEXPORT or SECUREEXPORT keywords to specify whether keys covered by the profile can participate in secure import and export operations (as described in "Restricting asymmetric keys from being used in secure import and export operations" on page 26). These keywords can be specified along with the NOHANDSHAKE or HANDSHAKE keywords when entering the RDEFINE or RALTER command.

```
RALTER CSFKEYS RSA.SAMMY.EXPORT ICSF(ASYMUSAGE(NOSECUREEXPORT HANDSHAKE))
SETROPTS RACLIST(CSFKEYS) REFRESH
```

Placing restrictions on exporting symmetric keys: The Symmetric Key Export (CSNDSYX or CSNFSYX) callable service lets a calling application transfer a symmetric (AES or DES) key from encryption under a master key to encryption under an application-supplied RSA public key. This callable service is needed because a secure key (which is encrypted under a master key in the ICSF environment) might need to be shared with a partner, and to transfer it to that partner securely, it will need to be encrypted under an RSA key provided by the partner. The partner will then be able to decrypt it using a corresponding private key. Due to the nature of the operation performed by the Symmetric Key Export callable service, you may want to place additional restrictions on its use. "Increasing the level of authority required to export symmetric keys" on page 23 describes how you can enable the Symmetric Key Label Export controls to specify that a user needs UPDATE authority in the XCSFKEY class (instead of the default READ authority in the CSFKEYS class) to export a symmetric key. By enabling the PKA Key Management Extensions control, can also specify that a symmetric key covered by a CSFKEYS or XCSFKEY profile:

- cannot be exported.
- can be exported by any asymmetric key in the PKDS
- can be exported only by certain asymmetric keys in the PKDS (as specified by a supplied list).
- can be exported by any asymmetric key, provided it is bound to an identity in a key certificate in a trusted certificate repository (either a PKCS #11 token or a SAF key ring).
- can be exported only by an asymmetric key that is bound to certain identities (as specified by a supplied list of key certificates in a trusted certificate repository).

When an application calls the CSNDSYX or CSNFSYX service, access to the symmetric key (the AES or DES key to be re-encrypted) is determined by a profile in the CSFKEYS class or, if the Symmetric Key Label Export control has been enabled, the XCSFKEY class. In addition to specifying user access to the key, the CSFKEYS or XCSFKEY profile can also place restrictions (in the ICSF segment of the profile) on export of the symmetric key. In the ICSF segment of a CSFKEYS or XCSFKEY profile, the SYMEXPORTABLE field contains a keyword that determines if the key can be exported, and, if so, how ICSF will determine the asymmetric keys (the RSA public keys) that can export (re-encrypt) the key.

Table 8. Keyword settings for symmetric key export using the ICSF segment's SYMEXPORTABLE field

This field/keyword	Specifies:
SYMEXPORTABLE(BYNONE)	The symmetric key can not be exported.
SYMEXPORTABLE(BYLIST)	The symmetric key can be exported, but only by certain RSA public keys in the PKDS (as specified by a supplied list), or only by RSA public keys bound to certain identities (as specified by a supplied list of key certificates). • To supply a list of RSA public keys in the PKDS that can export the symmetric key, you use the SYMEXPORTKEYS field on the ICSF segment. You can list the RSA public keys by label, or you can use a special character setting in this field to specify that any RSA public key in the PKDS can export the symmetric key. • To supply a list a key certificates, you use the SYMEXPORTCERTS field of the ICSF segment. You can list the certificates by label, or you can use a special character setting in this field to specify that any RSA public key bound to an identity in any certificate in the repository can export the symmetric key.
SYMEXPORTABLE(BYANY)	There are no additional restrictions placed on export of the key. Provided no other access requirement or control prevents it, the symmetric key can be exported by any asymmetric key. This is the default.

- For more information on the BYNONE keyword, refer to "Restricting the symmetric key from being exported" on page 29.

- For more information on using the BYLIST keyword and the SYMEXPORTKEYS field, refer to “Identifying RSA public keys that can export the symmetric key.”
- For more information on using the BYLIST keyword and the SYMEXPORTCERTS field, refer to “Identifying key certificates for symmetric key export” on page 30.
- For more information on the BYANY keyword, refer to “Placing no additional restrictions on symmetric key export” on page 32.

Restricting the symmetric key from being exported: CSFKEYS and XCSFKEY profiles can contain an ICSF segment. Fields of the ICSF segment specify rules for key use. In the SYMEXPORTABLE field of the ICSF segment, the BYNONE keyword specifies that the symmetric key(s) covered by the profile can not be exported, regardless of a user's access authority to the key. If an application attempts to use the Symmetric Key Export (CSNDSYX or CSNFSYX) callable service to transfer a symmetric (AES or DES) key covered by the profile, the operation will fail and the service will return an error.

For example, the CKDS contains a DES key labeled DES.BRADY.CASTLE that should never be exported. The Symmetric Key Label Export control for DES keys has not been enabled, so the key is covered by a profile in the CSFKEYS class. The following RALTER command modifies the discrete profile DES.BRADY.CASTLE to indicate that the key should never be exported. The SETROPTS RACLIST command is used to refresh the profile in common storage.

```
RALTER CSFKEYS DES.BRADY.CASTLE ICSF(SYMEXPORTABLE(BYNONE))
SETROPTS RACLIST(CSFKEYS) REFRESH
```

Identifying RSA public keys that can export the symmetric key: CSFKEYS and XCSFKEY profiles can contain an ICSF segment. Fields of the ICSF segment specify rules for key use. In the SYMEXPORTABLE field of the ICSF segment, the BYLIST keyword specifies that the symmetric key(s) covered by the profile can be exported by keys identified using the SYMEXPORTKEYS or SYMEXPORTCERTS fields.

Using the SYMEXPORTKEYS field, you can list the RSA public keys in the PKDS that are allowed to export the symmetric key. The SYMEXPORTKEYS list consists of one or more PKDS key labels identifying the RSA public keys under which the symmetric key can be re-encrypted. These labels follow the normal ICSF label conventions; they can be space separated, and quotes are optional.

Note: Key Store Policy must be active in order for the PKA Key Management Extensions to be enabled. Because Key Store Policy for the PKDS is active, ICSF knows the key label(s) associated with each key token. Tokens associated with multiple labels are considered equivalent. Be aware that as long as one of the labels associated with the token appears in the SYMEXPORTKEYS list, the RSA public key can export symmetric key.

A special key label is the asterisk character (*). If the SYMEXPORTKEYS field contains this special key label, any RSA public key in the PKDS can export the symmetric key (provided no other access requirement or control prevents it).

If an application attempts to use the Symmetric Key Export (CSNDSYX or CSNFSYX) callable service to transfer a symmetric (AES or DES) key covered by the profile, ICSF will compare the RSA public key identified by the application with those identified in the SYMEXPORTKEYS list. If the key is in the list, the operation is allowed to continue. If it is not in the list, and is also not bound to an identity in a

certificate listed in the SYMEXPORTCERTS field (as described in "Identifying key certificates for symmetric key export"), the operation will fail and the service will return an error.

For example, the following RALTER command modifies the discrete profile DES.BRADY.CASTLE so that the DES key it covers can be exported only by the RSA public key RSA.BRADY.CASTLE. In this example, the Symmetric Key Label Export control has been enabled for DES keys, so the DES.BRADY.CASTLE profile is defined in the XCSFKEY class. The SETROPTS RACLIST command is used to refresh the profile in common storage.

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(SYMEXPORTABLE(BYLIST) SYMEXPORTKEYS(RSA.BRADY.CASTLE))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

To instead allow any RSA public key in the PKDS to export the symmetric key covered by the DES.BRADY.CASTLE profile, you would specify the asterisk character (*) in the SYMEXPORTKEYS field.

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(SYMEXPORTABLE(BYLIST) SYMEXPORTKEYS(*))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

The ADDSYMEXPORTKEYS keyword of the ICSF segment enables you to add labels to a SYMEXPORTKEYS list without having to recreate the entire list. For example, to add the label RSA.BKNIGHT.CASTLE to the list, you would enter:

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(ADDSYMEXPORTKEYS(RSA.BKNIGHT.CASTLE))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

Similarly, you can delete labels from a SYMEXPORTKEYS list using the DELSYMEXPORTKEYS keyword:

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(DELSYMEXPORTKEYS(RSA.BKNIGHT.CASTLE))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

You can also delete the entire SYMEXPORTKEYS field using the NOSYMEXPORTKEYS keyword.

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(NOSYMEXPORTKEYS)
SETROPTS RACLIST(XCSFKEY) REFRESH
```

Identifying key certificates for symmetric key export: CSFKEYS and XCSFKEY profiles can contain an ICSF segment. Fields of the ICSF segment specify rules for key use. In the SYMEXPORTABLE field of the ICSF segment, the BYLIST keyword specifies that the symmetric key(s) covered by the CSFKEYS or the XCSFKEY profile can be exported by keys identified using the SYMEXPORTKEYS or SYMEXPORTCERTS fields.

Using the SYMEXPORTCERTS field, you can supply a list of certificate labels in a trusted certificate repository (either a PKCS #11 token or a SAF key ring). As described in "Enabling PKA Key Management Extensions" on page 32, you enable the PKA Key Management Extensions control by creating a CSF.PKAEXTNS.ENABLE profile in class XFACILIT. You can use the APPLDATA field in that profile to identify the type and name of the trusted certificate repository. If the APPLDATA field is not used to provide this information, the default certificate repository is a PKCS #11 token named CSF.TRUSTED.KEYRING. The format of the SYMEXPORTCERTS field depends on whether the trusted certificate repository is a PKCS #11 token or a SAF key ring.

- If the trusted certificate repository is a PKCS #11 token, the certificate labels are listed in the format '*cka-id*/*cert-label*', where:

  cka-id is the CKA_ID attribute of the certificate object. This portion of the

specification is optional, and only necessary if multiple certificate objects have the same CKA_LABEL. If provided, RACF will convert this portion of the specification into uppercase before storing it in the profile.

/cert-label
is the CKA_LABEL attribute of the certificate object. Note that the forward slash character (/) is required even if the optional *cka-id* portion of the specification is omitted. If this portion of the specification contains blank characters, the entire specification must be enclosed in single quotes.

- If the trusted certificate repository is a SAF key ring, the certificate labels are listed in the format '*userID*/*cert-label*', where:

  userID is the owner of the certificate. This portion of the specification is optional, and only necessary if multiple certificates have the same label. If provided, RACF will convert this portion of the specification into uppercase before storing it in the profile.

  /cert-label
  is the label of the digital certificate that was assigned when the certificate was created. Note that the forward slash character (/) is required even if the optional *userID* portion of the specification is omitted. If this portion of the specification contains blank characters, the entire specification must be enclosed in single quotes.

Regardless of whether you are using a PKCS #11 token or a SAF key ring, you can also use the asterisk character (*) in the SYMEXPORTCERTS field to match any certificate in the trusted certificate repository. Using the asterisk character in the SYMEXPORTCERTS field is the same as listing all the certificates in the trusted certificate repository.

If an application attempts to use the Symmetric Key Export (CSNDSYX or CSNFSYX) callable service to transfer a symmetric (AES or DES) key covered by the profile, ICSF will compare the RSA public key identified by the application with those bound to identities in certificates in the SYMEXPORTCERTS list. If any of the listed certificates contains the RSA public key, the operation is allowed to continue. If none of the listed certificates contain the public key, and the key is also not listed in the SYMEXPORTKEYS field (as described in “Identifying RSA public keys that can export the symmetric key” on page 29), the operation will fail and the service will return an error.

For example, say you want to allow export of a the symmetric key DES.BRADY.CASTLE only by the user and public key bound by a certificate in a SAF key ring. The SAF key ring was identified to ICSF when the PKA Key Management Extensions control was enabled (using the APPLDATA field of the CSF.PKAEXTNS.ENABLE profile). The label of the digital certificate in the SAF key ring is "Mister Ink", and the discrete profile covering the key has already been defined in the XCSFKEY class. The following RALTER command specifies that the only RSA public key that can export the symmetric key is the one bound to the identity in the "Mister Ink" certificate. The SETROPTS RACLIST command is used to refresh the profile in common storage.

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(SYMEXPORTABLE(BYLIST) SYMEXPORTCERTS('/Mister Ink'))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

The preceding example assumes that no other certificates have the same label. If other certificates do have the same label, you would want to include the user ID of the certificate owner in the SYMEXPORTCERTS list specification. For example, if the user BKNIGHT is the certificate owner, you would enter:

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(SYMEXPORTABLE(BYLIST) SYMEXPORTCERTS('BKNIGHT/Mister Ink'))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

You can also use the asterisk character (*) in the SYMEXPORTCERT field to match any certificate in the certificate repository.

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(SYMEXPORTABLE(BYLIST) SYMEXPORTCERTS(*))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

The ADDSYMEXPORTCERTS keyword of the ICSF segment enables you to add certificate labels to a SYMEXPORTCERTS list without having to recreate the entire list. For example, to add the certificate 'SERRIN/Mister Ink' to the list of certificate labels, you would enter:

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(ADDSYMEXPORTCERTS('SERRIN/Mister Ink'))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

Similarly, you can delete certificate labels from a SYMEXPORTCERTS list using the DELSYMEXPORTCERTS keyword:

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(DELSYMEXPORTCERTS('BKNIGHT/Mister Ink'))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

You can also delete the entire SYMEXPORTCERTS field using the NOSYMEXPORTCERTS keyword.

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(NOSYMEXPORTCERTS)
SETROPTS RACLIST(XCSFKEY) REFRESH
```

Placing no additional restrictions on symmetric key export: If no keyword value is specified in the ICSF segment's SYMEXPORTABLE field, then, by default, no additional restrictions are placed on the export of symmetric keys covered by the profile. Provided no other access requirement or control prevents it, the symmetric key can be exported by any RSA public key. Although this is the default behavior, you can also explicitly specify it using the BYANY keyword. You might want to do this, for example, if you had previously specified the BYNONE or BYLIST keyword in the SYMEXPORTABLE field, and now want to return to the default behavior.

For example, to specify that there are no restrictions on the export of the symmetric key covered by the profile DES.BRADY.CASTLE in the XCSFKEY class, and that any RSA key can be used in the export operation (provided the user has access permission to the key), you could enter the following RALTER command. The SETROPTS RACLIST command is used to refresh the profile in common storage.

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(SYMEXPORTABLE(BYANY))
SETROPTS RACLIST(CSFKEYS) REFRESH
```

You can also return to the default behavior by deleting the entire SYMEXPORTABLE field using the NOSYMEXPORTABLE keyword.

```
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(NOSYMEXPORTABLE)
SETROPTS RACLIST(XCSFKEY) REFRESH
```

Enabling PKA Key Management Extensions: The rules for cryptographic key usage defined in the ICSF segment of CSFKEYS and XCSFKEY profiles (described in “Restricting asymmetric keys from being used in secure import and export operations” on page 26, “Restricting asymmetric keys from being used in handshake operations” on page 27, and “Placing restrictions on exporting symmetric keys” on page 28) will not be in effect unless PKA Key Management Extensions are enabled. PKA Key Management Extensions cannot be enabled unless Key Store Policy is active for both the CKDS and PKDS.

Enabling any one of the following controls will activate Key Store Policy for a CKDS:

- CSF.CKDS.TOKEN.CHECK.LABEL.WARN
- CSF.CKDS.TOKEN.CHECK.LABEL.FAIL
- CSF.CKDS.TOKEN.NODUPLICATES

Enabling any one of the following controls will activate Key Store Policy for a PKDS:

- CSF.PKDS.TOKEN.CHECK.LABEL.WARN
- CSF.PKDS.TOKEN.CHECK.LABEL.FAIL
- CSF.PKDS.TOKEN.NODUPLICATES

The following table shows the controls for enabling PKA Key Management Extensions in either warning or fail mode. To enable one of the controls, create the appropriate profile in the XFACILIT class. The XFACILIT class does not need to be active or RACLISTed for the control to be enabled.

Table 9. Key Store Policy controls: The PKA Key Management Extensions controls

The existence of this resource profile in the XFACILIT class:	**Does this:**
CSF.PKAEXTNS.ENABLE.WARNONLY	Enables PKA Key Management Extensions in warning mode. The ICSF segment of CSFKEYS or XCSFKEY profiles will be checked to: • determine if a symmetric key can be exported, and, if so, which asymmetric keys can be used in the operation to re-encrypt the symmetric key. • determine if an asymmetric key can be used in secure export and import operations, or in handshake operations. However, because this is warning mode, ICSF will allow the operation to continue even if the ICSF segment indicates that the operation is not allowed.
CSF.PKAEXTNS.ENABLE	Enables PKA Key Management Extensions in fail mode. The ICSF segment of CSFKEYS or XCSFKEY profiles will be checked to: • determine if a symmetric key can be exported, and, if so, which asymmetric keys can be used in the operation to re-encrypt the symmetric key. • determine if an asymmetric key can be used in secure export and import operations, or in handshake operations. If the ICSF segment indicates that the operation is not allowed, the service returns with an error.

For example, you've already used the ICSF segment of profiles in the CSFKEYS or XCSFKEY class to define various restrictions on how keys covered by the profiles can be used. You're not certain that all applications at your installation are using the keys according to the new restrictions, and do not want to disrupt current work patterns at your installation. For this reason, you decide to allow a warning period during which you can identify noncompliant applications without causing application failure. To do this, you would:

1. Enable PKA Key Management Extensions in warning mode:

   ```
   RDEFINE XFACILIT CSF.PKAEXTNS.ENABLE.WARNONLY
   ```

2. Because you have enabled PKA Key Management Extensions in warning mode, ICSF will allow applications to use keys in ways that violate ICSF segment specifications. However, ICSF will generate SMF type 82 subtype 27 records for any violation. Using the information in these records, you can modify your installation's applications as needed.
3. When you are ready to move to a stricter implementation of the policy, you enable the PKA Key Management Extensions control for fail mode, and disable the one for warning mode.

```
RDEFINE XFACILIT CSF.PKAEXTNS.ENABLE
RDELETE XFACILIT CSF.PKAEXTNS.ENABLE.WARNONLY
```

If you accidentally enable PKA Key Management Extensions in both warning and fail mode, the control for fail mode will take precedence.

As described in “Identifying key certificates for symmetric key export” on page 30, you can use the ICSF segment's SYMEXPORTCERTS field to provide a list of certificate labels in a trusted certificate repository (either a PKCS #11 token or a SAF key ring). This enables you to specify that symmetric keys covered by a CSFKEYS or XCSFKEY profile can be exported only by RSA public keys that are bound to identities in the listed certificates. If using the SYMEXPORTCERTS field to provide a list of certificate labels in a trusted certificate repository, you will need to identify that trusted certificate repository to ICSF. You do this using the APPLDATA field of the CSF.PKAEXTNS.ENABLE profile. If the trusted key repository is a PKCS #11 token, it should be identified in the APPLDATA field in the format *TOKEN*/*PKCS-token-name*. If the trusted key repository is a SAF key ring, it should be identified in the APPLDATA field in the format *userID*/*key-ring-name*. For example, if the trusted key repository was a SAF key ring named TRUSTED.KEY.EXPORTERS created by BOBADMIN, you would enter:

```
RDEFINE XFACILIT CSF.PKAEXTNS.ENABLE APPLDATA(BOBADMIN/TRUSTD.KEY.EXPORTERS)
```

If an APPLDATA field is not provided on the CSF.PKAEXTNS.ENABLE, the default certificate repository is a PKCS #11 token named CSF.TRUSTED.KEYRING.

PKA key management extensions example: The following example provides additional illustration of the ICSF segment fields and keywords that you can use to place restrictions on how cryptographic keys can be used.

A DES key has been created for encrypting transactions between a Company and its Business Partner. The Business Partner's public key has previously been added to the PKDS for the purpose of exporting the DES key. The Company's security administrator wants to be sure that only the Business Partner's public key can be used to export the DES key that the Company and its Business Partner are sharing. There is already a profile covering the label of the RSA public key in the PKDS, but no profile covering the label of the new DES key. The security administrator needs to alter the profile for the RSA public key label, and define a new profile for the DES key label. The security administrator has also enabled the Symmetric Key Label Export Control to increase the level of authority needed to export symmetric keys, and so the profile covering the DES key is defined in the XCSFKEY class.

```
RALTER CSFKEYS RSA.BRADY.CASTLE ICSF(ASYMUSAGE(SECUREEXPORT NOHANDSHAKE))
RDEFINE XCSFKEY DES.BRADY.CASTLE ICSF(SYMEXPORTABLE(BYLIST) SYMEXPORTKEYS(RSA.BRADY.CASTLE)) UACC(NONE)
PERMIT DES.BRADY.CASTLE CL(XCSFKEY) ID(SAMPRTNR) UPDATE
SETROPTS RACLIST(CSFKEYS) REFRESH
SETROPTS RACLIST(XCSFKEY) REFRESH
```

Key Store Policy is active for both the CKDS and PKDS, so the security administrator only needs to enable the PKA Key Management Extensions control:

```
RDEFINE XFACILIT CSF.PKAEXTNS.ENABLE
```

Later, the security administrator wants further restrictions on exporting the DES key that the Company and its Business Partner are sharing. The security administrator wants to bind an existing RSA public key to an identity, and allow export of the DES key only by the user and public key bound by a particular certificate. The security administrator creates the certificate for the RSA key, creates a SAF key ring, and adds the certificate to the key ring.

```
RACDCERT ID(BOBADMIN) GENCERT                                              +
         SUBJECTSDN(CN('Mister Ink Inc')O('Business Partner')C('uk'))      +
         WITHLABEL('Mister Ink')SIGNWITH(CERTAUTH LABEL(LocalCertauth'))   +
         KEYUSAGE(DOCSIGN)                                                 +
         NOTAFTER(DATE(2020-12-31))                                        +
         FROMICSF(RSA.BRADY.CASTLE)                                        +
RACDCERT ID(BOBADMIN) ADDRING(TRUSTD.KEY.EXPORTERS)
RACDCERT ID(BOBADMIN) CONNECT(LABEL('Mister Ink' RING(TRUSTD.KEY.EXPORTERS)  +
         USAGE(PERSONAL))
RALTER XCSFKEY DES.BRADY.CASTLE ICSF(NOSYMEXPORTKEYS                       +
         SYMEXPORTCERTS('/Mister Ink'))
SETROPTS RACLIST(XCSFKEY) REFRESH
```

Because the security administrator knows that only one certificate with the label "Mister Ink" will be present in the key ring, he does not specify the user ID portion of the string in the SYMEXPORTCERTS list. Note, however, that the security administrator still needs to include the forward slash (/) delimiter even though a user ID was not provided. Also note that the NOSYMEXPORTKEYS keyword is used to remove the SYMEXPORTKEYS list that had been previously defined.

The security administrator modifies the CSF.PKAEXTNS.ENABLE profile in the XFACILIT class to identify the SAF key ring as the certificate repository.

```
RDEFINE XFACILIT CSF.PKAEXTNS.ENABLE APPLDATA(TRUSTD.KEY.EXPORTERS)
```

For more information on the ICSF fields and keywords, refer to “Restricting asymmetric keys from being used in secure import and export operations” on page 26, “Restricting asymmetric keys from being used in handshake operations” on page 27, and “Placing restrictions on exporting symmetric keys” on page 28.

Chapter 4. Update of z/OS Cryptographic Services ICSF Application Programmer's Guide, SA22-7522-12, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522-12,, for the PKA Key Management Extensions enhancements provided by this APAR. Refer to this source document if background information is needed.

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 4758 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.

This callable service supports invocation in AMODE(64). 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: Output Type: Integer

The return code specifies the general result of the callable service.

reason_code

Direction: Output Type: Integer

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: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit_data* parameter.

exit_data

Direction: Input/Output Type: String

The data that is passed to the installation data.

rule_array_count

Direction: Input Type: Integer

The number of keywords you are supplying in *rule_array*. Value must be 1 or 2

rule_array

Direction: Input Type: 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 10.

Table 10. Keywords for ICSF Query Service

Keyword	Meaning
Coprocessor (optional) - parameter is ignored for ICSFSTAT.	
COPROCxx	Specifies the specific coprocessor to execute the request. xx may be 00 through 63 inclusive. This may be the processor number of a PCICC or a PCIXCC/CEX2C.
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.
Information to return (required)	
ICSFSTAT	Get ICSF related status information.
ICSFST2	Get additional ICSF related status information.
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 10. Keywords for ICSF Query Service (continued)

Keyword	Meaning
STATCARD	Get coprocessor-related basic status information.
STATDIAG	Get coprocessor-related basic status information.
STATEID	Get coprocessor-related basic status information.
STATEXPT	Get coprocessor-related basic status information.

returned_data_length

Direction: Input/Output Type: 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: Output Type: String

This field will contain the output from the service. It has the format of 8-byte elements of character data.

The format of the output *returned_data* depends on the value of the input *rule_array* and the information requested. Different information is returned depending on what the input keyword is.

For *returned_data* 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 with contain the character string '2 '.

For ICSFSTAT, the coprocessor keyword is ignored. The output *returned_data* for the ICSFSTAT keyword is defined in Table 11.

Table 11. 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” on page 51)

Table 11. Output for option ICSFSTAT (continued)

3	ICSF Status Field 2	Status of ICSF **Number** **Meaning** **0** 64-bit callers not supported **1** 64-bit callers supported **2** PKCS #11 is available. This is returned when ICSF is running on z/OS V1R9 or later.
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
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

Table 11. Output for option ICSFSTAT (continued)

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 (PCICAs) **Number** **Meaning** **0** Not available **1** At least one available for application use.
11	Future Use	Currently blanks
12	Future Use	Currently blanks

For ICSFST2 the coprocessor rule array keyword is ignored. The output *returned_data* for the ICSFST2 keyword is defined in Table 12.

Table 12. 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” on page 51)
4	ICSF Status Field 2	Status of ICSF **Number** **Meaning** **0** PKCS #11 is not available **1** PKCS #11 is available

Table 12. Output for option ICSFST2 (continued)

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
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 12. Output for option ICSFST2 (continued)

		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.** 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

Table 12. Output for option ICSFST2 (continued)

		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. 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.
8	Future use	Currently blanks
9	Future use	Currently blanks
10	Future use	Currently blanks
11	Future use	Currently blanks
12	Future use	Currently blanks

Table 13. Output for option STATAES

Element Number	Name	Description

Table 13. Output for option STATAES (continued)

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 14. Output for option STATCCA

Element Number	Name	Description
1	NMK Status	State of the DES New Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a partially complete key **3** Register contains a complete key
2	CMK Status	State of the DES Current Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a key
3	OMK Status	State of the DES Old Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a 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 15. Output for option STATCCAE

Element Number	Name	Description
1	Symmetric NMK Status	State of the DES Symmetric New Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a partially complete key **3** Register contains a complete key
2	Symmetric CMK Status	State of the DES Symmetric Current Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a key
3	Symmetric OMK Status	State of the DES Symmetric Old Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a 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.
7	Asymmetric NMK Status	State of the Asymmetric New Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a partially complete key **3** Register contains a complete key
8	Asymmetric CMK Status	State of the Asymmetric Current Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a key
9	Asymmetric OMK Status	State of the Asymmetric Old Master Key Register: **Number** **Meaning** **1** Register is clear **2** Register contains a key

Table 16. 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.
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.

Table 17. 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
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

Table 17. Output for option STATDIAG (continued)

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
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 18. Output for option STATEID

Element Number	Name	Description
1	EID	During initialization, a value of zero is set in the coprocessor.

Table 19. Output for option STATEXPT

Element Number	Name	Description

Table 19. Output for option STATEXPT (continued)

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 **1** CDMF encryption is 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 19. Output for option STATEXPT (continued)

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** DSA not available **1024** DSA 1024 key size **2048** DSA 2048 key size **4096** RSA 4096 key size

reserved_data_length

Direction: Input Type: Integer

The length of the *reserved_data* parameter. Currently, the value must be 0.

reserved_data

Direction: Input Type: String

This field is currently not used.

Restrictions

Caller must be task mode and must not be SRB mode, when running on z900/z800 servers with PCICC.

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 PCICC or PCIXCC/CEX2C that is ACTIVE

The options ICSFSTAT and ICSFST2 report on the state of PKA callable services. ICSFSTAT reports it in element 2. ICSFST2 reports it in element 3. There is a subtle difference between the two 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 reports PKA callable services as enabled when the DES and/or AES master key is loaded and valid.

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

Table 20. ICSF Query Service required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 800 IBM @server zSeries 900	None.	
IBM @server zSeries 990 IBM @server zSeries 890	None.	
IBM System z9 EC IBM System z9 BC	None.	
IBM System z10 EC IBM System z10 BC	None	

Reason Codes for Return Code 8 (8)

Table 21 shows the reason codes related to the PKA Key Management Extensions. These reason codes are returned from certain callable services that give return code 8.

Table 21. PKA Key Management Extensions Reason Codes for Return Code 8 (8)

Reason Code Hex (Decimal)	Description
BF5 (3061)	The provided asymmetric key identifier can not 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 can not 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.

Chapter 5. Update of z/OS Cryptographic Services ICSF System Programmer's Guide, SA22-7520-13, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF System Programmer's Guide*, SA22-7520-13, for the PKA Key Management Extensions enhancements provided by this APAR. Refer to this source document if background information is needed.

SMF Records

SMF records are documented in *z/OS MVS System Management Facilities (SMF)* and published on release boundaries. As a migration aid for HCR7751, which is not on a release boundary, new and changed SMF records for FMID HCR7751 are listed here.

SMF type 82 subtype 14 - PCI Cryptographic Coprocessor Master Key Entry

Table 22. SMF type 82 subtype 14

Offset (Dec)	Offset (Hex)	Name	Length	Format	Description
0	0	SMF82AAB	4	binary	Flags **Bit** Meaning when set **0** DES NMK verification pattern is valid. **1** Asymmetric-key NMK verification pattern is valid. **2** DES key part verification pattern is valid. **3** Asymmetric-Key key part verification pattern is valid. **4** AES NMK verification pattern is valid. **5** AES key part verification pattern is valid. **6** Reserved for future use **7** Reserved for future use **8** Coprocessor is not a PCI Cryptographic Coprocessor **9** Coprocessor is a PCI X Cryptographic Coprocessor **10** Coprocessor is a CEX2C **11** Reserved for future use The remaining bits are reserved for future use.

SMF type 82 subtype 24 - Duplicate key tokens

This record is generated only when the security administrator has indicated that duplicate key tokens must be identified. The label of each secure token that makes up the set of duplicates is in the record.

Note: No NULL token labels are listed in an SMF type 82 subtype 24 record. More detail about the duplicated tokens can be generated by the CSFDUTIL utility. Duplicate tokens appear within the CKDS or the PKDS.

Table 23. SMF type 82 subtype 24

Offset (Dec)	Offset (Hex)	Name	Length	Format	Description
Header/Self defining section (defined by SMF)					
0	0	SMF82LEN	2	binary	Record length. This field and the next field (total of four bytes) form the RDW (record descriptor word). See *z/OS MVS System Management Facilities (SMF)* for a detailed description.
2	2	SMF82SEG	2	binary	Segment descriptor (see record length field).
4	4	SMF82FLG	1	binary	System indicator **Bit** Meaning when set **0–2** Reserved **3–6** Version indicators* **7** Reserved *See *z/OS MVS System Management Facilities (SMF)* for a detailed description.
5	5	SMF82RTY	1	binary	Record type 82 (X'52'')
6	6	SMF82TME	4	binary	Time since midnight, in hundredths of a second, that the record was moved into the SMF buffer.
10	A	SMF82DTE	4	packed	Date when the record was moved into the SMF buffer, in the form 0cyydddF.
14	E	SMF82SID	4	EBCDIC	System identification (from the SID parameter).
18	12	SMF82SSI	4	EBCDIC	Subsystem identification.
22	16	SMF82STY	2	binary	Record subtype. Value is X'18'
24	18	SMF82DCNTSTRT	4	binary	Start duplicate labels
28	1C	SMF82DCNTEND	4	binary	End duplicate labels
32	20	SMF82DCNT	4	binary	Number of duplicate labels
36	24	*	4	*	Reserved
40	28	SMF82DNAM	44	EBCDIC	Name of key data set
The following is repeated 'count' number of times.					
0	0	SMF82DLAB	64	EBCDIC	Labels of duplicate tokens

SMF type 82 subtype 25 – Duplicate Tokens Found

SMF type 82 subtype 25 records are logged for key store policies.

Table 24. SMF type 82 subtype 25

Offset (Dec)	Offset (Hex)	Name	Length	Format	Description
Header/Self defining section (defined by SMF)					
0	0	SMF82LEN	2	binary	Record length. This field and the next field (total of four bytes) form the RDW (record descriptor word).
2	2	SMF82SEG	2	binary	Segment descriptor (see record length field).
4	4	SMF82FLG	1	binary	System indicator **Bit** Meaning when set **0–2** Reserved **3–6** Version indicators* **7** Reserved *See *z/OS MVS System Management Facilities (SMF)* for a detailed description.
5	5	SMF82RTY	1	binary	Record type 82 (X'52'')
6	6	SMF82TME	4	binary	Time since midnight, in hundredths of a second, that the record was moved into the SMF buffer.
10	A	SMF82DTE	4	packed	Date when the record was moved into the SMF buffer, in the form 0cyydddF.
14	E	SMF82SID	4	EBCDIC	System identification (from the SID parameter).
18	12	SMF82SSI	4	EBCDIC	Subsystem identification.
22	16	SMF82STY	2	binary	Record subtype. Value is X'19'
24	18	SMF82KDS	44	EBCDIC	Key data store
68	44	SMF82KLF	4	binary	Key store policy flags **Bit** Meaning when set **0** Warning **1** List is incomplete **2** List is from CKDS **3** List is from PKDS **4–31** Reserved
72	48	SMF82KLC	4	binary	Number of key labels following
The following is repeated 'count' number of times.					
76	4C	SMF82DKL	72	EBCDIC	Unauthorized duplicate key label and key type (one or more of these will follow.)*

*If a label passes CSFKEYS check, then it is stored at offset 76 and the number of key labels following (SMF82KLC) is zero. If none of the labels pass the CSFKEYS check, then beginning at offset 76, the number of key labels following (SMF82KLC) is the number of labels checked. The key labels follow in the log record.

SMF type 82 subtype 26 - PKDS Data Space Refresh

SMF type 82 subtype 26 records are logged after a refresh of the PKDS data space.

Table 25. SMF type 82 subtype 26

Offset (Dec)	Offset (Hex)	Name	Length	Format	Description
Header/Self defining section (defined by SMF)					
0	0	SMF82LEN	2	binary	Record length. This field and the next field (total of four bytes) form the RDW (record descriptor word).
2	2	SMF82SEG	2	binary	Segment descriptor (see record length field).
4	4	SMF82FLG	1	binary	System indicator **Bit** Meaning when set **0–2** Reserved **3–6** Version indicators* **7** Reserved *See *z/OS MVS System Management Facilities (SMF)* for a detailed description.
5	5	SMF82RTY	1	binary	Record type 82 (X'52'')
6	6	SMF82TME	4	binary	Time since midnight, in hundredths of a second, that the record was moved into the SMF buffer.
10	A	SMF82DTE	4	packed	Date when the record was moved into the SMF buffer, in the form 0cyydddF. See ″Standard SMF Record Header″ on page 13-1 for a detailed description.
14	E	SMF82SID	4	EBCDIC	System identification (from the SID parameter).
18	12	SMF82SSI	4	EBCDIC	Subsystem identification.
22	16	SMF82STY	2	binary	Record subtype. Value is X'1A'
24	16	SMF82PREF _FLAG	4	binary	Flags **Bit** Meaning when set **0** PKDS was refreshed **1–31** Reserved
28	1C	SMF82PREF_ OLDDS	44	EBCDIC	Old PKDS name
72	48	SMF82PREF_ NEWDS	44	EBCDIC	New PKDS name

SMF type 82 subtype 27 - PKA Key Management Extensions

SMF Record Type 82 is used to record information about the events and operations of ICSF. Record type 82 is written to the SMF data set at the completion of certain cryptographic functions. SMF Record Type 82, Subtype 27 is used to record information related to PKA Key Management Extensions.

Table 26. SMF type 82 subtype 27

Offset (Dec)	Offset (Hex)	Name	Length	Format	Description
0	0	SMF82LEN	2	binary	Record length. This field and the next field (total of four bytes) form the RDW (record descriptor word).
2	2	SMF82SEG	2	binary	Segment descriptor (see record length field).
4	4	SMF82FLG	1	binary	System indicator: **Bit** **Meaning when set** **0-2** Reserved **3-6** Version indicators **7** Reserved
5	5	SMF82RTY	1	binary	Record type 82 (X'52')
6	6	SMF82TME	4	binary	Time since midnight, in hundredths of a second, that the record was moved into the SMF buffer.
10	A	SMF82DTE	4	packed	Date when the record was moved into the SMF buffer, in the form 0cyydddF.
14	E	SMF82SID	4	EBCDIC	System identification (from the SID parameter).
18	12	SMF82SSI	4	EBCDIC	Subsystem identification.
22	16	SMF82STY	2	binary	Record subtype. Value is '1B'X

Table 26. SMF type 82 subtype 27 (continued)

Offset (Dec)	Offset (Hex)	Name	Length	Format	Description
24	18	SMF82PKE_FLAGS	4	binary	PKA Key Management Extension flags **Bit** **Meaning when set on** **0** PKA token may not be used for requested function **1** SYM token may not be exported by the provided PKA token **2** PKA label list is incomplete **3** SYM label list is incomplete **24** Trusted certificate repository has changed. **25** PKA Key Management Extensions in WARNONLY mode. **26** An error was detected during processing. **27** Trusted cert repository was empty. **28** An error was detected while extracting APPLDATA **29** The repository wasn't found **30** One or more certs were unable to be parsed. Bits 0-3 are set during callable services. Bits 24-30 are set during repository parsing. Bits 4-23 and 31 are reserved.
28	1C	SMF82PKE_FUNCTION	8	EBCDIC	Name of the service that issued this SMF record. The name will be in the form CSFzzz.
36	24	SMF82PKE_APPLDATALEN	1	binary	Length of the enablement profile APPLDATA or current repository name.
37	25	SMF82PKE_APPLDATA	247	EBCDIC	Enablement profile APPLDATA or current repository name.
284	11C	SMF82PKE_FUNCSPEC	0	binary	Function-specific section of the record
284	11C	SMF82PKE_APPLDATA_PARSING	0	binary	APPLDATA parsing results section
284	11C	SMF82PKE_SAF_RC	2	binary	SAF_RC or 'FFFF'X
286	11E	SMF82PKE_SERV_RC	2	binary	RACF RC or ICSF RC

Table 26. SMF type 82 subtype 27 (continued)

Offset (Dec)	Offset (Hex)	Name	Length	Format	Description
288	120	SMF82PKE_SERV_RS	4	binary	RACF RS or ICSF RS
284	11C	SMF82PKE_SERVICE_SECTION	0	binary	Callable services section
284	11C	SMF82PKE_PKA_REC_CNT	4	binary	Number of PKA labels present in this record
288	120	SMF82PKE_SYM_REC_CNT	4	binary	Number of SYM labels present in this record.
The following is repeated SMF82PKE_PKA_REC_CNT number of times.					
292	124	SMF82PKE_PKA_LABELS	64	EBCDIC	PKA key label.
The following is repeated SMF82PKE_SYM_REC_CNT number of times.					
292 + zzz	124 + zzz	SMF82PKE_SYM_LABELS	72	EBCDIC	SYM key label.

Chapter 6. Update of z/OS Cryptographic Services ICSF Messages, SA22-7523-12, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF Messages*, SA22-7523-12, for the PKA Key Management Extensions enhancements provided by this APAR. Refer to this source document if background information is needed.

The following message is added for PKA Key Management Extensions.

CSFM612I PKA KEY EXTENSIONS CONTROL IS *state*

Explanation: If *state* is DISABLED, either the profile that enables the PKA Key Management Extensions control is not defined, or one or both of the profiles that enable Key Token Authorization Checking for the CKDS and PKDS are not defined. If *state* is ENABLED, the profile is defined.

The existence of a profile for the CSF.PKAEXTNS.ENABLE resource in the XFACILIT class enables the PKA Key Management Extensions control. RACF commands can be used to define, change, list, or delete the profiles that cover this resource in the XFACILIT class.

This message may be issued during ICSF initialization or when ICSF detects that the policy is either activated or deactivated.

System action: Processing continues.

Operator response: None

System programmer response: None