

IBM

@server

iSeries

Cryptographic hardware





@server

iSeries

Cryptographic hardware

Note

Before using this information and the product it supports, read the information in "Notices," on page 277.

Seventh Edition (August 2005)

This edition applies to version 5, release 3, modification 0 of IBM Operating System/400 (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 1998, 2005. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

| | | | |
|---|----------|--|------------|
| Cryptographic hardware. | 1 | Configure the Cryptographic Coprocessor | 20 |
| What's new for V5R3 | 2 | Migrate to the Cryptographic Coprocessor | 94 |
| Print this topic. | 3 | Manage the Cryptographic Coprocessor | 120 |
| Cryptography concepts | 3 | Troubleshoot the Cryptographic Coprocessor | 261 |
| Cryptographic hardware concepts | 6 | Related information for cryptographic hardware | 275 |
| 2058 Cryptographic Accelerator for iSeries | 8 | Code disclaimer information | 276 |
| Features | 8 | | |
| Scenario: Enhance iSeries SSL performance | 9 | Appendix. Notices | 277 |
| Plan for the 2058 Cryptographic Accelerator | 10 | Trademarks | 278 |
| Configure the 2058 Cryptographic Accelerator. | 10 | Terms and conditions for downloading and | |
| Cryptographic Coprocessors | 11 | printing publications | 278 |
| Features | 12 | Code disclaimer information | 279 |
| Cryptographic Coprocessor scenarios. | 13 | | |
| Plan for the Cryptographic Coprocessor | 16 | | |

Cryptographic hardware

IBM offers cryptographic hardware solutions for customers who require a high level of security for protecting data stored on their servers and for securing transaction processing such as those involving financial PINs. Cryptographic hardware can also be used to improve the performance of i5/OS SSL transactions.

The cryptographic hardware options available for iSeries along with the benefits of each are listed below:

- IBM 4764 Cryptographic Coprocessor (Hardware Feature code 4806)
 - New PCI-X adapter for iSeries i5 models
 - Tamper-resistant hardware features
 - Provides improved performance over that of the 4758 Cryptographic Coprocessor, and highly secure key management over that of the 2058 Cryptographic Accelerator
 - Offers a rich set of cryptographic functions for applications, including Triple-DES, RSA digital signature support, financial PIN processing, and robust key management services
 - Enables EMV (Europay, MasterCard, Visa) payment system applications using integrated circuit cards (also called smart or chip cards)
 - Numerous configuration options, enabling you to customize its functions to fit your needs
 - Provides secure key storage for applications and SSL transactions
 - Offers a large capacity for accelerating SSL transactions
 - Minimal installation effort
- IBM 4758 Cryptographic Coprocessor (Hardware Feature code 4801)
 - PCI adapter offering tamper-resistant hardware features
 - Offers a rich set of cryptographic functions for applications, including Triple-DES, RSA digital signature support, financial PIN processing, and robust key management services
 - Support to enable the EMV (Europay, MasterCard, Visa) payment system applications using integrated circuit cards (also called smart or chip cards)
 - Numerous configuration options, enabling you to customize its functions to fit your needs
 - Provides secure key storage for applications and SSL transactions
 - Minimal installation effort
- IBM 2058 e-Business Cryptographic Accelerator (Hardware Feature code 4805)
 - PCI adapter offering large capacity for accelerating SSL transactions
 - Minimal installation and configuration effort
 - Minimal management requirements

By installing cryptographic hardware in your server, you can make cryptography an integral part of your security plan. To ensure that you understand how cryptographic hardware works, review the following pages:


- What's new for V5R3 makes available to you the new features or functions available for this release of the server.
- Print this topic explains the printing options available to you for this topic.
- Usage scenarios offer some example configurations or uses of cryptographic hardware with iSeries™ servers:
 - Cryptographic hardware scenario: Enhance iSeries SSL performance
 - Cryptographic hardware scenario: Protect private keys with cryptographic hardware


- | – Cryptographic hardware scenario: Write an i5/OS or OS/400® application to use the Cryptographic Coprocessor
- | • Cryptography concepts enable you to better understand how to maximize your usage of cryptography with your server.
- | • Cryptographic hardware concepts explain some basic concepts regarding cryptographic hardware, enabling you to better understand how to maximize your usage of cryptography and cryptographic hardware options with your iSeries server.
- | • Choosing the best cryptographic hardware for your server:
 - | – Cryptographic Coprocessor for iSeries includes planning and configuration information for the Cryptographic Coprocessors.
- | **Note:** This information includes programming examples. Read the “Code disclaimer information” on page 276 for important legal information.
- | – 2058 Cryptographic Accelerator for iSeries includes planning and configuration information for the 2058 Cryptographic Accelerator.
- | • Related information points to other sources of cryptographic information, as well as related product information sites.

What’s new for V5R3

If you are looking for the latest information regarding new cryptographic hardware, and added features to existing cryptographic hardware options for your server, you have come to the right place.



The following updates have been made to this information during the V5R3 release:

- | • Support for EMV (Europay, MasterCard, Visa) applications has been added for the IBM Cryptographic Coprocessor. See “Cryptographic Coprocessors” on page 11.
- | • The new IBM 4764 Cryptographic Coprocessor is supported. This coprocessor provides improved performance and includes EMV support.
- | • The table, “Supported number of IBM 4758 Cryptographic Coprocessors” has been updated with new server models that support cryptographic hardware. See “Cryptographic Coprocessors” on page 11.
- | • The “Cryptography concepts” on page 3 have been updated with several additional concepts. Additionally, the concepts which pertain specifically to cryptographic hardware have been moved to a new location, “Cryptographic hardware concepts” on page 6.
- | • A program example has been added for the procedure of migrating Cryptographic Support for iSeries cross-domain key files to your Cryptographic Coprocessor. See “Migrate Cryptographic Support for AS/400 cross-domain key files” on page 95.
- | • The Uniform Resource Locator (URL) for the CCA 2.41 Basic Services Manual has been updated in “Related information for cryptographic hardware” on page 275. Additionally, the URL for the CCA 2.52 Basic Services Manual is also now available.
- | • The CCA 3.20 Basic Services Manual which pertains to the new IBM 4764 Cryptographic Coprocessor is now available in the IBM® PCI Cryptographic Coprocessor documentation library .

To find other information about what’s new or changed this release, see the Memo to Users .

How to see what’s new or changed

To help you see where technical changes have been made, this information uses:

- The  image to mark where new or changed information begins.
- The  image to mark where new or changed information ends.

Print this topic


You can view or download the PDF version of Cryptographic hardware (about 756 KB or 298 pages), which contains all of the information regarding IBM cryptographic hardware supported for the V5R3 release.

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser.
2. Click **Save Target As**.
3. Navigate to the directory in which you would like to save the PDF.
4. Click **Save**.

Downloading Adobe Acrobat Reader

If you need Adobe Acrobat Reader to view or print these PDFs, you can download a copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html) .

Cryptography concepts

Cryptography

Cryptography is the art and science of keeping data secure. Cryptographic services help ensure data privacy, maintain data integrity, authenticate communicating parties, and prevent repudiation (when a party refutes having sent a message).

Basic encryption allows you to store information or to communicate with other parties while preventing non-involved parties from understanding the stored information or understanding the communication. Encryption transforms understandable text (plaintext) into an unintelligible piece of data (ciphertext). Decryption restores the understandable text from the unintelligible data. Both functions involve a mathematical formula (the algorithm) and secret data (the key).

Cryptographic algorithms

There are two types of cryptographic algorithms:

1. With a **secret** or **symmetric** key algorithm, the key is a shared secret between two communicating parties. Encryption and decryption both use the same key. The Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are examples of symmetric key algorithms.

There are two types of symmetric key algorithms:

Block ciphers

In a block cipher, the actual encryption code works on a fixed-size block of data. Normally, the user's interface to the encrypt/decrypt operation will handle data longer than the block size by repeatedly calling the low-level encryption function. If the length of data is not on a block size boundary, it must be padded.

Stream ciphers

Stream ciphers do not work on a block basis, but convert 1 bit (or 1 byte) of data at a time.

2. With a **public key** (PKA) or **asymmetric** key algorithm, a pair of keys is used. One of the keys, the private key, is kept secret and not shared with anyone. The other key, the public key, is not secret and can be shared with anyone. When data is encrypted by one of the keys, it can only be decrypted and recovered by using the other key. The two keys are mathematically related, but it is virtually impossible to derive the private key from the public key. The RSA algorithm is an example of a public key algorithm.

Public key algorithms are slower than symmetric key algorithms. Applications typically use public key algorithms to encrypt symmetric keys (for key distribution) and to encrypt hashes (in digital signature generation).

Together, the key and the cryptographic algorithm transform the data. All of the supported algorithms are in the public domain. Therefore it is the key that controls access to the data. You must safeguard the keys to protect the data.

Cryptographic operations

Different cryptographic operations may use one or more algorithms. You choose the cryptographic operation and algorithm(s) depending on your purpose. For example, for the purpose of ensuring data integrity, you might want to use a MAC (message authentication code) operation with the AES algorithm.

APIs provide access to the various cryptographic operations. The iSeries system supports the following cryptographic API sets.

- **Common Cryptographic Architecture (CCA)** API set is provided for running cryptographic operations on a Cryptographic Coprocessor for iSeries. For more information on the CCA APIs, refer to the Features information for the Cryptographic Coprocessors for iSeries.
- **Cryptographic Services** API set is provided for running cryptographic operations on the 2058 Cryptographic Accelerator or within i5/OS LIC. For more information on the Cryptographic Services APIs, refer to the Cryptographic Services APIs.
- **The Java Cryptography Extension (JCE)** is a standard extension to the Java Software Development Kit. For more information, refer to Java Cryptography Extension.

Data privacy

Cryptographic operations for the purpose of data privacy (confidentiality) prevent an unauthorized person from reading a message. The following operations are included in data privacy:

Encrypt/Decrypt

The encrypt operation changes plaintext data into ciphertext through the use of a cipher algorithm and key. To restore the plaintext data, the decrypt operation must employ the same algorithm and key.

Encryption/decryption may be employed at any level of the operating system. Basically, there are three levels:

Field level encryption

With field level encryption, the user application explicitly requests cryptographic services. The user application completely controls key generation, selection, distribution, and what data to encrypt.

Session level encryption

With encryption at the session layer, the system requests cryptographic services instead of an application. The application may or may not be aware that encryption is happening.

Link level encryption

Link level encryption is performed at the lowest level of the protocol stack, usually by specialized hardware.

The Cryptographic Coprocessors for iSeries and 2058 Cryptographic Accelerator may be used for both field level encryption and Secure Sockets Layer (SSL) session establishment encryption. The system, however, does not use either for VPN.

Translate

The translate operation is used to decrypt data from encryption under one key to encryption under another key. This is done in one step to avoid exposing the plaintext data within the application program.

Data integrity, authenticity, and non-repudiation

Encrypted data does not mean the data can not be manipulated (e.g. repeated, deleted, or even altered). To rely on data, you need to know that it comes from an authorized source and is unchanged. Additional cryptographic operations are required for these purposes.

Hash (Message Digest)

A cryptographic hash operation produces a fixed-length output string (often called a digest) from a variable-length input string. For all practical purposes, the following statements are true of a good hash function:

- Collision resistant: If any portion of the data is modified, a different hash will be generated.
- One-way: The function is irreversible. That is, given a digest, it is not possible to find the data that produces it.

These properties make hash operations useful for authentication purposes. For example, you can keep a copy of a digest for the purpose of comparing it with a newly generated digest at a later date. If the digests are identical, the data has not been altered.

MAC (Message Authentication Code)

A MAC operation uses a secret key and cipher algorithm to produce a value (the MAC) which later can be used to ensure the data has not been modified. Typically, a MAC is appended to the end of a transmitted message. The receiver of the message uses the same MAC key, and algorithm as the sender to reproduce the MAC. If the receiver's MAC matches the MAC sent with the message, the data has not been altered.

The MAC operation helps authenticate messages, but does not prevent unauthorized reading because the transmitted data remains as plaintext. You must use the MAC operation and then encrypt the entire message to ensure both data privacy and integrity.

HMAC (Hash MAC)

An HMAC operation uses a cryptographic hash function and a secret shared key to produce an authentication value. It is used in the same way a MAC is used.

Sign/Verify

A sign operation produces an authentication value called a digital signature. A sign operation works as follows:

1. The data to be signed is hashed, to produce a digest.
2. The digest is encrypted using a PKA algorithm and a private key, to produce the signature.

The verify operation works as follows:

1. The signature is decrypted using the sender's PKA public key, to produce digest 1.
2. The data that was signed is hashed, to produce digest 2.
3. If the two digests are equal, the signature is valid.

Theoretically, this also verifies the sender because only the sender should possess the private key. However, how can the receiver verify that the public key actually belongs to the sender? Certificates are used to help solve this problem. Refer to Digital Certificate Manager for more information on certificates.

Key and random number generation

Many security-related functions rely on random number generation, for example, salting a password or generating an initialization vector. An important use of random numbers is in the generation of cryptographic key material. Key generation has been described as the most sensitive of all computer security functions. If the random numbers are not cryptographically strong, the function will be subject to attack.

The iSeries operating system contains a pseudorandom number generator (PRNG). The PRNG is used by many system functions and is available for application use through the Cryptographic Services API set.

The PRNG is composed of two parts: pseudorandom number generation and seed management. Pseudorandom number generation is performed using the FIPS 186-1 algorithm. Cryptographically strong pseudorandom numbers rely on good seed. The FIPS 186-1 key and seed values are obtained from a system seed digest. The system automatically generates seed using data collected from system information or by using the random number generator function on a cryptographic coprocessor if one is available. System-generated seed can never be truly unpredictable. If a cryptographic coprocessor is not available, you should add your own random seed to the system seed digest. This should be done as soon as possible any time the Licensed Internal Code is installed.

Key management and distribution

Key management operations include key storage, key retrieval, and key form conversions. Various uses of a key will require the key to be in different forms. For example, keys received from other sources will normally be converted to an internal format before storing (an import operation). Likewise, keys sent out of the system are converted to a standard external format before sending (an export operation). Certain key forms are standard, such as an ASN.1 BER-encoded form, and others are peculiar to a cryptographic service provider, such as the cryptographic coprocessors for iSeries. V5R3 key storage on the iSeries system includes the following:

- CCA key store for use with the Cryptographic Coprocessors for iSeries. See “Initialize a key store file” on page 136 for more information.
- Digital Certificate Manager certificate store. Refer to Certificate Stores for more information.
- JCE key store. Refer to the Java Cryptography Extension for more information.

Keys can also be stored within cryptographic hardware. Refer to “Cryptographic Coprocessors” on page 11 for more information.

In addition, the various cryptographic API sets provide operations to help in key distribution - for example, import and export operations and Diffie-Hellman shared secret key generation.

Cryptographic hardware concepts

Key types associated with the Cryptographic Coprocessor

Your Coprocessor uses various key types. Not all DES or Triple DES keys can be used for all symmetric key operations. Likewise, not all public key algorithm (PKA) keys can be used for all asymmetric key operations. This is a list of the various key types which the Coprocessor uses:

Master key

This is a clear key, which means that no other key encrypted it. The Coprocessor uses the master key to encrypt all operational keys. The Coprocessor stores the master key in a tamper-responding module. You cannot retrieve the master key from the Coprocessor. The Coprocessor responds to tamper attempts by destroying the master key and destroying its factory certification. The coprocessors have two master keys: one for encrypting DES keys and one for encrypting PKA keys.

Double-length key-encrypting keys

Your Coprocessor uses this type of Triple-DES key to encrypt or decrypt other DES or Triple DES keys. Key-encrypting-keys are generally used to transport keys between systems. However, they can also be used for storing keys offline for backup. If key-encrypting-keys are used to transport keys, the clear value of the key-encrypting-key itself must be shared between the two systems. Exporter key-encrypting keys are used for export operations where a key encrypted under the master key is decrypted and then encrypted under the key-encrypting key. Importer key-encrypting keys are used for import operations where a key encrypted under the key-encrypting key is decrypted and then encrypted under the master key.

Double-length PIN keys

Your Coprocessor uses this type of key to generate, verify, encrypt, and decrypt PINs used in financial operations. These are Triple DES keys.

MAC keys

Your Coprocessor uses this type of key to generate Message Authentication Codes (MAC). These can be either DES or Triple DES keys.

Cipher keys

Your Coprocessor uses this type of key to encrypt or decrypt data. These can be either DES or Triple DES keys.

Single-length compatibility keys

Your Coprocessor uses this type of key to encrypt or decrypt data and generate MACs. These are DES keys and are often used when encrypted data or MACs are exchanged with systems that do not implement the Common Cryptographic Architecture.

Private keys

Your Coprocessor uses private keys for generating digital signatures and for decrypting DES or Triple DES keys encrypted by the public key.

Public keys

Your Coprocessor uses public keys for verifying digital signatures, for encrypting DES or Triple DES keys, and for decrypting data encrypted by the private key.

Key forms

The Coprocessor works with keys in one of four different forms. The key form, along with the key type, determines how a cryptographic process uses that key. The four forms are:

Clear form

The clear value of the key is not protected by any cryptographic means. Clear keys are not usable by the Coprocessor. The clear keys must first be imported into the secure module and encrypted under the master key and then stored outside the secure module.

Operational form

Keys encrypted under the master key are in operational form. They are directly usable for cryptographic operations by the Coprocessor. Operational keys are also called internal keys. All keys that are stored in the server key store file are operational keys. However, you do not need to store all operational keys in the key store file.

Export form

Keys encrypted under an exporter key-encrypting key as the result of an export operation are in export form. These keys are also called external keys. A key in export form can also be described as being in import form if an importer key-encrypting key with the same clear key value as the exporter key-encrypting key is present. You may store keys in export form in any manner you choose except in key store files.

Import form

Keys encrypted under an importer key-encrypting key are in import form. Only keys in import form can be used as the source for an import operation. These keys are also called external keys. A key in import form can also be described as being in export form if an exporter key-encrypting key with the same clear key value as the importer key-encrypting key is present. You may store keys in import form in any manner you choose except in key store files.

Function control vector

IBM provides a digitally signed value known as a function control vector. This value enables the cryptographic application within the Coprocessor to yield a level of cryptographic service consistent with applicable import regulations and export regulations. The function control vector is shipped with the IBM Cryptographic Access Provider (5722-ACx) product you install on your

system. The path name of the file is /QIBM/ProdData/CAP/FCV.CRT. The function control vector provides your Coprocessor with the key length information necessary to create keys.

Control vectors

A control vector, different from a function control vector, is a known value associated with a key that governs the following:

- Key type
- What other keys this key can encrypt
- Whether your Coprocessor can export this key
- Other allowed uses for this key

The control vector is cryptographically linked to a key and can not be changed without changing the value of the key at the same time.

Key store file

An i5/OS database file that is used to store keys which you encrypted under the master key of the Coprocessor.

Key token

A data structure that can contain a cryptographic key, a control vector, and other information related to the key. Key tokens are used as parameters on most of the CCA API verbs that either act on or use keys.

2058 Cryptographic Accelerator for iSeries

The 2058 Cryptographic Accelerator is available for customers to use with a V5R2 (or later) iSeries server. The 2058 Cryptographic Accelerator provides a competitive option to customers who do not require the high security of a Cryptographic Coprocessor, but do need the high cryptographic performance that hardware acceleration provides to offload a host processor. The 2058 Cryptographic Accelerator has been designed to improve the performance of those SSL applications that do not require secure key storage.

New for V5R3, you can also use the 2058 Cryptographic Accelerator to offload processing for DES, Triple DES, SHA-1, and RSA encryption methods, when using the new V5R3 Cryptographic Services API's. The 2058 Cryptographic Accelerator does not provide tamper-resistant storage for keys, like the Cryptographic Coprocessor hardware. Depending on the model of iSeries server you have, you can install up to a maximum of eight Cryptographic Accelerators. You can install a maximum of four Cryptographic Accelerators per partition.

The 2058 Cryptographic Accelerator provides special hardware which is optimized for RSA encryption (modular exponentiation) with data key lengths up to 2048 bits. It also provides functions for DES, TDES, and SHA-1 encryption methods. The 2058 Accelerator uses multiple RSA (Rivest, Shamir and Adleman algorithm) engines. Refer to the iSeries Performance Management web site for performance information specific to your iSeries server model.

For more information about the 2058 Cryptographic Accelerator, refer to the following pages:

- 2058 Cryptographic Accelerator features
- Cryptographic hardware scenario: Enhance iSeries SSL performance
- Plan for the 2058 Cryptographic Accelerator
- Configure the 2058 Cryptographic Accelerator

Features

Some features of the 2058 Cryptographic Accelerator include:

- Single card high performance cryptographic adapter (standard PCI card)
- Designed and optimized for RSA encryption
- On-board hardware-based RNG (random number generator)

- Five mounted IBM UltraCypher Cryptographic Engines

See the following information regarding the 2058 Cryptographic Accelerator:

- Plan for the 2058 Cryptographic Accelerator
- Configure the 2058 Cryptographic Accelerator

Scenario: Enhance iSeries SSL performance

To give you an idea of how you can use this cryptographic hardware with your iSeries server, we have added this usage scenario.

Situation

A company's iSeries server handles thousands of secured Internet transactions per day. The company's transactions utilize the Secure Sockets layer and Transport Layer Security protocols (SSL and TLS) – a common method for securing Internet transactions. This company's system administrator, Sue, wants to free up server resources for additional application processing, including the ability to support even more SSL transactions. Sue is looking for a solution that fits these objectives:

- A sizeable increase in the available server resources for application processing, including additional SSL transactions
- Minimal installation and configuration effort
- Minimal resource management requirements

Based on these objectives, Sue orders and installs an IBM 2058 e-Business Cryptographic Accelerator. (hereafter referred to as a 2058 Cryptographic Accelerator). The 2058 Cryptographic Accelerator is a PCI (Peripheral Component Interconnect) card, which is specially designed to accelerate the very compute intensive processing required when establishing a SSL/TLS session. On iSeries servers the 2058 Cryptographic Accelerator can be obtained by ordering hardware feature code 4805.

Details

1. The iSeries server has a 2058 Cryptographic Accelerator installed and configured.
2. The iSeries server receives a high number of SSL transaction requests from the network.
3. The 2058 Cryptographic Accelerator performs the cryptographic processing in the initiation of SSL transactions, and caches the private keys that are associated with the digital certificates for SSL transactions.

Prerequisites and assumptions

This scenario assumes that Sue has planned for the installation of the 2058 Cryptographic Accelerator, and then configured the card properly (see Plan for the 2058 Cryptographic Accelerator, and Configure the 2058 Cryptographic Accelerator). This scenario also assumes that Sue has already set up a digital certificate for SSL.

Configuration steps

Sue completes the following steps to enhance the SSL performance of her company's iSeries server:

1. Order Hardware Feature code 4805, which provides the 2058 Cryptographic Accelerator.
2. Install the 2058 Cryptographic Accelerator.
3. Create a device description for the 2058 Cryptographic Accelerator, and vary-on the device (see Configure the 2058 Cryptographic Accelerator for details).

Plan for the 2058 Cryptographic Accelerator

2058 Cryptographic Accelerator: Depending on the model of iSeries server you have, you can install up to a maximum of eight Cryptographic Accelerators. You can install a maximum of four Cryptographic Accelerators per partition. Your server must meet these requirements before you install and use the 2058 Cryptographic Accelerator.

Hardware requirements

The IBM e-Business Cryptographic Accelerator (orderable feature code 4805, and hereafter referred to as the 2058 Cryptographic Accelerator). The 4805 feature is a standard PCI card, and is supported on the following server models:

- | • eServer i5 520, 550, 570, and 595
- | • eServer i5 270, 810, 820, 825, 830, 840, 870, and 890
- | • eServer i5 expansion units 5074, 5075, 5078, 5079, 5088, 5094, 5095, 5294, and 5790

i5/OS and SSL requirements

The 2058 Cryptographic Accelerator requires OS/400 V5R2M0 (Version 5 Release 2 Modification 0) software, or subsequent i5/OS software. Although the 2058 Cryptographic Accelerator is fully enabled for cryptographic operations, the Cryptographic Access Provider 128-bit (5722-AC3) licensed program product must also be installed on the server to enable the cryptographic functions in the software that SSL also uses.

Configure the 2058 Cryptographic Accelerator

You must create a device description so that the operating system's SSL can direct RSA cryptographic operations to the 2058 Cryptographic Accelerator. You can create a device description by using the Create Device Crypto CL command.

Create device description

To create a device description using the CL command, follow these steps:

1. Type CRTDEVCRP at the command line.
2. Specify a name for the device as prompted.
3. Accept the default name of the PKA key store: *NONE.
4. Accept the name default of the DES key store: *NONE.
5. Specify an APPTYPE of *NONE.
6. Specify a description as prompted. This is optional.
7. Use either the Vary Configuration (VRYCFG) or the Work with Configuration Status (WRKCFGSTS) CL commands to vary on the device once you have created the device description.

For digital certificates that are generated by software, and stored in software, the operating system's SSL automatically starts using the 2058 Cryptographic Accelerator once the device is varied-on. The private key processing associated with SSL and TLS session establishment is off-loaded to the 2058 Cryptographic Accelerator. When the device is varied-off, the operating system's SSL switches back to software based encryption for establishing SSL and TLS sessions, thereby placing the private key processing load back on the server.

Note: This is only true for certificates and private keys that were not created by the Cryptographic Coprocessor. If a certificate was generated using the Cryptographic Coprocessor, the Cryptographic Coprocessor has to be used for those SSL or TLS sessions which use that particular certificate.

The Cryptographic hardware scenario: Enhance SSL performance page offers a server usage scenario for the 2058 Cryptographic Accelerator, once it has been installed and varied on.

Cryptographic Coprocessors

The IBM 4764 Cryptographic Coprocessor is available on eServer i5 models as hardware feature code 4806. Depending on the model of server you have, the following table shows the maximum number of Cryptographic Coprocessors supported:

Table 1. Supported number of 4764 Cryptographic Coprocessors

| Server models | Maximum per server | Maximum per partition |
|--------------------------------------|--------------------|-----------------------|
| eServer i5 Models 570 8/12/16W, 595 | 32 | 8 |
| eServer i5 Models 520, 550, 570 2/4W | 8 | 8 |

The IBM 4758-023 Cryptographic Coprocessor is available on eServer i5 servers as hardware feature code 4801. Depending on the model of server you have, the following number of Cryptographic Coprocessors are supported:

Table 2. Supported number of 4758 Cryptographic Coprocessors

| Server models | Maximum per server | Maximum per partition |
|---|--------------------|-----------------------|
| eServer Models 840, 870, 890 and eServer i5 Models 570 8/12/16W, 595 | 32 | 8 |
| eServer Models 810, 820, 825, 830 and eServer i5 Models 520, 550, 570 2/4W | 8 | 8 |
| eServer i5 Model 800 | 4 | 4 |
| eServer i5 Model 270 | 3 | 3 |

The Cryptographic Coprocessors can be used to augment your server in the following ways:

- You can use a Cryptographic Coprocessor to implement a broad range of i5/OS based applications. Examples are applications for performing financial PIN transactions, bank-to-clearing-house transactions, EMV transactions for integrated circuit (chip) based credit cards, and basic SET block processing. To do this, you or an applications provider must write an application program, using a security programming interface (SAPI) to access the security services of your Cryptographic Coprocessor. The SAPI for the Cryptographic Coprocessor conforms to IBM's Common Cryptographic Architecture (CCA). The SAPI is contained in the CCA Cryptographic Service Provider (CCA CSP) which is delivered as i5/OS Option 35.

To meet capacity and availability requirements, an application can control up to eight Coprocessors. The application must control access to individual Coprocessor by using the Cryptographic_Resource_Allocate (CSUACRA) and Cryptographic_Resource_Deallocate (CSUACRD) CCA APIs.

- You can use a Cryptographic Coprocessor along with DCM to generate and store private keys associated with SSL digital certificates. A Cryptographic Coprocessor provides a performance assist enhancement by handling SSL private key processing during SSL session establishment.
- When using multiple Coprocessors, DCM configuration gives you the following options for using hardware to generate and store the private key associated with a digital certificate.

1. Private key generated in hardware and stored (i.e., retained) in hardware.

With this option the private key never leaves the Coprocessor, and thus the private key cannot be used or shared with another Coprocessor. This means that you and your application have to manage multiple private keys and certificates.

2. Private key generated in hardware and stored in software (i.e., stored in a key store file).

This option allows a single private key to be shared amongst multiple Coprocessors. A requirement is that each Coprocessor must share the same master key—you can use "Clone master keys" on page 185 to set up your Coprocessors to have the same master key. The private key is generated in

one of the Coprocessors and is then saved in the key store file, encrypted under the master key of that Coprocessor. Any Coprocessor with an identical master key can use that private key.

See “Manage multiple Cryptographic Coprocessors” on page 175 for more information regarding the management of multiple cryptographic coprocessors.

For more information about Cryptographic Coprocessors, refer to the following pages:

“Features”

Cryptographic Coprocessors contain hardware engines, which perform cryptographic operations used by i5/OS application programs and i5/OS SSL transactions. Each IBM Cryptographic Coprocessor contains a tamper-resistant hardware security module (HSM) which provides secure storage for store master keys. The HSM is designed to meet FIPS 140 security requirements. To meet your capacity and high availability needs, multiple Cryptographic Coprocessors are supported.

The features page describes in greater detail what the Cryptographic Coprocessors and CCA CSP have to offer.

“Requirements” on page 16

Your server must meet some requirements before you can install and use a Cryptographic Coprocessor. Use the requirements page to determine whether you are ready to install and use a Cryptographic Coprocessor on your server.

“Cryptography concepts” on page 3

Depending on your familiarity with cryptography, you may need more information about a term or concept. This page introduces you to some basic cryptographic concepts.

See Related information for additional sources of cryptography information recommended by IBM.

Features

| IBM 4758 and 4764 Cryptographic Coprocessors

| Cryptographic Coprocessors provide cryptographic processing capability and a means to securely store
| cryptographic keys. Cryptographic functions supported include encryption for keeping data confidential,
| message digests and message authentication codes for ensuring that data has not been changed, and
| digital signature generation and verification. In addition, the Coprocessors provide a rich set of basic
| services for financial PIN, EMV, and SET applications.

| The main benefit of IBM Cryptographic Coprocessors is that they provide a secure environment for
| executing cryptographic functions and managing cryptographic keys. Master keys are stored in a battery
| backed-up, tamper-resistant hardware security module (HSM). The HSM is designed to meet Federal
| Information Processing Standard (FIPS) PUB 140 security requirements.

| You can use the Coprocessors with i5/OS SSL or with i5/OS application programs written by you or an
| application provider. The 4764 Cryptographic Coprocessor offers improved performance over that of the
| 4758 Cryptographic Coprocessor.

| SSL application features

| Establishment of secure sockets layer (SSL), also called transport layer security (TLS), sessions requires
| computationally-intensive cryptographic processing. The 4764 Cryptographic Coprocessor when used
| with i5/OS SSL can off-load this intensive cryptographic processing and free the server CPU for

| application processing. Another reason to use the Coprocessor is the hardware-based protection it provides for the private key that is associated with the server's SSL digital certificate.

| When configured with SSL, your Coprocessor can be used to create and store a private key in the FIPS 140 certified HSM. Or the Coprocessor can be used to create a private key, encrypt it with the master key – all performed within the HSM – and then store the encrypted private key via system software in a key store file. This enables a given private key to be used by multiple Coprocessor cards. Master keys are always stored in the FIPS 140 certified hardware module.

| **i5/OS CCA application features**

| You can use your Cryptographic Coprocessor to provide a high-level of cryptographic security for your applications. To implement i5/OS applications using the facilities of a Cryptographic Coprocessor you or an applications provider must write an application program using a security application programming interface (SAPI) to access the security services of your Cryptographic Coprocessor. The SAPI for the Cryptographic Coprocessor conforms to IBM's Common Cryptographic Architecture (CCA) and is supplied by i5/OS Option 35 CCA Cryptographic Service Provider (CCA CSP).

| With i5/OS the Cryptographic Coprocessor SAPI supports application software that is written in ILE C, RPG, and Cobol. Application software via the SAPI can call on CCA services to perform a wide range of cryptographic functions, including Triple-Data Encryption Standard (T-DES), RSA, MD5, SHA-1, and RIPEMD-160 algorithms. Basic services supporting financial PIN, EMV2000 (Europay, MasterCard, Visa) standard, and SET (Secure Electronic Transaction) block processing are also available. In support of an optional layer of security the Cryptographic Coprocessor provides a role-based access control facility, which allows you to enable and control access to individual cryptographic operations that are supported by the Coprocessor. The role-based access controls define the level of access that you give to your users.

| The SAPI is also used to access the key management functions of the Coprocessor. Key-encrypting keys and data encryption keys can be defined. These keys are generated in the Cryptographic Coprocessor and encrypted under the master key so that you can store these encrypted keys outside of your Coprocessor. You store these encrypted keys in a key store file, which is an i5/OS database file. Additional key management functions include the following:

- | • Create keys using cryptographically secure random-number generator
- | • Import and export encrypted T-DES and RSA keys securely
- | • Clone a master key securely

| Multiple Cryptographic Coprocessor cards can be used to meet your performance capacity and/or high-availability requirements. See "Manage multiple Cryptographic Coprocessors" on page 175 for more information.

| Security APIs for the 4758/4764 Cryptographic Coprocessors are documented in the following manuals:

- | • For the 4764 Cryptographic Coprocessor you can find these APIs described in the IBM PCI Cryptographic Coprocessor CCA Basic Services Reference and Guide, Release 3.20.
- | • For the 4758 Cryptographic Coprocessor you can find these APIs described in the IBM PCI Cryptographic Coprocessor CCA Basic Services Reference and Guide, Release 2.54.

| Go to the IBM PCI Cryptographic Coprocessor documentation library  to find these publications.

Cryptographic Coprocessor scenarios

To give you some ideas of how you can use this cryptographic hardware with your server, we have added the following usage scenarios:

- Cryptographic hardware scenario: Protect private keys with cryptographic hardware
This scenario might be useful for a company that needs to increase the security of the server digital certificate private keys that are associated with the SSL-secured business transactions.

- Cryptographic hardware scenario: Write an i5/OS application to use the Cryptographic Coprocessor
This scenario could help an i5/OS or OS/400 programmer reason through the process of writing a program that calls the Cryptographic Coprocessor to verify user data such as financial personal identification numbers (PINs), which are entered at automatic teller machines (ATMs).

Scenario: Protect private keys with cryptographic hardware

Situation

A company has a server dedicated to handling business-to-business (B2B) transactions. This company's server specialist, Sam, has been informed by management of a security requirement from its B2B customers. The requirement is to increase the security of the server digital certificate private keys that are associated with the SSL-secured business transactions that Sam's company performs. Sam has heard that there is a cryptographic hardware option available for servers that both encrypts and stores private keys associated with SSL transactions in tamper-responding hardware: a Cryptographic Coprocessor card.

Sam researches the Cryptographic Coprocessor, and learns that he can use it with the i5/OS Digital Certificate Manager (DCM) to provide secure SSL private key storage, as well as increase server performance by off-loading from the server the cryptographic operations which are completed during SSL-session establishment.

Note: To support load balancing and performance scaling, Sam can use multiple Cryptographic Coprocessors with SSL on the server. See Implementing multiple cryptographic hardware cards on the iSeries for more information.

Sam decides that the Cryptographic Coprocessor meets his company's requirement to increase the security of his company's server.

Details

1. The company's server has a Cryptographic Coprocessor installed and configured to store and protect private keys.
2. Private keys are generated by the Cryptographic Coprocessor.
3. Private keys are then stored on the Cryptographic Coprocessor.
4. The Cryptographic Coprocessor resists both physical and electronic hacking attempts.

Prerequisites and assumptions

1. The server has a Cryptographic Coprocessor installed and configured properly (see Plan for the Cryptographic Coprocessor, and Configure the Cryptographic Coprocessor). Planning for the Cryptographic Coprocessor includes getting SSL running on the server.

Note: To use multiple Cryptographic Coprocessor cards for application SSL handshake processing, and securing private keys, Sam will need to ensure that his application can manage multiple private keys and certificates.

2. Sam's company has Digital Certificate Manager (DCM) installed and configured, and uses it to Manage public Internet certificates for SSL communications sessions.
3. Sam's company obtain certificates from a public Certificate Authority (CA).
4. The Cryptographic Coprocessor is varied on prior to using DCM. Otherwise, DCM will not provide a page for selecting a storage option as part of the certificate creation process.

Configuration steps

Sam needs to perform the following steps to secure private keys with cryptographic hardware on his company's server:

1. Ensure that the prerequisites and assumptions for this scenario have been met.
2. Use the IBM Digital Certificate Manager (DCM) to create a new digital certificate, or renew a current digital certificate:
 - a. Select the type of certificate authority (CA) that is signing the current certificate.
 - b. Select the **Hardware** as your storage option for certificate's private key.
 - c. Select which cryptographic hardware device you want to store the certificate's private key on.
 - d. Select a public CA to use.

The private key associated with the new digital certificate is now stored on the Cryptographic Coprocessor specified in Step 2.c. Sam can now go into the configuration for his company's web server and specify that the newly created certificate be used. Once he restarts the web server, it will be using the new certificate.

Scenario: Write an i5/OS or OS/400 application to use the Cryptographic Coprocessor

Situation

Suppose you are a programmer for a large financial Credit Union. You have been assigned the task of getting a Cryptographic Coprocessor PCI card that is installed in the Credit Union server to verify members' financial personal identification numbers (PINs) when they are entered at automatic teller machines (ATMs).

You decide to write a server application program using the CCA CSP (cryptographic service provider) APIs that are a part of Option 35 to access the cryptographic services in the Cryptographic Coprocessors to verify members' PINs. Server application programs written for the Cryptographic Coprocessor utilize the coprocessor to perform security-sensitive tasks and cryptographic operations.


Note: Multiple Cryptographic Coprocessors can be used via the CCA CSP. The application must control access to individual Coprocessor by using the `Cryptographic_Resource_Allocate` (CSUACRA) and `Cryptographic_Resource_Deallocate` (CSUACRD) CCA APIs.

Details

1. A Credit Union member enters his or her PIN at an ATM.
2. The PIN is encrypted at the ATM, and then sent along the network to the Credit Union's server.
3. The server recognizes the transaction request, and calls a program to verify the member's PIN.
4. The program sends a request containing the encrypted PIN, member's account number, PIN-generating key, and PIN encrypting key to the Cryptographic Coprocessor.
5. The Cryptographic Coprocessor confirms or denies the validity of the PIN.
6. The program sends the Cryptographic Coprocessor's results to the ATM.
 - a. If the PIN is confirmed, the member can successfully complete a transaction with the Credit Union.
 - b. If the PIN is denied, the member is unable to complete a transaction with the Credit Union.

Prerequisites and assumptions

1. Your company has a server with a properly installed and configured Cryptographic Coprocessor. Refer to the following information:
 - a. Plan for the Cryptographic Coprocessor
 - b. Configure the Cryptographic Coprocessor

- c. Configure the Cryptographic Coprocessor for use with i5/OS or OS/400 applications
2. You are familiar with Option 35: The Common Cryptographic Architecture Cryptographic Service Provider (CCA CSP). It is packaged as i5/OS or OS/400 Option 35, and provides a security application programming interface (SAPI) to which you can write applications that allow you to access the cryptographic services of the Cryptographic Coprocessor.
3. You have access to the CCA Basic Services Guide , where you can find Financial Services Support verbs to use in your application.

Configuration steps

One way to accomplish your objective of using the Cryptographic Coprocessor to validate PINs is to write two server applications:

1. Write a program that loads the both the PIN verification keys, and PIN encrypting keys, and stores them in a key store file. Assuming that clear key parts are used, you need to use the following APIs:
 - Logon_Control (CSUALCT)
 - Key_Part_Import (CSNBKPI)
 - Key-Token_Build (CSNBKTB)
 - Key_Record_Create (CSNBKRC)
 - Key_Record_Write (CSNBKRW)
 - Optional API: KeyStore_Designate (CSUAKSD)
2. Write a second program that calls the Encrypted_PIN_Verify (CSNBPVR) API to verify encrypted PINs, and then reports their valid or invalid status back to the ATM.

Related page: [Configure the Cryptographic Coprocessor for use with i5/OS or OS/400 applications](#)

Plan for the Cryptographic Coprocessor

The following information is pertinent to those planning to install a Cryptographic Coprocessor in a server:

- Requirements for the Cryptographic Coprocessor
- Secure access to the Cryptographic Coprocessor
- Object authorities that are required for SAPI

| Requirements

| Your server must meet these requirements before you install and use any of the Cryptographic Coprocessors.

| 4764 Cryptographic Coprocessor: Hardware requirements

| The 4764 Cryptographic Coprocessor can be ordered by specifying a Hardware Feature code 4806, which is supported on the following iSeries i5 server models:

- | • iSeries i5 520, 550, 570, and 595
- | • I/O Expansion units 5074, 5088, 5095, 0595, 5294, and 5790

| Your Cryptographic Coprocessor is a PCI card. Install the card as described in the eServer PCI adapter installation manual. See the PCI Adapter  PDF for more information.

| Your Cryptographic Coprocessor requires the following software:


- | • i5/OS (5722-SS1): The 4764 Cryptographic Coprocessor requires i5/OS Version 5 Release 3 Modification 0.
- | • i5/OS Option 35 Common Cryptographic Architecture Cryptographic Service Provider (CCA CSP) provides the SAPI.

- | • i5/OS 5722–CY1 Cryptographic Device Manager provides the CCA control code for the 4764 Cryptographic Coprocessor.
- | • i5/OS Option 34 Digital Certificate Manager (if you are planning on using the Cryptographic Coprocessor configuration web-based utility)
- | • i5/OS 57xx–TC1 TCP/IP Connectivity Utilities (if you are planning on using the Cryptographic Coprocessor configuration web-based utility)
- | • i5/OS 57xx–DG1 IBM HTTP Server (if you are planning on using the Cryptographic Coprocessor configuration web-based utility)
- | • i5/OS 5722–AC3 Cryptographic Access Provider 128-bit must be installed on your server to enable the encryption capabilities of the Cryptographic Coprocessor. This option enables your Cryptographic Coprocessor to use 56-bit DES keys, 112 bit Triple DES keys, and 2048-bit RSA keys.

| **4758 Cryptographic Coprocessor: Hardware requirements**

| The 4758 Cryptographic Coprocessor can be ordered by specifying a Hardware Feature code 4801, which is supported on the following server models:

- | • iSeries i5 520, 550, 570, and 595
- | • iSeries 250 and 270 (250 requires the 7102 expansion unit), 810, 820, 825, 830, 840, 870, and 890
- | • Expansion units 5074, 5075, 5078, 5079, 5088, 5094, 5095, 5294, and 5790

| Your Cryptographic Coprocessor is a PCI card. Install the card as described in the eServer PCI adapter installation manual. See the PCI Adapter  PDF for more information.

| Your Cryptographic Coprocessor requires the following software:

- | • i5/OS: The 4758 Cryptographic Coprocessors requires i5/OS Version 5 Release 3 Modification 0 or OS/400 Version 4 Release 5 Modification 0 or later.
- | • i5/OS or OS/400 Option 35 Common Cryptographic Architecture Cryptographic Service Provider (CCA CSP)
- | • i5/OS or OS/400 Option 34 Digital Certificate Manager (if you are planning on using the Cryptographic Coprocessor configuration web-based utility)
- | • i5/OS or OS/400 57xx–TC1 TCP/IP Connectivity Utilities (if you are planning on using the Cryptographic Coprocessor configuration web-based utility)
- | • i5/OS or OS/400 57xx–DG1 IBM HTTP Server (if you are planning on using the Cryptographic Coprocessor configuration web-based utility)
- | • The Cryptographic Access Provider 128-bit for iSeries (5722-AC3) licensed program product must be installed on your server to enable the encryption capabilities of the Cryptographic Coprocessor. This option enables your Cryptographic Coprocessor to use 56-bit DES keys, 112 bit Triple DES keys, and 2048-bit RSA keys.

| Software Notes:

- | 1. The Cryptographic Coprocessor configuration web-based utility debuted with OS/400 Version 5 Release 1 Modification 0.
- | 2. You may have a previous version of a Cryptographic Access Provider (e.g. 5769–AC1, 5769–AC2, 5769–AC3) installed. These products are compatible with Version 5 Release 2 Modification 0 and later versions.

| Hardware Note: The Cryptographic Coprocessors destroy their factory certification if allowed to cool below -15 degrees C (5 degrees F). If your Coprocessor destroys its factory certification, you can no longer use the card. Contact your hardware service provider about ordering a new card.

Secure access

Access control restricts the availability of system resources to only those users you have authorized to interact with the resources. The server allows you to control authorization of users to system resources.

Your organization should identify each system resource in the organization's security hierarchy. The hierarchy should clearly delineate the levels of access authorization users have to resources.

All of the service programs in i5/OS or OS/400 Option 35 are shipped with *EXCLUDE authority for *PUBLIC. You must give users *USE authority for the service program that they need to use. In addition, you must also give users *USE authority to the QC6SRV service program in library QCCA.

Users who take part in setting up a Cryptographic Coprocessor must have *IOSYSCFG special authority to use the Master_Key_Process (CSNBMKP), Access_Control_Initialize (CSUAACI), or Cryptographic_Facility_Control (CSUACFC) security application programming interfaces (SAPIs). These three SAPIs are used to perform all configuration steps for the Cryptographic Coprocessors. For all SAPIs, users may require additional object authorities. Refer to "Object authorities that are required for SAPI."

For the most secure environments, consider assigning the role of Coprocessor Administrators to a set of users who do not have *ALLOBJ special authority. This way, users with *ALLOBJ special authority cannot alter the configuration of the Coprocessor because they will not be able to log on to an administrative role on the Coprocessor. They can, however, control object authority to the SAPI service programs, preventing misuse by the administrators.

In order to use the Cryptographic Coprocessor configuration web utility, users must have *SECADM special authority.

Cryptographic Coprocessors have separate access controls which are unrelated to the access controls of the server. The Cryptographic Coprocessor access controls allow you to control access to the Cryptographic Coprocessor hardware commands. For more information about these commands, see "Create and define roles and profiles" on page 23.

For even more security, limit the capabilities of the default role within your Cryptographic Coprocessor. Assign capabilities among other roles to require two or more people to perform security-sensitive functions, like changing the master key. You can do this when you work with roles and profiles as described in "Create and define roles and profiles" on page 23.

Note: You should consider some standard physical security measures as well, such as keeping your server behind a locked door.

Object authorities that are required for SAPI:

| SAPI | *USE for device | *USE for DES keystore | *CHANGE for DES keystore | *USE for DES Keystore Library | *USE for PKA keystore | *CHANGE for PKA keystore | *USE for PKA Keystore Library |
|---------|-----------------|-----------------------|--------------------------|-------------------------------|-----------------------|--------------------------|-------------------------------|
| CSNBCKI | Y | | Y ¹ | Y ¹ | | | |
| CSNBCKM | Y | | Y ² | Y | | | |
| CSNBCPA | Y | Y ¹ | | Y ¹ | | | |
| CSNBCPE | Y | Y ¹ | | Y ¹ | | | |
| CSNBCSG | Y | Y ¹ | | Y ¹ | | | |
| CSNBCSV | Y | Y ¹ | | Y ¹ | | | |
| CSNBCVE | Y | Y ¹ | | Y ¹ | | | |
| CSNBCVG | | | | | | | |
| CSNBCVT | Y | Y ¹ | | Y ¹ | | | |
| CSNBDEC | Y | Y ¹ | | Y ¹ | | | |

| SAPI | *USE for device | *USE for DES keystore | *CHANGE for DES keystore | *USE for DES Keystore Library | *USE for PKA keystore | *CHANGE for PKA keystore | *USE for PKA Keystore Library |
|-----------------------|-----------------|-----------------------|--------------------------|-------------------------------|-----------------------|--------------------------|-------------------------------|
| CSNBDKG | Y | | Y ¹ | Y ¹ | | | |
| CSNBDKM | Y | Y ² | Y ² | Y ¹ | | | |
| CSNBDKX | Y | Y ¹ | | Y ¹ | | | |
| CSNBENC | Y | Y ¹ | | Y ¹ | | | |
| CSNBEPG | Y | Y ¹ | | Y ¹ | | | |
| CSNBKEX | Y | Y ¹ | | Y ¹ | | | |
| CSNBKGN | Y | Y ² | Y ² | Y ¹ | | | |
| CSNBKPI | Y | | Y ¹ | Y ¹ | | | |
| CSNBKRC | Y | | Y | Y | | | |
| CSNBKRD | Y | | Y | Y | | | |
| CSNBKRL | Y | Y | | Y | | | |
| CSNBKRR | Y | Y | | Y | | | |
| CSNBKRW | Y | | Y | Y | | | |
| CSNBKSI | Y | | Y ³ | Y ³ | | Y ³ | Y ³ |
| CSNBKTC | Y | | Y ¹ | Y ¹ | | | |
| CSNBKTP | | | | | | | |
| CSNBKTR | Y | Y ¹ | | Y ¹ | | | |
| CSNBKYT | Y | Y ¹ | | Y ¹ | | | |
| CSNBKYTX ⁴ | Y | Y ¹ | | Y ¹ | | | |
| CSNBMDG | Y | | | | | | |
| CSNBMGN | Y | Y ¹ | | Y ¹ | | | |
| CSNBMKP | Y | | | | | | |
| CSNBOWH | | | | | | | |
| CSNBPCU | Y | Y ¹ | | Y ¹ | | | |
| CSNBPEX | Y | Y ¹ | | Y ¹ | | | |
| CSNBPEXX ⁴ | Y | Y ¹ | | Y ¹ | | | |
| CSNBPGN | Y | Y ¹ | | Y ¹ | | | |
| CSNBSPN | Y | Y ¹ | | Y ¹ | | | |
| CSNBPTR | Y | Y ¹ | | Y ¹ | | | |
| CSNBPVR | Y | Y ¹ | | Y ¹ | | | |
| CSNBSKY | Y | Y ¹ | | Y ¹ | | | |
| CSNBTRX | Y | Y | | Y | | | |
| CSNDDSG | Y | | | | Y ¹ | | Y ¹ |
| CSNDDSV | Y | | | | Y ¹ | | Y ¹ |
| CSNDKRC | | | | | | Y | Y |
| CSNDKRD | | | | | | Y | Y |
| CSNDKRL | | | | | Y | | Y |
| CSNDKRR | | | | | Y | | Y |
| CSNDKRW | | | | | | Y | Y |

| SAPI | *USE for device | *USE for DES keystore | *CHANGE for DES keystore | *USE for DES Keystore Library | *USE for PKA keystore | *CHANGE for PKA keystore | *USE for PKA Keystore Library |
|---------|-----------------|-----------------------|--------------------------|-------------------------------|-----------------------|--------------------------|-------------------------------|
| CSNDKTC | Y | | | | | Y ¹ | Y ¹ |
| CSNDPKB | | | | | | | |
| CSNDPKG | Y | Y ¹ | | | | Y ¹ | Y ¹ |
| CSNDPKH | Y | | | | | | |
| CSNDPKI | Y | Y ¹ | | | | Y ¹ | Y ¹ |
| CSNDPKR | Y | | | | | | |
| CSNDPKX | Y | | | | Y ¹ | | Y ¹ |
| CSNDRKD | Y | | | | | | |
| CSNDRKL | Y | | | | | | |
| CSNDSBC | Y | | | | Y ¹ | | Y ¹ |
| CSNDSBD | Y | | | | Y ¹ | | Y ¹ |
| CSNDSYG | Y | | | | | Y ¹ | Y ¹ |
| CSNDSYI | Y | | Y ¹ | Y ¹ | Y ¹ | | Y ¹ |
| CSNDSYX | Y | | Y ¹ | Y ¹ | Y ¹ | | Y ¹ |
| CSUAACI | Y | | | | | | |
| CSUAACM | Y | | | | | | |
| CSUACFC | Y | | | | | | |
| CSUACFQ | Y | | | | | | |
| CSUACRA | Y | | | | | | |
| CSUACRD | Y | | | | | | |
| CSUAKSD | | | | | | | |
| CSUALCT | Y | | | | | | |
| CSUAMKD | Y | | | Y | | | |

¹Use of Data Encryption Standard (DES) or public key algorithm (PKA) keystore for this API is optional.

²More than one parameter may optionally use keystore. The authority requirements differ on each of those parameters.

³The Key_Store_Initialize SAPI does not require authority to both files simultaneously.

⁴These SAPIs pertain only to 4764 Cryptographic Coprocessors.

Configure the Cryptographic Coprocessor

Configuring your Cryptographic Coprocessor allows you to begin to use all of its cryptographic operations.

The easiest and fastest way to configure your Cryptographic Coprocessor is to use the Cryptographic Coprocessor configuration web-based utility found off of the server Tasks page at <http://server-name:2001> (specify another port if you have changed it from port 2001). The utility includes the Basic configuration wizard that is used for configuring (and initializing) a Coprocessor that has not been previously configured. If HTTP and SSL have not been previously configured, you will need to do the following before using the Configuration Wizard.

- Start the HTTP Administrative server.
- Configure the HTTP Administrative server to use SSL.
- Use DCM to create a certificate, specifying that the private key be generated and stored in software.
- Use DCM to receive the signed certificate.
- Associate the certificate with the HTTP Administrative server application ID.
- Restart the HTTP Administrative server to enable it for SSL processing.

If the Cryptographic Coprocessor has already been configured, then click on the **Manage configuration** option to change the configuration for specific portions of the Coprocessor.

If you would prefer to write your own application to configure the Coprocessor, you can do so by using the `Cryptographic_Facility_Control` (CSUACFC), `Access_Control_Initialize` (CSUAACI), `Master_Key_Process` (CSNBMKP), and `Key_Store_Initialize` (CSNBKSI) API verbs. Many of the pages in this section include one or more program examples that show how to configure the Coprocessor via an application. Change these programs to suit your specific needs.

Whether you choose to use the Cryptographic Coprocessor configuration utility or write your own applications, the following outlines the steps you must take to properly configure your Cryptographic Coprocessor:

1. “Create a device description.” The device description specifies a default location for key storage. You can create a device description with or without naming any key store files.
2. “Name files to key store file” on page 22. Before you can perform any operation using a key store file or key stored in a key store file, you must name the key store file. You can explicitly name key store files by using a program, or you can name them on the device description. You name one file to store Data Encryption Standard (DES) and Triple-DES keys and another file to store public key algorithm (PKA) keys. By naming the files in which to store your keys, you set up that database to contain your DES (and Triple-DES) and PKA keys. You should name key store files by using a program if you want to keep your keys in your own key store file. If you do not name key store files with a program, the CCA CSP will store your keys in the key store file named on the device description.
3. “Create and define roles and profiles” on page 23. When you assign users to these roles and profiles, you determine what cryptographic functions your Cryptographic Coprocessor will allow the users to use.
4. “Set the environment ID and clock” on page 60. Your Cryptographic Coprocessor uses the EID to verify which Coprocessor created a key token. It uses the clock for time and date stamping and to control whether a profile can log on.
5. “Load a function control vector” on page 70. The function control vector tells the Cryptographic Coprocessor what key length to use to create keys. You cannot perform any cryptographic functions without loading a function control vector.
6. “Load and set a master key” on page 82. After you load a function control vector, load and set the master key. You can use your master key to encrypt other keys.

Create a device description

You must create a device description for your Cryptographic Coprocessor on your server. The device description is used by CCA CSP to help direct cryptographic requests to the Coprocessor. Additionally, the device description gives your Coprocessor a default location for key store file storage. The Basic configuration wizard in the Cryptographic Coprocessor configuration utility, found off of the server Tasks page at <http://server-name:2001>, can create a device description for you, or you can create a device description yourself by using the **Create Device Crypto CL** command.

To create a device description using the Basic configuration wizard, follow these steps:

1. Point your web browser to the server Tasks page: <http://server-name:2001>
2. Click on Cryptographic Coprocessor configuration.
3. Click on the button labeled **Start secure session**.

4. Click **Basic configuration** wizard.
5. Click **continue** on the **Welcome** page.
6. Click on the list entry with the device name set to *CREATE for the resource you want to use.
7. Continue as instructed by the Basic configuration wizard.

To create a device description using the CL command, follow these steps:

1. Type CRTDEVCRP at the CL command line.
2. Specify a name for the device as prompted. If you want to set up a default device, name the device CRP01. Otherwise, each application you create must use the Cryptographic Resource Allocate (CSUACRA) API in order to access your device description.
3. Specify the name of a default PKA key store file or let the parameter default to *NONE.
4. Specify the name of a default DES key store file or let the parameter default to *NONE.
5. Specify a description as prompted. This is optional.
6. Use either the Vary Configuration (VRYCFG) or the Work with Configuration Status (WRKCFGSTS) CL commands to vary on the device once you have created the device description.
7. This typically takes one minute, but it may take ten minutes to complete.

Note: The APPTYPE defaults to *CCA, so you do not need to specify it on the **Create** command. However, if you have changed it to another value, you need to change it back to *CCA before the device can vary on.

You have now completed creation of the device description.

Name files to key store file

Before you can perform any operation using a key store file or a key stored in a key store file, you must name the key file. This points your Cryptographic Coprocessor to the correct file. You can name two types of key store files. One type stores Data Encryption Standard (DES) keys and Triple-DES keys. DES and Triple DES are symmetric cryptographic algorithms; the Cryptographic Coprocessor uses the same key to encrypt and decrypt. The other type stores public key algorithm (PKA) keys. Public key algorithms are asymmetric; keys are created in pairs. Cryptographic Coprocessors use one key to encrypt and the other to decrypt. Cryptographic Coprocessors support the RSA public key algorithm.

You can name a key store file explicitly by using a program, or you can name it by configuring it on the device description. To name a key store file from a program, use the Key_Store_Designate (CSUKSD) security application programming interface (SAPI). If you name key store files that use a program, your Cryptographic Coprocessor only uses the names for the job that ran the program. However, by naming key store files explicitly in your program, you can use separate key store files from other users. If you name key store files on the device description, you do not have to name them in your program. This may help if you are trying to maintain the same program source across multiple IBM platforms. It is also useful if you are porting a program from another implementation of Common Cryptographic Architecture.

You need to store your cryptographic keys in a secure form so that you can use them over time and exchange them with other users and servers, as appropriate. You can store your cryptographic keys by using your own methods, or you can store them in a key store file. You can have as many key store files as you want, and you can create multiple key store files for each type of key. You can place as many cryptographic keys in your key store files as you want.

Since each key store file is a separate server object, you can authorize different users to each file. You can save and restore each key store file at different times. This depends on how often the file's data changes or which data it is protecting.

Create and define roles and profiles

Cryptographic Coprocessors use role-based access control. In a role-based system, you define a set of roles, which correspond to the classes of Coprocessor users. You can enroll each user by defining an associated user profile to map the user to one of the available roles.

The capabilities of a role are dependent on the access control points or cryptographic hardware commands that are enabled for that role. You can then use your Cryptographic Coprocessor to create profiles that are based on the role you choose.

A role-based system is more efficient than one in which the authority is assigned individually for each user. In general, you can separate the users into just a few different categories of access rights. The use of roles allows you to define each of these categories just once, in the form of a role.

The role-based access control system and the grouping of permissible commands that you can use are designed to support a variety of security policies. In particular, you can set up Cryptographic Coprocessors to enforce a dual-control, split-knowledge policy. Under this policy no one person should be able to cause detrimental actions other than a denial-of-service attack, once the Cryptographic Coprocessor is fully activated. To implement this policy, and many other approaches, you need to limit your use of certain commands. As you design your application, consider the commands you must enable or restrict in the access-control system and the implications to your security policy.

Every Cryptographic Coprocessor must have a role called the default role. Any user that has not logged on to the Cryptographic Coprocessor will operate with the capabilities defined in the default role. Users who only need the capabilities defined in the default role do not need a profile. In most applications, the majority of the users will operate under the default role, and will not have user profiles. Typically, only security officers and other special users need profiles.

When Cryptographic Coprocessors are in an un-initialized state, the default role has the following access control points enabled:

- PKA96 One Way Hash
- Set Clock
- Re-initialize Device
- Initialize access control system roles and profiles
- Change the expiration data in a user profile
- Reset the logon failure count in a user profile
- Read public access control information
- Delete a user profile
- Delete a role

The default role is initially defined such that the functions permitted are those functions that are related to access control initialization. This guarantees that the Cryptographic Coprocessor will be initialized before you do any useful cryptographic work. The requirement prevents security "accidents" in which someone might accidentally leave authority intact when you put the Coprocessor into service.

Defining roles

The easiest and fastest way to define new roles (and redefine the default role) is to use the Cryptographic Coprocessor configuration web-based utility found off of the server Tasks page at <http://server-name:2001>. The utility includes the Basic configuration wizard that is used when the Coprocessor is in an un-initialized state. The Basic configuration wizard can define either 1 or 3 administrative roles along with redefining the default role. If the Coprocessor already has been initialized, then click on **Manage configuration** and then click on **Roles** to define new roles or change or delete existing ones.

If you would prefer to write your own application to manage roles, you can do so by using the `Access_Control_Initialization (CSUAACI)` and `Access_Control_Maintenance (CSUAACM)` API verbs. To change the default role in your Coprocessor, specify "DEFAULT" encoded in ASCII into the proper parameter. You must pad this with one ASCII space character. Otherwise, there are no restrictions on the characters that you may use for role IDs or profile IDs. Four example programs are provided for your consideration. Two of them are written in ILE C, while the other two are written in ILE RPG. Both sets perform the same function.

- "Example: ILE C program for creating roles and profiles for your Coprocessor" on page 27
- "Example: ILE C program for enabling all access control points in the default role for your Coprocessor" on page 47
- "Example: ILE RPG program for creating roles or profiles for your Coprocessor" on page 38
- "Example: ILE RPG program for enabling all access control points in the default role for your Coprocessor" on page 52

Note: If you choose to use one of the program examples provided, change it to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

Defining profiles

After you create and define a role for your Coprocessor, you can create a profile to use under this role. A profile allows users to access specific functions for your Coprocessor that may not be enabled for the default role.

The easiest and fastest way to define new profiles is to use the Cryptographic Coprocessor configuration web-based utility, located on the server Tasks page at <http://server-name:2001>. The utility includes the Basic configuration wizard that is used when the Coprocessor is in an un-initialized state. The Basic configuration wizard can define either one or three administrative profiles. If the Coprocessor has already been initialized, click **Manage configuration—>Profiles** to define new profiles or change or delete existing ones.

If you want to write your own application to manage profiles, you can use the `Access_Control_Initialization (CSUAACI)` and `Access_Control_Maintenance (CSUAACM)` API verbs. Two example programs are provided for you:

- "Example: ILE C program for changing an existing profile for your Coprocessor" on page 55
- "Example: ILE RPG program for changing an existing profile for your Coprocessor" on page 57

Note: If you choose to use one of the program examples provided, change it to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

If you will be using the Coprocessor for SSL, the default role must at least be authorized to the following access control points:

- Digital Signature Generate
- Digital Signature Verify
- PKA Key Generate
- PKA Clone Key Generate
- RSA Encipher Clear Data
- RSA Decipher Clear Data
- Delete Retained Key
- List Retain Keys

The Basic configuration wizard in the Cryptographic Coprocessor configuration utility automatically redefines the default role such that it can be used for SSL without any changes.

To avoid security hazards, consider denying the following access control points (also called cryptographic hardware commands) for the default role, after you have set up all of the roles and profiles:

Note: You should enable only those access control points that are necessary for normal operations. At a maximum, you should only enable specifically required functions. To determine which access control points are required, refer to the CCA Basic Services Guide. Each API lists the access control points that are required for that API. If you do not need to use a particular API, consider disabling the access control points that are required for it.

- Load first part of Master Key
- Combine Master Key Parts
- Set Master Key
- Generate Random Master Key
- Clear New Master Key Register
- Clear Old Master Key Register
- Translate CV
- Set Clock

Attention: If you intend to disable the Set Clock access control point from the default role, ensure that the clock is set before you disable access. The clock is used by the Coprocessor when users try to log on. If the clock is set incorrectly, users can not log on.

- Re-initialize device
- Initialize access control system
- Change authentication data (for example, pass phrase)
- Reset password failure count
- Read Public Access Control Information
- Delete user profile
- Delete role
- Load Function Control Vector
- Clear Function Control Vector
- Force User Logoff
- Set EID
- Initialize Master Key Cloning Control
- Register Public Key Hash
- Register Public Key, with Cloning
- Register Public Key
- PKA Clone Key Generate (Access control point required for SSL)
- Clone-Information Obtain Parts 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
- Clone-Information Install Parts 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
- Delete retained key (Access control point required for SSL)
- List retained keys (Access control point required for SSL)
- Encipher Under Master Key
- Data Key Export
- Data Key Import
- Re-encipher to Master Key
- Re-encipher from Master Key

- Load First Key Part
- Combine Key Parts
- Add Key Part
- Complete Key part

For the most secure environment, consider locking the access-control system after initializing it. You can render the access-control system unchangeable by deleting any profile that would allow use of the Access Control Initialization or the Delete Role access control point. Without these access control points, further changes to any role are not possible. With authority to use either the Initialize Access Control or Delete Role access control points, one can delete the DEFAULT role.

Deleting the DEFAULT role will cause the automatic recreation of the initial DEFAULT role. The initial DEFAULT role permits setting up any capabilities. Users with access to these access control points have unlimited authority through manipulation of the access-control system. Before the Coprocessor is put into normal operation, the access-control setup can be audited through the use of the Access_Control_Maintenance (CSUAACM) and Cryptographic_Facility_Query (CSUACFQ) API verbs.

If for any reason the status response is not as anticipated, the Coprocessor should not be used for application purposes until it has been configured again to match your security policy. If a role contains permission to change a pass phrase, the pass phrase of any profile can be changed. You should consider if passphrase changing should be permitted and, if so, which role(s) should have this authority.

If any user reports an inability to log on, this should be reported to someone other than (or certainly in addition to) an individual with pass phrase changing permission. Consider defining roles so that dual-control is required for every security sensitive operation to protect against a malicious insider acting on his/her own. For example, consider splitting the following groups of access control points between two or more roles. It is recommended that one person should not be able to use all of the commands in the Master key group, because this could represent a security risk.

The Master key group consists of these access control points:

- Load 1st part of Master Key
- Combine Master Key Parts
- Set Master Key
- Generate Random Master Key
- Clear New Master Key Register
- Clear Old Master Key Register

By the same token, one person should not be authorized to all of the commands in the Cloning key group.

The Cloning key group consists of these access control points:

- Initialize Master Key Cloning Control
- Register Public Key Hash
- Register Public Key, with Cloning
- Register Public Key
- PKA Clone Key Generate
- Clone-Information Obtain Parts 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
- Clone-Information Install Parts 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

After you create and define a profile for your Coprocessor, you must load a function control vector for your Coprocessor as described in “Load a function control vector” on page 70. Without the function control vector, your Coprocessor cannot perform any cryptographic functions.

Example: ILE C program for creating roles and profiles for your Coprocessor: Change this program example to suit your needs for creating a role or a profile for your Coprocessor.

Note: Read the "Code disclaimer information" on page 276 for important legal information.

```

/*-----*/
/* CRTROLEPRF */
/* */
/* Sample program to create roles and profiles in the */
/* cryptographic adapter. */
/* */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 1999 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: */
/* none. */
/* */
/* Example: */
/* CALL PGM(CRTROLEPRF) */
/* */
/* Use these commands to compile this program on a server: */
/* CRTCMOD MODULE(CRTROLEPRF) SRCFILE(SAMPLE) */
/* CRTPGM PGM(CRTROLEPRF) MODULE(CRTROLEPRF) */
/* BNDSRVPGM(QCCA/CSUAACI QCCA/CSNBOWH) */
/* */
/* Note: Authority to the CSUAACI and CSNBOWH service programs */
/* in the QCCA library is assumed. */
/* */
/* The Common Cryptographic Architecture (CCA) verbs used are */
/* Access_Control_Initialization (CSUAACI) and */
/* One_Way_Hash (CSNBOWH). */
/* */
/* Note: This program assumes the device you want to use is */
/* already identified either by defaulting to the CRP01 */
/* device or has been explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/* */
/* Note: Before running this program, the clock in the must be */
/* set using Cryptographic_Facility_Control (CSUACFC) in order */
/* to be able to logon afterwards. */
/* */
/*-----*/

#include "csucincl.h" /* header file for CCA Cryptographic
                    Service Provider for iSeries */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main(int argc, char *argv[]) {

```

```

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR    -1
#define OK       0
#define WARNING  4

/*-----*/
/* Variables used for parameters on CCA APIs */
/*-----*/
long return_code;
long reason_code;
long exit_data_length;
char exit_data[2];
char rule_array[4][8];
long rule_array_count;
long verb_data1_length;
long verb_data2_length;
long hash_length;
long text_length;
char *text;
char chaining_vector[128];
long chaining_vector_length;

/*-----*/
/* Definitions for profiles */
/*-----*/
typedef struct
{
    char    version[2];          /* Profile structure version */
    short   length;             /* length of structure */
    char    comment[20];        /* Description */
    short   checksum;
    char    logon_failure_count;
    char    reserved;
    char    userid[8];          /* Name for this profile */
    char    role[8];            /* Role that profile uses */
    short   act_year;           /* Activation date - year */
    char    act_month;          /* Activation date - month */
    char    act_day;            /* Activation date - day */
    short   exp_year;           /* Expiration date - year */
    char    exp_month;          /* Expiration date - month */
    char    exp_day;            /* Expiration date - day */
    short   total_auth_data_length;
    short   field_type;
    short   auth_data_length_1;
    short   mechanism;          /* Authentication mechanism */
    short   strength;           /* Strength of mechanism */
    short   mech_exp_year;      /* Mechanism expiration - year */
    char    mech_exp_month;     /* Mech. expiration - month */
    char    mech_exp_day;       /* Mechansim expiration - day */
    char    attributes[4];
    char    auth_data[20];      /* Secret data */
} profile_T;

typedef struct
{
    long    number;             /* Number profiles in struct */
    long    reserved;
    profile_T profile[3];
} aggregate_profile;

aggregate_profile * verb_data1; /* Aggregate structure for */
/* defining profiles */

```

```

/*-----*/
/* Definitions for roles */
/*-----*/
/*-----*/
/* Default role - access control points list - */
/*           authorized to everything EXCEPT: */
/* 0x0018 - Load 1st part of Master Key */
/* 0x0019 - Combine Master Key Parts */
/* 0x001A - Set Master Key */
/* 0x0020 - Generate Random Master Key */
/* 0x0032 - Clear New Master Key Register */
/* 0x0033 - Clear Old Master Key Register */
/* 0x0053 - Load 1st part of PKA Master Key */
/* 0x0054 - Combine PKA Master Key Parts */
/* 0x0057 - Set PKA Master Key */
/* 0x0060 - Clear New PKA Master Key Register */
/* 0x0061 - Clear Old PKA Master Key Register */
/* 0x0110 - Set Clock */
/* 0x0111 - Reinitialize device */
/* 0x0112 - Initialize access control system */
/* 0x0113 - Change user profile expiration date */
/* 0x0114 - Change authentication data (eg. passphrase) */
/* 0x0115 - Reset password failure count */
/* 0x0116 - Read Public Access Control Information */
/* 0x0117 - Delete user profile */
/* 0x0118 - Delete role */
/* 0x0119 - Load Function Control Vector */
/* 0x011A - Clear Function Control Vector */
/* 0x011B - Force User Logoff */
/* 0x0200 - Register PKA Public Key Hash */
/* 0x0201 - Register PKA Public Key, with cloning */
/* 0x0202 - Register PKA Public Key */
/* 0x0203 - Delete Retained Key */
/* 0x0204 - PKA Clone Key Generate */
/* 0x0211 - 0x21F - Clone information - obtain 1-15 */
/*-----*/
/* For access control points 0x01 - 0x127 */
char default_bitmap[] =
    { 0x00, 0x03, 0xF0, 0x1D, 0x00, 0x00, 0x00, 0x00,
      0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x0A, 0x80, 0x00, 0x88, 0x2F, 0x71, 0x10,
      0x10, 0x04, 0x03, 0x31, 0x80, 0x00, 0x00, 0x00,
      0xFF, 0x7F, 0x40, 0x6B, 0x80};

/* For access control points 0x200 - 0x23F */
char default2_bitmap[] =
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE6, 0x0F };

/*-----*/
/* role #1 - authorized to same as default plus also */
/*           authorized to: */
/* 0x0018 - Load 1st part of Master Key */
/* 0x0020 - Generate Random Master Key */
/* 0x0032 - Clear New Master Key Register */
/* 0x0053 - Load 1st part of PKA Master Key */
/* 0x0060 - Clear New PKA Master Key Register */
/* 0x0119 - Load Function Control Vector */
/* 0x0201 - Register PKA Public Key, with cloning */
/* 0x0202 - Register PKA Public Key */
/* 0x0203 - Delete Retained Key */
/* 0x0204 - PKA Clone Key Generate */
/* 0x0211 - 0x215 - Clone information - obtain 1-5 */
/* 0x0221 - 0x225 - Clone information - install 1-5 */
/*-----*/
char role1_bitmap[] =
    { 0x00, 0x03, 0xF0, 0x9D, 0x80, 0x00, 0x20, 0x00,
      0x80, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x00,

```

```

        0x00, 0x0A, 0x80, 0x00, 0x88, 0x1F, 0x71, 0x10,
        0x10, 0x04, 0x03, 0x11, 0x80, 0x00, 0x00, 0x00,
        0xFF, 0x7F, 0x00, 0x4F, 0x80};
char role1_bitmap2[] =
    { 0x78, 0x00, 0x7C, 0x00, 0x7C, 0x00, 0xE6, 0x0F };

/*-----*/
/* role #2 - authorized to same as default plus also */
/*     authorized to: */
/* 0x0019 - Combine Master Key Parts */
/* 0x001A - Set Master Key */
/* 0x0033 - Clear Old Master Key Register */
/* 0x0054 - Combine PKA Master Key Parts */
/* 0x0057 - Set PKA Master Key */
/* 0x0061 - Clear Old Master Key Register */
/* 0x011A - Clear Function Control Vector */
/* 0x0200 - Register PKA Public Key Hash */
/* 0x0201 - Register PKA Public Key, with cloning */
/* 0x0203 - Delete Retained Key */
/* 0x0204 - PKA Clone Key Generate */
/* 0x0216 - 0x21A - Clone information - obtain 6-10 */
/* 0x0226 - 0x22A - Clone information - install 6-10 */
/*-----*/
char role2_bitmap[] =
    { 0x00, 0x03, 0xF0, 0x7D, 0x80, 0x00, 0x10, 0x00,
      0x80, 0x00, 0x09, 0x00, 0x40, 0x00, 0x00, 0x00,
      0x00, 0x0A, 0x80, 0x00, 0x88, 0x1F, 0x71, 0x10,
      0x10, 0x04, 0x03, 0x31, 0x80, 0x00, 0x00, 0x00,
      0xFF, 0x7F, 0x00, 0x2F, 0x80};
char role2_bitmap2[] =
    { 0xD8, 0x00, 0x03, 0xE0, 0x03, 0xE0, 0xE6, 0x0F };

/*-----*/
/* role #3 - authorized to same as default plus also */
/*     authorized to: */
/* 0x0110 - Set Clock */
/* 0x0111 - Reinitialize device */
/* 0x0112 - Initialize access control system */
/* 0x0113 - Change user profile expiration date */
/* 0x0114 - Change authentication data (eg. passphrase) */
/* 0x0115 - Reset password failure count */
/* 0x0116 - Read Public Access Control Information */
/* 0x0117 - Delete user profile */
/* 0x0118 - Delete role */
/* 0x011B - Force User Logoff */
/* 0x0200 - Register PKA Public Key Hash */
/* 0x0201 - Register PKA Public Key, with cloning */
/* 0x0203 - Delete Retained Key */
/* 0x0204 - PKA Clone Key Generate */
/* 0x021B - 0x21F - Clone information - obtain 11-15 */
/* 0x022B - 0x22F - Clone information - install 11-15 */
/*-----*/
char role3_bitmap[] =
    { 0x00, 0x03, 0xF0, 0x1D, 0x00, 0x00, 0x00, 0x00,
      0x80, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x00,
      0x00, 0x0A, 0x80, 0x00, 0x88, 0x1F, 0x71, 0x10,
      0x10, 0x04, 0x03, 0x31, 0x80, 0x00, 0x00, 0x00,
      0xFF, 0x7F, 0xFF, 0x9F, 0x80};
char role3_bitmap2[] =
    { 0xD8, 0x00, 0x00, 0x1F, 0x00, 0x1F, 0xE6, 0x0F };

/*-----*/
/* Structures for defining the access control points in a role */
/*-----*/
struct access_control_points_header
{
    short        number_segments;    /* Number of segments of */

```



```

                /* Copy header into verb data
                2 storage. */
memcpy(work_ptr, (void*)&aggregate_role_header,
        sizeof(aggregate_role_header));

                /* Adjust work pointer to point
                after header. */
work_ptr += sizeof(aggregate_role_header);

/*-----*/
/* Fill in the fields of the role definitions. */
/* Each role is version 1, has authentication strength of 0, */
/* has valid time from 12:00 Midnight (0) to 23:59 (x173B), */
/* is valid every day of the week. (xFE is 7 bits set), */
/* has one access control points segment that starts at bit 0 */
/* and goes to bit x11F, and has 20 spaces for a comment. */
/*-----*/
role_header.version[0] = 1;
role_header.version[1] = 0;
role_header.length = sizeof(role_header) +
    sizeof(access_control_points_header) +
    2 * sizeof(access_control_points_segment_header) +
    sizeof(default_bitmap) + sizeof(default2_bitmap);
role_header.checksum = 0;
role_header.reserved1 = 0;
role_header.auth_strength = 0;
role_header.lower_time = 0;
role_header.upper_time = 0x173B;
role_header.valid_days_of_week = 0xFE;
role_header.reserved2 = 0;
memset(role_header.comment, ' ', 20);

access_control_points_header.number_segments = 2;
access_control_points_header.reserved = 0;
access_control_points_segment_header.reserved = 0;

for (i=0; i<3; i++)
{
    switch (i) {
        /*-----*/
        /* Set name for ROLE1 */
        /*-----*/
        case 0:
            memcpy(role_header.role, "ROLE1 ", 8);
            bitmap1 = role1_bitmap;
            bitmap2 = role1_bitmap2;

            break;

            /*-----*/
            /* Set name for ROLE2 */
            /*-----*/
        case 1:
            memcpy(role_header.role, "ROLE2 ", 8);
            bitmap1 = role2_bitmap;
            bitmap2 = role2_bitmap2;
            break;

            /*-----*/
            /* Set name for ROLE3 */
            /*-----*/
        case 2:
            memcpy(role_header.role, "ROLE3 ", 8);
            bitmap1 = role3_bitmap;
            bitmap2 = role3_bitmap2;
    }
}

```

```

/*-----*/
/* Copy role header */
/*-----*/
memcpy(work_ptr, (void*)&role_header, sizeof(role_header));

/* Adjust work pointer to
point after role header. */
work_ptr += sizeof(role_header);

/*-----*/
/* Copy access control points header */
/*-----*/
memcpy(work_ptr,
(void *)&access_control_points_header,
sizeof(access_control_points_header));

/* Adjust work pointer to
point after header. */
work_ptr += sizeof(access_control_points_header);

/*-----*/
/* Copy access control points segment 1 */
/*-----*/
access_control_points_segment_header.start_bit = 0;
access_control_points_segment_header.end_bit = 0x127;
access_control_points_segment_header.number_bytes =
sizeof(default_bitmap);
memcpy(work_ptr,
(void *)&access_control_points_segment_header,
sizeof(access_control_points_segment_header));

/* Adjust work pointer to
point after header. */
work_ptr += sizeof(access_control_points_segment_header);

/*-----*/
/* Copy access control points segment 1 bitmap */
/*-----*/
memcpy(work_ptr, bitmap1, sizeof(default_bitmap));

/* Adjust work pointer to
point after bitmap. */
work_ptr += sizeof(default_bitmap);

/*-----*/
/* Copy access control points segment 2 */
/*-----*/
access_control_points_segment_header.start_bit = 0x200;
access_control_points_segment_header.end_bit = 0x23F;
access_control_points_segment_header.number_bytes =
sizeof(default2_bitmap);

memcpy(work_ptr,
(void *)&access_control_points_segment_header,
sizeof(access_control_points_segment_header));

/* Adjust work pointer to
point after header. */
work_ptr += sizeof(access_control_points_segment_header);

/*-----*/
/* Copy access control points segment 2 bitmap */
/*-----*/
memcpy(work_ptr, bitmap2, sizeof(default2_bitmap));

/* Adjust work pointer to
point after bitmap. */

```

```

    work_ptr += sizeof(default2_bitmap);
}

/*-----*/
/* Allocate storage for aggregate profile structure */
/*-----*/
verb_data1 = malloc(sizeof(aggregate_profile));

verb_data1->number = 3; /* Define 3 profiles */
verb_data1->reserved = 0;

/*-----*/
/* Each profile: */
/* will be version 1, */
/* have an activation date of 1/1/00, */
/* have an expiration date of 6/30/2005, */
/* use passphrase hashed with SHA1 for the mechanism (0x0001), */
/* will be renewable (attributes = 0x8000) */
/* and has 20 spaces for a comment */
/*-----*/
for (i=0; i<3; i++)
{
    verb_data1->profile[i].length = sizeof(profile_T);
    verb_data1->profile[i].version[0] = 1;
    verb_data1->profile[i].version[1] = 0;
    verb_data1->profile[i].checksum = 0;
    verb_data1->profile[i].logon_failure_count = 0;
    verb_data1->profile[i].reserved = 0;
    verb_data1->profile[i].act_year = 2000;
    verb_data1->profile[i].act_month = 1;
    verb_data1->profile[i].act_day = 1;
    verb_data1->profile[i].exp_year = 2005;
    verb_data1->profile[i].exp_month = 6;
    verb_data1->profile[i].exp_day = 30;
    verb_data1->profile[i].total_auth_data_length = 0x24;
    verb_data1->profile[i].field_type = 0x0001;
    verb_data1->profile[i].auth_data_length_1 = 0x20;
    verb_data1->profile[i].mechanism = 0x0001;
    verb_data1->profile[i].strength = 0;
    verb_data1->profile[i].mech_exp_year = 2005;
    verb_data1->profile[i].mech_exp_month = 6;
    verb_data1->profile[i].mech_exp_day = 30;
    verb_data1->profile[i].attributes[0] = 0x80;
    verb_data1->profile[i].attributes[1] = 0;
    verb_data1->profile[i].attributes[2] = 0;
    verb_data1->profile[i].attributes[3] = 0;

    memset(verb_data1->profile[i].comment, ' ', 20);

    memcpy(rule_array, "SHA-1 ", 8);
    rule_array_count = 1;
    chaining_vector_length = 128;
    hash_length = 20;

    switch (i) {
        /*-----*/
        /* Set name, role, passphrase of profile 1 */
        /*-----*/
        case 0:
            memcpy(verb_data1->profile[i].userid, "SECOFR1 ", 8);
            memcpy(verb_data1->profile[i].role, "ROLE1 ", 8);
            text_length = 10;
            text = "Is it safe";
            break;
        /*-----*/
        /* Set name, role, passphrase of profile 2 */
        /*-----*/
    }
}

```



```

        /*-----*/
case 1:
    memcpy(verb_data1->profile[i].userid,"SECOFR2 ",8);
    memcpy(verb_data1->profile[i].role, "ROLE2  ",8);
    text_length = 18;
    text        = "I think it is safe";
    break;
        /*-----*/
        /* Set name, role, passphrase of profile 3 */
        /*-----*/
case 2:
    memcpy(verb_data1->profile[i].userid,"SECOFR3 ",8);
    memcpy(verb_data1->profile[i].role, "ROLE3  ",8);
    text_length = 12;
    text        = "Is what safe";
}

/*-----*/
/* Call One_Way_Hash to hash the pass-phrase */
/*-----*/
CSNBOWH( &return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (char*)rule_array,
        &text_length,
        text,
        &chaining_vector_length,
        chaining_vector,
        &hash_length,
        verb_data1->profile[i].auth_data);
}

/*-----*/
/* Call Access_Control_Initialize (CSUAACI) to create */
/* the roles and profiles. */
/*-----*/
rule_array_count = 2;
memcpy(rule_array, "INIT-AC REPLACE ", 16);
verb_data1_length = sizeof(aggregate_profile);
verb_data2_length = sizeof(aggregate_role_header) +
        sizeof(role_header) * 3 +
        sizeof(access_control_points_header) * 3 +
        sizeof(access_control_points_segment_header)
        * 6 + /* 3 roles * 2 segments each */
        sizeof(default_bitmap) * 3 +
        sizeof(default2_bitmap) * 3;

CSUAACI( &return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (char *)rule_array,
        (long *) &verb_data1_length,
        (char *) verb_data1,
        (long *) &verb_data2_length,
        (char *) verb_data2);

if (return_code > WARNING)
    printf("Access_Control_Initialize failed. Return/reason codes: \
%d/%d\n",return_code, reason_code);
else
    printf("The new roles and profiles were successfully created\n");

/*-----*/

```

```

/* The Access_Control_Initialize SAPI verb needs to be      */
/* called one more time to replace the DEFAULT role so that */
/* a user that does not log on is not able to change any   */
/* settings in the .                                       */
/*-----*/
work_ptr = verb_data2;          /* Set working pointer to
                                start of verb data 2 storage */

aggregate_role_header.number = 1; /* Define/replace 1 roles */
aggregate_role_header.reserved = 0;
memcpy(work_ptr, (void*)&aggregate_role_header,
        sizeof(aggregate_role_header));

                                /* Adjust work pointer to
                                point after header. */
work_ptr += sizeof(aggregate_role_header);

/*-----*/
/* Fill in the fields of the role definitions.              */
/* Each role is version 1, has authentication strength of 0, */
/* has valid time from 12:00 Midnight (0) to 23:59 (x173B), */
/* is valid every day of the week. (xFE is 7 bits set),     */
/* has one access control points segment that starts at bit 0 */
/* and goes to bit x11F, and has 20 spaces for a comment.   */
/*-----*/
role_header.version[0]          = 1;
role_header.version[1]          = 0;
role_header.length              = sizeof(role_header) +
                                sizeof(access_control_points_header) +
                                2 * sizeof(access_control_points_segment_header) +
                                sizeof(default_bitmap) + sizeof(default2_bitmap);
role_header.checksum            = 0;
role_header.reserved1           = 0;
role_header.auth_strength       = 0;
role_header.lower_time          = 0;
role_header.upper_time          = 0x173B;
role_header.valid_days_of_week  = 0xFE;
role_header.reserved2           = 0;
memset(role_header.comment, ' ', 20);

access_control_points_header.number_segments = 2;
access_control_points_header.reserved       = 0;
access_control_points_segment_header.reserved = 0;

                                /* DEFAULT role id must be in */
                                /* ASCII representation.      */
memcpy(role_header.role, "\x44\x45\x46\x41\x55\x4C\x54\x20", 8);
bitmap1 = default_bitmap;
bitmap2 = default2_bitmap;

/*-----*/
/* Copy role header                                          */
/*-----*/
memcpy(work_ptr, (void*)&role_header, sizeof(role_header));

                                /* Adjust work pointer to
                                point after header. */
work_ptr += sizeof(role_header);

/*-----*/
/* Copy access control points header                          */
/*-----*/
memcpy(work_ptr,
        (void *)&access_control_points_header,
        sizeof(access_control_points_header));

                                /* Adjust work pointer to

```

```

                                point after header. */
work_ptr += sizeof(access_control_points_header);

/*-----*/
/* Copy access control points segment 1 */
/*-----*/
access_control_points_segment_header.start_bit = 0;
access_control_points_segment_header.end_bit = 0x127;
access_control_points_segment_header.number_bytes =
                                sizeof(default_bitmap);
memcpy(work_ptr,
        (void *)&access_control_points_segment_header,
        sizeof(access_control_points_segment_header));

                                /* Adjust work pointer to
                                point after header. */
work_ptr += sizeof(access_control_points_segment_header);

/*-----*/
/* Copy access control points segment 1 bitmap */
/*-----*/
memcpy(work_ptr, bitmap1, sizeof(default_bitmap));

                                /* Adjust work pointer to
                                point after bitmap. */
work_ptr += sizeof(default_bitmap);

/*-----*/
/* Copy access control points segment 2 */
/*-----*/
access_control_points_segment_header.start_bit = 0x200;
access_control_points_segment_header.end_bit = 0x23F;
access_control_points_segment_header.number_bytes =
                                sizeof(default2_bitmap);

memcpy(work_ptr,
        (void *)&access_control_points_segment_header,
        sizeof(access_control_points_segment_header));

                                /* Adjust work pointer to
                                point after header. */
work_ptr += sizeof(access_control_points_segment_header);

/*-----*/
/* Copy access control points segment 2 bitmap */
/*-----*/
memcpy(work_ptr, bitmap2, sizeof(default2_bitmap));

rule_array_count = 2;
memcpy(rule_array, "INIT-AC REPLACE ", 16);
verb_data1_length = 0;
verb_data2_length = sizeof(aggregate_role_header) +
                    sizeof(role_header) +
                    sizeof(access_control_points_header) +
                    sizeof(access_control_points_segment_header)
                    * 2 +
                    sizeof(default_bitmap) +
                    sizeof(default2_bitmap);

CSUAACI( &return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (char *)rule_array,
        (long *) &verb_data1_length,
        (char *) verb_data1,

```

```

        (long *) &verb_data2_length,
        (char *) verb_data2);

if (return_code > 4)
    printf("The default role was not replaced. Return/reason code:\n
           %d/%d\n",return_code, reason_code);
else
    printf("The default role was successfully updated.\n");
}

```

Example: ILE RPG program for creating roles or profiles for your Coprocessor: Change this program example to suit your needs for creating roles and profiles for your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* CRTROLEPRF
D*
D* Sample program to create 3 roles and 3 profiles in the
D* and change the authority for the default role.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D*       IBM CCA Basic Services Reference and Guide
D*       (SC31-8609) publication.
D*
D* Parameters: None
D*
D* Example:
D*   CALL PGM(CRTROLEPRF)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(CRTROLEPRF) SRCFILE(SAMPLE)
D* CRTPGM PGM(CRTROLEPRF) MODULE(CRTROLEPRF)
D*       BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSUAACI service program in the
D*       QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Access_Control_Initialize (CSUAACI)
D*
D*****
D*-----
D* Declare variables used by CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE      S          9B 0
D*          ** Reason code
DREASONCODE      S          9B 0
D*          ** Exit data length
DEXITDATALEN     S          9B 0
D*          ** Exit data
DEXITDATA        S          4

```

```

D*          ** Rule array count
DRULEARRAYCNT S          9B 0
D*          ** Rule array
DRULEARRAY   S          16
D*          ** Text length
DTEXTLEN    S          9B 0
D*          ** Text to hash
DTEXT       S          20
D*          ** Chaining vector length
DCHAINVCTLEN S          9B 0 INZ(128)
D*          ** Chaining vector
DCHAINVCT   S          128
D*          ** Hash length
DHASHLEN    S          9B 0 INZ(20)
D*-----
D* VERBDATA1 contains the aggregate profile structure which
D*   in turn contains 3 profiles.
D*-----
DVERBDATALEN1 S          9B 0 INZ(278)
DVERBDATA1   DS          278
D*          ** Define 3 Profiles
DNUMPROFS    S          9B 0 INZ(3)
D*          ** Reserved field
DRESR1       S          9B 0 INZ(0)
DPROF1       S          90
DPROF2       S          90
DPROF3       S          90
D*-----
D* Define the profile structure
D*-----
DPROFILESTRUCT DS
D*          ** Version 1 struct
DPROFVERS    S          2   INZ(X'0100')
D*          ** Length of profile
DPROFLEN     S          2   INZ(X'005A')
D*          ** Description of profile
DCOMMENTP    S          20  INZ('          ')
D*          ** Checksum is not used
DCHECKSUMP   S          2   INZ(X'0000')
D*          ** Logon failure count
DLOGFC       S          1   INZ(X'00')
D*          ** Reserved
DRESR2       S          1   INZ(X'00')
D*          ** Profile name
DUSERID      S          8
D*          ** Role used
DROLENAME    S          8
D*          ** Activation year (2000)
DACTYEAR     S          2   INZ(X'07D0')
D*          ** Activation month (01)
DACTMONTH    S          1   INZ(X'01')
D*          ** Activation day (01)
DACTDAY      S          1   INZ(X'01')
D*          ** Expiration year (2004)
DEXPYEAR     S          2   INZ(X'07D4')
D*          ** Expiration month (12)
DEXPMONTH    S          1   INZ(X'0C')
D*          ** Expiration day (31)
DEXPDAY      S          1   INZ(X'1F')
D*          ** Total authentication
D*          ** data length
DTOTAUTDTALEN S          2   INZ(X'0024')
D*          ** Field type
DFIELDTYPE   S          2   INZ(X'0001')
D*          ** Authentication data len
DAUTDATLEN   S          2   INZ(X'0020')

```

```

D*          ** Authentication mechanism
DMECHANISM          2  INZ(X'0001')
D*          ** Mechanism strength
DSTRENGTH          2  INZ(X'0000')
D*          ** Mech expiration year (2004)
DMCHEXPYEAR        2  INZ(X'07D4')
D*          ** Mech expiration month (12)
DMCHEXPMONTH       1  INZ(X'0C')
D*          ** Mech expiration day (31)
DMCHEXPDAY         1  INZ(X'1F')
D*          ** Attributes
DATTRIBUTES        4  INZ(X'80000000')
D*          ** Authentication data
DAUTHDATA          20  INZ('          ')
D*
D*-----
D* The Default role is being replaced
D*   Verb_data_2 length set to the length of the default role
D*-----
DVERBDATALEN2      S          9B 0 INZ(335)
D*-----
D* VERBDATA2 contains the aggregate role structure which
D*   in turn contains 3 roles.
D*-----
DVERBDATA2         DS
D*          ** Define 3 Roles
DNUMROLES          9B 0 INZ(3)
D*          ** Reserved field
DRESR3            9B 0 INZ(0)
DROLE1            109
DROLE2            109
DROLE3            109
D*
D*-----
D* Define the role structure
D*-----
DROLESTRUCT        DS
D*          ** Version 1 struct
DROLEVERS         2  INZ(X'0100')
D*          ** Length of role
DROLELEN          2  INZ(X'006D')
D*          ** Description of role
DCOMMENTR         20  INZ('          ')
D*          ** Checksum is not used
DCHECKSUMR        2  INZ(X'0000')
D*          ** Reserved field
DRESR4            2  INZ(X'0000')
D*          ** Role Name
DROLE             8
D*          ** Authentication strength is set to 0
DAUTHSTRN         2  INZ(X'0000')
D*          ** Lower time is 00:00
DLWRTIMHR         1  INZ(X'00')
DLWRTIMMN         1  INZ(X'00')
D*          ** Upper time is 23:59
DUPRTIMHR         1  INZ(X'17')
DUPRTIMMN         1  INZ(X'3B')
D*          ** Valid days of week
DVALIDDOW         1  INZ(X'FE')
D*          ** Reserved field
DRESR5            1  INZ(X'00')
D*          ** 2 Access control points segments are defined
DNUMSEG           2  INZ(X'0002')
D*          ** Reserved field
DRESR6            2  INZ(X'0000')
D*          ** Starting bit of segment 1 is 0
DSTART1           2  INZ(X'0000')

```

```

D*          ** Ending bit of segment 1 is 295 (Hex 127).
DEND1          2      INZ(X'0127')
D*          ** 37 Bytes in segment 1
DNUMBYTES1    2      INZ(X'0025')
D*          ** Reserved field
DRESR7        2      INZ(X'00')
D*          ** Segment 1 access control pointer
DBITMAP1A     8
DBITMAP1B     8
DBITMAP1C     8
DBITMAP1D     8
DBITMAP1E     5
D*          ** Starting bit of segment 2 is 512 (Hex 200)
DSTART2       2      INZ(X'0200')
D*          ** Ending bit of segment 2 is 575 (Hex 23F)
DEND2         2      INZ(X'023F')
D*          ** 8 Bytes in segment 2
DNUMBYTES2    2      INZ(X'0008')
D*          ** Reserved field
DRESR8        2      INZ(X'0000')
D*          ** Segment 2 access control points
DBITMAP2      8
D*
D*          *-----*
D*          * DEFAULT expressed in ASCII *
D*          *-----*
DDEFAULT      S          8      INZ(X'44454641554C5420')
D*
D*****
D* Prototype for Access_Control_Initialize (CSUAACI)
D*****
DCSUAACI      PR
DRETCODE      9B 0
DRSNCODE      9B 0
DEXTDTALEN    9B 0
DEXTDTA       4
DRARRAYCT     9B 0
DRARRAY       16
DVRBDTALEN1   9B 0
DVRBDTA1      278
DVRBDTALEN2   9B 0
DVRBDTA2      335
D*
D*****
D* Prototype for One_Way_Hash (CSNBOWH)
D*****
DCSNBOWH      PR
DRETCOD       9B 0
DRSNCOD       9B 0
DEXTDTALN    9B 0
DEXTDT        4
DRARRAYCT     9B 0
DRARRAY       16
DTXTLEN       9B 0
DTXT          20
DCHNVCTLEN    9B 0
DCHNVCT       128
DHSLEN        9B 0
DHS           20
D*
D*-----*
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSENDPM API
D*-----*
DMSG          S          64      DIM(3) CTDATA PERRCD(1)
MSGLENGTH     S          9B 0 INZ(64)
D             DS

```

```

DMSGTEXT          1    75
DSAPI              1    7
DFAILRETC         41   44
DFAILRSNC         46   49
DMESSAGEID        S     7  INZ('      ')
DMESSAGEFILE      S    21  INZ('      ')
DMSGKEY           S     4  INZ('      ')
DMSGTYPE          S    10  INZ('*INFO ')
DSTACKENTRY       S    10  INZ('*      ')
DSTACKCOUNTER     S     9B 0 INZ(2)
DERRCODE          DS
DBYTESIN          1    4B 0 INZ(0)
DBYTESOUT         5    8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* * * * *
C*-----*
C* Set up roles in verb data 2 *
C*-----*
C*   Set ROLE name (ROLE1)
C   MOVEL   'ROLE1 '   ROLE
C* *-----*
C* * Set Access Control Points for ROLE1
C* *
C* * DEFAULT is authorized to all access control points
C* * except for the following:
C* * 0x0018 - Load 1st part of Master Key
C* * 0x0019 - Combine Master Key Parts
C* * 0x001A - Set Master Key
C* * 0x0020 - Generate Random Master Key
C* * 0x0032 - Clear New Master Key Register
C* * 0x0033 - Clear Old Master Key Register
C* * 0x00D6 - Translate CV
C* * 0x0110 - Set Clock
C* * 0x0111 - Reinitialize device
C* * 0x0112 - Initialize access control system
C* * 0x0113 - Change user profile expiration date
C* * 0x0114 - Change authentication data (eg. passphrase)
C* * 0x0115 - Reset password failure count
C* * 0x0116 - Read Public Access Control Information
C* * 0x0117 - Delete user profile
C* * 0x0118 - Delete role
C* * 0x0119 - Load Function Control Vector
C* * 0x011A - Clear Function Control Vector
C* * 0x011B - Force User Logoff
C* * 0x0200 - Register PKA Public Key Hash
C* * 0x0201 - Register PKA Public Key, with cloning
C* * 0x0202 - Register PKA Public Key
C* * 0x0203 - Delete Retained Key
C* * 0x0204 - PKA Clone Key Generate
C* * 0x0211 - 0x21F - Clone information - obtain 1-15
C* * 0x0221 - 0x22F - Clone information - install 1-15
C* *
C* * ROLE 1 is authorized to all access control points
C* * to which the DEFAULT role is authorized plus the following:
C* *
C* * 0x0018 - Load 1st part of Master Key
C* * 0x0020 - Generate Random Master Key
C* * 0x0032 - Clear New Master Key Register
C* * 0x0053 - Load 1st part of PKA Master Key
C* * 0x0060 - Clear New PKA Master Key Register
C* * 0x0119 - Load Function Control Vector
C* * 0x0201 - Register PKA Public Key, with cloning
C* * 0x0202 - Register PKA Public Key
C* * 0x0203 - Delete Retained Key
C* * 0x0204 - PKA Clone Key Generate

```



```

C* * 0x0211 - 0x215 - Clone information - obtain 1-5
C* * 0x0221 - 0x225 - Clone information - install 1-5
C* *
C* *-----
C          EVAL      BITMAP1A = X'0003F09D80002000'
C          EVAL      BITMAP1B = X'8000100080000000'
C          EVAL      BITMAP1C = X'000A8000881F7110'
C          EVAL      BITMAP1D = X'1004031180000000'
C          EVAL      BITMAP1E = X'FF7F004F80'
C          EVAL      BITMAP2  = X'78007C007C00E60F'
C*      Copy role into aggregate structure
C          MOVE      ROLESTRUCT      ROLE1
C*      Set ROLE name (ROLE2)
C          MOVE      'ROLE2  '      ROLE
C* *-----
C* * Set Access Control Points for ROLE2
C* *
C* * ROLE 2 is authorized to all access control points
C* * to which the DEFAULT role is authorized plus the following:
C* *
C* * 0x0019 - Combine Master Key Parts
C* * 0x001A - Set Master Key
C* * 0x0033 - Clear Old Master Key Register
C* * 0x0054 - Combine PKA Master Key Parts
C* * 0x0057 - Set PKA Master Key
C* * 0x0061 - Clear Old Master Key Register
C* * 0x011A - Clear Function Control Vector
C* * 0x0200 - Register PKA Public Key Hash
C* * 0x0201 - Register PKA Public Key, with cloning
C* * 0x0203 - Delete Retained Key
C* * 0x0204 - PKA Clone Key Generate
C* * 0x0216 - 0x21A - Clone information - obtain 6-10
C* * 0x0226 - 0x22A - Clone information - install 6-10
C* *
C* *-----
C          EVAL      BITMAP1A = X'0003F07D80001000'
C          EVAL      BITMAP1B = X'8000090040000000'
C          EVAL      BITMAP1C = X'000A8000881F7110'
C          EVAL      BITMAP1D = X'1004031180000000'
C          EVAL      BITMAP1E = X'FF7F002F80'
C          EVAL      BITMAP2  = X'D80003E003E0E60F'
C*      Copy role into aggregate structure
C          MOVE      ROLESTRUCT      ROLE2
C*      Set ROLE name (ROLE3)
C          MOVE      'ROLE3  '      ROLE
C* *-----
C* * Set Access Control Points for ROLE3
C* *
C* * ROLE 3 is authorized to all access control points
C* * to which the DEFAULT role is authorized plus the following:
C* *
C* * 0x0110 - Set Clock
C* * 0x0111 - Reinitialize device
C* * 0x0112 - Initialize access control system
C* * 0x0113 - Change user profile expiration date
C* * 0x0114 - Change authentication data (eg. passphrase)
C* * 0x0115 - Reset password failure count
C* * 0x0116 - Read Public Access Control Information
C* * 0x0117 - Delete user profile
C* * 0x0118 - Delete role
C* * 0x011B - Force User Logoff
C* * 0x0200 - Register PKA Public Key Hash
C* * 0x0201 - Register PKA Public Key, with cloning
C* * 0x0203 - Delete Retained Key
C* * 0x0204 - PKA Clone Key Generate
C* * 0x021B - 0x21F - Clone information - obtain 11-15
C* * 0x022B - 0x22F - Clone information - install 11-15

```

```

C* *
C* *-----*
C          EVAL      BITMAP1A = X'0003F01D00000000'
C          EVAL      BITMAP1B = X'80000000C0000000'
C          EVAL      BITMAP1C = X'000A8000881F7110'
C          EVAL      BITMAP1D = X'1004021180000000'
C          EVAL      BITMAP1E = X'FF7FFF9F80'
C          EVAL      BITMAP2  = X'D800001F001FE60F'
C*   Copy role into aggregate structure
C          MOVEL      ROLESTRUCT   ROLE3
C*-----*
C* Set up roles in verb data 1                                     *
C*-----*
C*   Set Profile name (SECOFR1)
C          MOVEL      'SECOFR1 '   USERID
C*   Set Role name (ROLE1)
C          MOVEL      'ROLE1 '     ROLENAM
C*   Hash pass-phrase for profile 1
C          SETOFF
C          EVAL      TEXT = 'Is it safe'
C          Z-ADD     10           TEXTLEN
C          EXSR      HASHMSG
C          05
C          SETON
C          05
C*   Copy profile into aggregate structure
C          MOVEL      PROFILESTRUCT PROF1
C*   Set Profile name (SECOFR2)
C          MOVEL      'SECOFR2 '   USERID
C*   Set Role name (ROLE2)
C          MOVEL      'ROLE2 '     ROLENAM
C*   Hash pass-phrase for profile 2
C          EVAL      TEXT = 'I think it is safe'
C          Z-ADD     18           TEXTLEN
C          EXSR      HASHMSG
C          05
C          SETON
C          05
C*   Copy profile into aggregate structure
C          MOVEL      PROFILESTRUCT PROF2
C*   Set Profile name (SECOFR3)
C          MOVEL      'SECOFR2 '   USERID
C*   Set Role name (ROLE3)
C          MOVEL      'ROLE3 '     ROLENAM
C*   Hash pass-phrase for profile 3
C          EVAL      TEXT = 'Is what safe'
C          Z-ADD     12           TEXTLEN
C          EXSR      HASHMSG
C          05
C          SETON
C          05
C*   Copy profile into aggregate structure
C          MOVEL      PROFILESTRUCT PROF3
C*-----*
C* Set the keywords in the rule array                               *
C*-----*
C          MOVEL      'INIT-AC '   RULEARRAY
C          MOVE       'REPLACE '   RULEARRAY
C          Z-ADD     2             RULEARRAYCNT
C*****
C* Call Access_Control_Initialize SAPI
C*****
C          CALLP      CSUAACI      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     VERBDATALEN1:
C                                     VERBDATA1:
C                                     VERBDATALEN2:
C                                     VERBDATA2)
C* *-----*

```

```

C* * Check the return code *
C* *-----*
C   RETURNCODE   IFGT   0
C* *-----*
C* * Send failure message *
C* *-----*
C           MOVEL   MSG(1)   MSGTEXT
C           MOVE    RETURNCODE FAILRETC
C           MOVE    REASONCODE FAILRSNC
C           MOVEL   'CSUAACI' SAPI
C           EXSR    SNDMSG
C           RETURN
C           ELSE
C* *-----*
C* * Send success message *
C* *-----*
C           MOVEL   MSG(2)   MSGTEXT
C           EXSR    SNDMSG
C           ENDIF
C*
C*-----*
C* Change the Default Role *
C*-----*
C*   Set the Role name
C           MOVEL   DEFAULT   ROLE
C* *-----*
C* * Set Access Control Points for DEFAULT
C* *
C* *-----*
C           EVAL    BITMAP1A = X'0003F01D00000000'
C           EVAL    BITMAP1B = X'8000000000000000'
C           EVAL    BITMAP1C = X'000A8000881F7110'
C           EVAL    BITMAP1D = X'1004021180000000'
C           EVAL    BITMAP1E = X'FF7F406B80'
C           EVAL    BITMAP2  = X'000000000000E60F'
C*   Copy role into aggregate structure
C           MOVEL   ROLESTRUCT   ROLE1
C*
C*   Set the new verb data 2 length
C           Z-ADD   117           VERBDATALEN2
C*
C*   Set the verb data 1 length to 0 (No profiles)
C           Z-ADD   0           VERBDATALEN1
C*   Change the number of roles to 1
C           Z-ADD   1           NUMROLES
C
C*****
C* Call Access_Control_Initialize SAPI
C*****
C           CALLP   CSUAACI   (RETURNCODE:
C                               REASONCODE:
C                               EXITDATALEN:
C                               EXITDATA:
C                               RULEARRAYCNT:
C                               RULEARRAY:
C                               VERBDATALEN1:
C                               VERBDATA1:
C                               VERBDATALEN2:
C                               VERBDATA2)
C*-----*
C* Check the return code *
C*-----*
C   RETURNCODE   IFGT   0
C* *-----*
C* * Send failure message *
C* *-----*
C           MOVEL   MSG(1)   MSGTEXT

```

```

C          MOVE      RETURNCODE  FAILRETC
C          MOVE      REASONCODE  FAILRSNC
C          MOVE     'CSUAACI'    SAPI
C          EXSR      SNDMSG
C*
C          ELSE
C*  *-----*
C*  * Send success message *
C*  *-----*
C          MOVE     MSG(3)      MSGTEXT
C          EXSR      SNDMSG
C*
C          ENDIF
C*
C          SETON                                     LR
C*
C*****
C* Subroutine to send a message
C*****
C  SNDMSG      BEGSR
C          CALL      'QMHSNDPM'
C          PARM      MESSAGEID
C          PARM      MESSAGEFILE
C          PARM      MSGTEXT
C          PARM      MSGLENGTH
C          PARM      MSGTYPE
C          PARM      STACKENTRY
C          PARM      STACKCOUNTER
C          PARM      MSGKEY
C          PARM      ERRCODE
C          ENDSR
C*
C*****
C* Subroutine to Hash pass-phrase
C*****
C  HASHMSG      BEGSR
C*  *-----*
C*  * Set the keywords in the rule array *
C*  *-----*
C          MOVE     'SHA-1 '    RULEARRAY
C          Z-ADD    1           RULEARRAYCNT
C*  *-----*
C*  * Call One Way Hash SAPI *
C*  *-----*
C          CALLP    CSNBOWH    (RETURNCODE:
C                                REASONCODE:
C                                EXITDATALEN:
C                                EXITDATA:
C                                RULEARRAYCNT:
C                                RULEARRAY:
C                                TEXTLEN:
C                                TEXT:
C                                CHAINVCTLEN:
C                                CHAINVCT:
C                                HASHLEN:
C                                AUTHDATA)
C*  *-----*
C*  * Check the return code *
C*  *-----*
C          RETURNCODE  IFGT  0
C*  *-----*
C*  * Send failure message *
C*  *-----*
C          MOVE     MSG(1)      MSGTEXT
C          MOVE     RETURNCODE  FAILRETC
C          MOVE     REASONCODE  FAILRSNC
C          MOVE     'CSNBOWH'  SAPI

```

```

C          EXSR      SNDMSG
C          SETON
C          ENDIF
C*
C          ENDSR

```

05

**

CSUAACI failed with return/reason codes 9999/9999.
SECOFR1, SECOFR2, and SECOFR3 profiles were successfully created.
The Default role was successfully changed.

Example: ILE C program for enabling all access control points in the default role for your Coprocessor: Change this program example to suit your needs for enabling all access control points in the default role for your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* SETDEFAULT */
/* */
/* Sample program to authorize the default role to all access */
/* control points in the . */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: */
/* none. */
/* */
/* Example: */
/* CALL PGM(SETDEFAULT) */
/* */
/* Use these commands to compile this program on iSeries: */
/* CRTCMOD MODULE(SETDEFAULT) SRCFILE(SAMPLE) */
/* CRTPGM PGM(SETDEFAULT) MODULE(SETDEFAULT) */
/* BNSRVPGM(QCCA/CSUAACI) */
/* */
/* Note: Authority to the CSUAACI service programs */
/* in the QCCA library is assumed. */
/* */
/* The Common Cryptographic Architecture (CCA) verb used is */
/* Access_Control_Initialization (CSUAACI). */
/* */
/* Note: This program assumes the device you want to use is */
/* already identified either by defaulting to the CRP01 */
/* device or has been explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/*-----*/

#include "csucincl.h" /* header file for CCA Cryptographic

```

```

                                Service Provider for iSeries          */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main(int argc, char *argv[]) {

/*-----*/
/* standard return codes                                     */
/*-----*/

#define ERROR    -1
#define OK       0
#define WARNING  4

/*-----*/
/* parameters for CCA APIs                                   */
/*-----*/

    long return_code;
    long reason_code;
    long exit_data_length;
    char exit_data[2];
    char rule_array[4][8];
    long rule_array_count;
    long verb_data1_length;
    long verb_data2_length;
    char  verb_data1[4];

/*-----*/
/* Structure for defining a role                             */
/*-----*/
    struct role_header
    {
        char          version[2];
        short         length;
        char          comment[20];
        short         checksum;
        short         reserved1;
        char          role[8];
        short         auth_strength;
        char          lower_time_hour;
        char          lower_time_minute;
        char          upper_time_hour;
        char          upper_time_minute;
        char          valid_days_of_week;
        char          reserved2;
    } role_header;

/*-----*/
/* Structure for defining aggregate roles                     */
/*-----*/
    struct aggregate_role
    {
        long          number;
        long          reserved;
    } aggregate_role_header;

/*-----*/
/* Structures for defining the access control points in a role */
/*-----*/
    struct access_control_points_header
    {
        short         number_segments;    /* Number of segments of */
                                           /* the access points map */
        short         reserved;
    }

```

```

    } access_control_points_header;

struct access_control_points_segment_header
{
    short    start_bit;          /* Starting bit in this */
                                /* segment.              */
    short    end_bit;           /* Ending bit           */
    short    number_bytes;      /* Number of bytes in  */
                                /* this segment         */
    short    reserved;
} access_control_points_segment_header;

/*-----*/
/* Default role - access control points list - */
/*          authorized to everything           */
/*-----*/
/* For access control points 0x01 - 0x127    */
/*-----*/
char default_bitmap[] =
{ 0x00, 0x03, 0xF0, 0xFD, 0x80, 0x00, 0x30, 0x00,
  0x80, 0x00, 0x19, 0x00, 0xC0, 0x00, 0x00, 0x00,
  0x00, 0x0A, 0x80, 0x00, 0x88, 0x2F, 0x71, 0x10,
  0x18, 0x04, 0x03, 0x31, 0x80, 0x00, 0x00, 0x00,
  0xFF, 0x7F, 0xFF, 0xFF, 0x80};

/*-----*/
/* For access control points 0x200 - 0x23F    */
/*-----*/
char default2_bitmap[] =
{ 0xF8, 0x00, 0x7F, 0xFF, 0x7F, 0xFF, 0xE6, 0x0F };

unsigned char * verb_data2;
unsigned char * work_ptr;

int i;          /* Loop counter */

/*-----*/
/* Start of code */
/*-----*/

/*-----*/
/* Allocate storage for the aggregate role structure */
/*-----*/
verb_data2 = malloc(sizeof(aggregate_role_header) +
                    sizeof(role_header) +
                    sizeof(access_control_points_header) +
                    sizeof(access_control_points_segment_header)
                    * 2 +
                    sizeof(default_bitmap) +
                    sizeof(default2_bitmap));

work_ptr = verb_data2;          /* Set up work pointer */

aggregate_role_header.number = 1; /* Define/replace 1 role */
aggregate_role_header.reserved = 0; /* Initialize reserved field*/

/* Copy header to verb_data2
storage. */
memcpy(work_ptr, (void*)&aggregate_role_header,
       sizeof(aggregate_role_header));

work_ptr += sizeof(aggregate_role_header); /* Set work pointer
after role header */

/*-----*/
/* Fill in the fields of the role definition. */
/*-----*/

```

```

role_header.version[0] = 1;      /* Version 1 role      */
role_header.version[1] = 0;      /* Set length of the role */
role_header.length = sizeof(role_header)
                    + sizeof(access_control_points_header)
                    + 2 *
                    sizeof(access_control_points_segment_header)
                    + sizeof(default_bitmap)
                    + sizeof(default2_bitmap);

role_header.checksum = 0;      /* Checksum is not used */
role_header.reserved1 = 0;     /* Reserved must be 0 */
role_header.auth_strength = 0; /* Authentication strength */
                                /* is set to 0. */

                                /* Lower time is 00:00 */
role_header.lower_time_hour = 0;
role_header.lower_time_minute = 0;
                                /* Upper time is 23:59 */
role_header.upper_time_hour = 23;
role_header.upper_time_minute = 59;
role_header.valid_days_of_week = 0xFE; /* Valid every day */
                                        /* 7 bits - 1 bit each day */

role_header.reserved2 = 0;     /* Reserved must be 0 */
                                /* Role is DEFAULT */
                                /* expressed in ASCII */
memcpy(role_header.role, "\x44\x45\x46\x41\x55\x4C\x54\x20", 8);

memset(role_header.comment, ' ',20); /* No description for role */

/*-----*/
/* Copy role header into verb_data2 storage */
/*-----*/
memcpy(work_ptr, (void*)&role_header, sizeof(role_header));
work_ptr += sizeof(role_header);

/*-----*/
/* Set up access control points header and then */
/* copy it into verb_data2 storage. */
/*-----*/
access_control_points_header.number_segments = 2;
access_control_points_header.reserved = 0;
access_control_points_segment_header.reserved = 0;

memcpy(work_ptr,
       (void *)&access_control_points_header,
       sizeof(access_control_points_header));

                                /* Adjust work_ptr to point to the */
                                /* first segment */
work_ptr += sizeof(access_control_points_header);

/*-----*/
/* Set up the segment header for segment 1 and then */
/* copy into verb_data2 storage */
/*-----*/
access_control_points_segment_header.start_bit = 0;
access_control_points_segment_header.end_bit = 0x127;
access_control_points_segment_header.number_bytes =
                                sizeof(default_bitmap);

memcpy(work_ptr,
       (void *)&access_control_points_segment_header,
       sizeof(access_control_points_segment_header));

```



```

        /* Adjust work_ptr to point to the
           first segment bitmap */
work_ptr += sizeof(access_control_points_segment_header);

/*-----*/
/* Copy access control points segment 1 bitmap */
/*-----*/
memcpy(work_ptr, default_bitmap, sizeof(default_bitmap));

        /* Adjust work_ptr to point to the
           second segment */
work_ptr += sizeof(default_bitmap);

/*-----*/
/* Set up the segment header for segment 2 and then */
/* copy into verb_data2 storage */
/*-----*/
access_control_points_segment_header.start_bit = 0x200;
access_control_points_segment_header.end_bit  = 0x23F;
access_control_points_segment_header.number_bytes =
        sizeof(default2_bitmap);

memcpy(work_ptr,
        (void *)&access_control_points_segment_header,
        sizeof(access_control_points_segment_header));

        /* Adjust work_ptr to point to the
           second segment bitmap */
work_ptr += sizeof(access_control_points_segment_header);

/*-----*/
/* Copy access control points segment 2 bitmap */
/*-----*/
memcpy(work_ptr, default2_bitmap, sizeof(default2_bitmap));

/*-----*/
/* Set the length of verb data 2 (Role definition) */
/*-----*/
verb_data2_length = sizeof(aggregate_role_header) +
        role_header.length;

/*-----*/
/* Set remaining parameters */
/*-----*/
rule_array_count = 2;
memcpy(rule_array, "INIT-AC REPLACE ", 16);
verb_data1_length = 0;

/*-----*/
/* Call Access_Control_Initialize (CSUAACI) to set the */
/* default role. */
/*-----*/
CSUAACI( &return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char *)rule_array,
        &verb_data1_length,
        (unsigned char *) verb_data1,
        &verb_data2_length,
        verb_data2);

if (return_code > 4)
    printf("The default role was not replaced. Return/reason code:\n

```

```

        %d/%d\n",return_code, reason_code);
else
    printf("The default role was successfully updated.\n");
}

```

Example: ILE RPG program for enabling all access control points in the default role for your Coprocessor: Change this program example to suit your needs for enabling all access control points in the default role for your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* SETDEFAULT
D*
D* Sample program to authorize the default role to all access
D* control points in the cardX.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: None
D*
D* Example:
D* CALL PGM(SETDEFAULT)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(SETDEFAULT) SRCFILE(SAMPLE)
D* CRTPGM PGM(SETID) MODULE(SETDEFAULT)
D* BNDSRVPGM(QCCA/CSUAACI)
D*
D* Note: Authority to the CSUAACI service program in the
D* QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Access_Control_Initialize (CSUAACI)
D*
D*****
D*-----
D* Declare variables used by CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S         9B 0
D*          ** Exit data
DEXITDATA S           4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY S          16

```

```

D*          ** Verb data 1 length
DVERBDATALEN1 S          9B 0 INZ(0)
D*          ** Verb data 1
DVERBDATA1   S          4
D*          ** Verb data 2 length
DVERBDATALEN2 S         9B 0 INZ(117)
D*-----
D* Verbdata 2 contains the aggregate role structure which
D*   in turn contains 1 role - the default role
D*-----
DVERBDATA2   DS          200
D*          ** Define 1 Role
DNUMROLES    9B 0 INZ(1)
D*          ** Reserved field
DRESR1       9B 0 INZ(0)
D*          ** Version 1 struct
DVERS        2 INZ(X'0100')
D*          ** Length of role
DROLELEN     2 INZ(X'006D')
D*          ** Description of role
DCOMMENT     20 INZ('
D*          ** Checksum is not used
DCHECKSUM    2 INZ(X'0000')
D*          ** Reserved field
DRESR2       2 INZ(X'0000')
D*          ** Role Name is DEFAULT expressed in ASCII
DROLE        8 INZ(X'44454641554C5420')
D*          ** Authentication strength is set to 0
DAUTHSTRN    2 INZ(X'0000')
D*          ** Lower time is 00:00
DLWRTIMHR    1 INZ(X'00')
DLWRTIMMN    1 INZ(X'00')
D*          ** Upper time is 23:59
DUPRTIMHR    1 INZ(X'17')
DUPRTIMMN    1 INZ(X'3B')
D*          ** Valid days of week
DVALIDDOW    1 INZ(X'FE')
D*          ** Reserved field
DRESR3       1 INZ(X'00')
D*          ** 2 Access control points segments are defined
DNUMSEG      2 INZ(X'0002')
D*          ** Reserved field
DRESR4       2 INZ(X'0000')
D*          ** Starting bit of segment 1 is 0.
DSTART1      2 INZ(X'0000')
D*          ** Ending bit of segment 1 is 295 (Hex 127).
DEND1        2 INZ(X'0127')
D*          ** 37 Bytes in segment 1
DNUMBYTES1   2 INZ(X'0025')
D*          ** Reserved field
DRESR5       2 INZ(X'00')
D*          ** Segment 1 access control points
DBITMAP1A    8 INZ(X'0003F0FD80003000')
DBITMAP1B    8 INZ(X'80001900C0000000')
DBITMAP1C    8 INZ(X'000A8000882F7110')
DBITMAP1D    8 INZ(X'1804033180000000')
DBITMAP1E    5 INZ(X'FF7FFFFFFF80')
D*          ** Starting bit of segment 2 is 512 (Hex 200).
DSTART2      2 INZ(X'0200')
D*          ** Ending bit of segment 2 is 575 (Hex 23F)
DEND2        2 INZ(X'023F')
D*          ** 8 Bytes in segment 2
DNUMBYTES2   2 INZ(X'0008')
D*          ** Reserved field
DRESR6       2 INZ(X'0000')
D*          ** Segment 2 access control points
DBITMAP2     8 INZ(X'F8007FFF7FFFE60F')

```

```

D*
D*****
D* Prototype for Access_Control_Initialize (CSUAACI)
D*****
DCSUAACI          PR
DRETCODE          9B 0
DRSNCODE          9B 0
DEXTDTALEN       9B 0
DEXTDTA           4
DRARRAYCT        9B 0
DRARRAY           16
DVRBDTALEN1      9B 0
DVRBDTA1         4
DVRBDTALEN2      9B 0
DVRBDTA2         200
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG             S          64   DIM(2) CTDATA PERRCD(1)
DMSGLENGTH      S          9B 0 INZ(64)
D               DS
DMSGTEXT         1         64
DFAILRETC       41        44
DFAILRSNC       46        49
DMESSAGEID      S          7   INZ('      ')
DMESSAGEFILE    S          21  INZ('          ')
DMSGKEY         S          4   INZ('      ')
DMSGTYPE        S          10  INZ('*INFO ')
DSTACKENTRY     S          10  INZ('*   ')
DSTACKCOUNTER   S          9B 0 INZ(2)
DERRCODE        DS
DBYTESIN        1         4B 0 INZ(0)
DBYTESOUT       5         8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* *
C*-----
C* Set the keywords in the rule array *
C*-----
C          MOVEL   'INIT-AC '   RULEARRAY
C          MOVE    'REPLACE '   RULEARRAY
C          Z-ADD   2           RULEARRAYCNT
C*****
C* Call Access_Control_Initialize SAPI
C*****
C          CALLP   CSUAACI      (RETURNCODE:
C                                REASONCODE:
C                                EXITDATALEN:
C                                EXITDATA:
C                                RULEARRAYCNT:
C                                RULEARRAY:
C                                VERBDATALEN1:
C                                VERBDATA1:
C                                VERBDATALEN2:
C                                VERBDATA2)
C*-----*
C* Check the return code *
C*-----*
C          RETURNCODE  IFGT      4
C* *-----*
C* * Send failure message *
C* *-----*
C          MOVEL   MSG(1)      MSGTEXT
C          MOVE    RETURNCODE  FAILRETC

```

```

C          MOVE      REASONCODE  FAILRSNC
C          EXSR      SNDMSG
C*
C          ELSE
C*      *-----*
C*      * Send success message *
C*      *-----*
C          MOVE      MSG(2)      MSGTEXT
C          EXSR      SNDMSG
C*
C          ENDIF
C*
C          SETON                      LR
C*
C*****
C* Subroutine to send a message
C*****
C          SNDMSG      BEGSR
C          CALL        'QMHSNDPM'
C          PARM        MESSAGEID
C          PARM        MESSAGEFILE
C          PARM        MSGTEXT
C          PARM        MSGLENGTH
C          PARM        MSGTYPE
C          PARM        STACKENTRY
C          PARM        STACKCOUNTER
C          PARM        MSGKEY
C          PARM        ERRCODE
C          ENDSR

```

**
CSUAACI failed with return/reason codes 9999/9999.
The Default role was successfully set.

Example: ILE C program for changing an existing profile for your Coprocessor: Change this program example to suit your needs for changing an existing profile for your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Change certain fields in a user profile on the */
/* card. This program changes the expiration date using a new */
/* date in the form YYYYMMDD. */
/* */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 1999 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: */
/* none. */
/* */
/* Example: */
/* CALL PGM(CHG_PROF)
/*

```

```

/*                                                                    */
/* Note: This program assumes the card with the profile is            */
/*       already identified either by defaulting to the CRP01         */
/*       device or by being explicitly named using the               */
/*       Cryptographic_Resource_Allocate verb. Also this            */
/*       device must be varied on and you must be authorized         */
/*       to use this device description.                              */
/*                                                                    */
/* The Common Cryptographic Architecture (CCA) verb used is         */
/* Access_Control_Initialization (CSUAACI).                          */
/*                                                                    */
/* Use these commands to compile this program on iSeries:           */
/* ADDLIB LIB(QCCA)                                                  */
/* CRTCMOD MODULE(CHG_PROF) SRCFILE(SAMPLE)                          */
/* CRTPGM  PGM(CHG_PROF) MODULE(CHG_PROF)                            */
/*        BNDSRVPGM(QCCA/CSUAACI)                                    */
/*                                                                    */
/* Note: Authority to the CSUAACI service program in the           */
/*       QCCA library is assumed.                                    */
/*                                                                    */
/* The Common Cryptographic Architecture (CCA) verb used is         */
/* Access_Control_Initialization (CSUAACI).                          */
/*                                                                    */
/*-----*/

#include "csucincl.h"          /* header file for CCA Cryptographic */
                              /* Service Provider for iSeries      */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <decimal.h>

/*-----*/
/* standard return codes                                           */
/*-----*/

#define ERROR    -1
#define OK       0
#define WARNING  4

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters                                     */
    /*-----*/

    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 2;
    char exit_data[4];
    char rule_array[8];
    long rule_array_count = 1;

    /*-----*/
    /* fields unique to this sample program                       */
    /*-----*/

    long verb_data_length;
    char * verb_data;
    long verb_data_length2;
    char * verb_data2;

    memcpy(rule_array,"CHGEXPDT",8);          /* set rule array keywords */

```

```

verb_data_length = 8;
verb_data = "SECOFR1 ";          /* set the profile name */
verb_data_length2 = 8;
verb_data2 = "20010621";        /* set the new date */
/* invoke verb to change the expiration date in specified profile */

CSUAACI( &return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (char *)rule_array,
        &verb_data_length,
        verb_data,
        &verb_data_length2,
        verb_data2);

if ( (return_code == OK) | (return_code == WARNING) )
{
    printf("Profile expiration date was changed successfully");
    printf(" with return/reason codes ");
    printf("%ld/%ld\n\n", return_code, reason_code);
    return(OK);
}

else
{
    printf("Change of expiration date failed with return/");
    printf("reason codes ");
    printf(" %ld/%ld\n\n", return_code, reason_code);
    return(ERROR);
}
}

```

Example: ILE RPG program for changing an existing profile for your Coprocessor: Change this program example to suit your needs for changing an existing profile for your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* CHG_PROF
D*
D* Change certain fields in a user profile on the
D* card. This program changes the expiration date using a new
D* date in the form YYYYMMDD.
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of

```

```

D*      IBM CCA Basic Services Reference and Guide
D*      (SC31-8609) publication.
D*
D* Parameters: Profile
D*
D* Example:
D* CALL PGM(CHG_PROF) PARM(PROFILE)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(CHG_PROF) SRCFILE(SAMPLE)
D* CRTPGM PGM(CHG_PROF) MODULE(CHG_PROF)
D*      BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSUAACI service program in the
D*      QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Access_Control_Initialize (CSUAACI)
D*
D* This program assumes the card with the profile is
D* already identified either by defaulting to the CRP01
D* device or by being explicitly named using the
D* Cryptographic_Resource_Allocate verb. Also this
D* device must be varied on and you must be authorized
D* to use this device description.
D*****
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE      S              9B 0
D*          ** Reason code
DREASONCODE      S              9B 0
D*          ** Exit data length
DEXITDATALEN     S              9B 0
D*          ** Exit data
DEXITDATA        S              4
D*          ** Rule array count
DRULEARRAYCNT   S              9B 0
D*          ** Rule array
DRULEARRAY       S              16
D*          ** Verb data 1 length
DVERBDATALEN1   S              9B 0 INZ(8)
D*          ** Verb data 1
DVERBDATA1       S              8
D*          ** Verb data 2 length
DVERBDATALEN2   S              9B 0 INZ(8)
D*          ** Verb data 2
DVERBDATA2       S              8
D*
D*
D*****
D* Prototype for Access_Control_Initialize (CSUAACI)
D*****
DCSUAACI         PR
DRETCODE         9B 0
DRSNCODE         9B 0
DEXTDTALEN       9B 0
DEXTDTA          4
DRARRAYCT        9B 0
DRARRAY          16
DVRBDTLEN1       9B 0
DVRBDTA1         8
DVRBDTLEN2       9B 0
DVRBDTA2         8
D*
D*-----

```



```

D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG          S          75    DIM(2) CTDATA PERRCD(1)
DMSGLENGTH   S          9B 0  INZ(75)
D
DMSGTEXT      1          75
DFAILRETC    41         44
DFAILRSNC    46         49
DMESSAGEID   S          7    INZ('      ')
DMESSAGEFILE S          21   INZ('      ')
DMSGKEY       S          4    INZ('      ')
DMSGTYPE     S          10   INZ('*INFO ')
DSTACKENTRY  S          10   INZ('*      ')
DSTACKCOUNTER S          9B 0  INZ(2)
DERRCODE     DS
DBYTESIN     1          4B 0  INZ(0)
DBYTESOUT    5          8B 0  INZ(0)
C*****
C* START OF PROGRAM *
C* *
C*-----*
C* Parameter is profile to be changed. *
C*-----*
C   *ENTRY      PLIST
C               PARM          VERBDATA1
C*-----*
C* Set the keywords in the rule array *
C*-----*
C               MOVE      'CHGEXPDT'  RULEARRAY
C               Z-ADD     1           RULEARRAYCNT
C*-----*
C* Set new expiration date *
C*-----*
C               MOVE      '20061231'  VERBDATA2
C*-----*
C* Call Access_Control_Initialize SAPI *
C*-----*
C               CALLP     CSUAACI      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     VERBDATALEN1:
C                                     VERBDATA1:
C                                     VERBDATALEN2:
C                                     VERBDATA2)
C*-----*
C* Check the return code *
C*-----*
C   RETURNCODE  IFGT      0
C*-----*
C* * Send error message *
C*-----*
C               MOVE      MSG(1)      MSGTEXT
C               MOVE      RETURNCODE  FAILRETC
C               MOVE      REASONCODE  FAILRSNC
C               EXSR      SNDMSG
C*
C               ELSE
C*-----*
C* * Send success message *
C*-----*
C               MOVE      MSG(2)      MSGTEXT
C               EXSR      SNDMSG
C*

```

```

C          ENDIF
C*
C          SETON                               LR
C*
C*****
C* Subroutine to send a message
C*****
C  SNDMSG      BEGSR
C              CALL      'QMHSNDPM'
C              PARM      MESSAGEID
C              PARM      MESSAGEFILE
C              PARM      MSGTEXT
C              PARM      MSGLENGTH
C              PARM      MSGTYPE
C              PARM      STACKENTRY
C              PARM      STACKCOUNTER
C              PARM      MSGKEY
C              PARM      ERRCODE
C              ENDSR
C*

```

```

**
CSUAACI failed with return/reason codes 9999/9999'
The request completed successfully

```

Set the environment ID and clock

The Environment ID (EID)

Your Coprocessor stores the EID as an identifier. The easiest and fastest way to set the EID is to use the Cryptographic Coprocessor configuration web-based utility found off of the iSeries Tasks page at <http://server-name:2001>. The utility includes the Basic configuration wizard that is used when the Coprocessor is in an un-initialized state. If the Coprocessor already has been initialized, then click on **Manage configuration** and then click on **Attributes** to set the EID.

If you would prefer to write your own application to set the EID, you can do so by using the Cryptographic_Facility_Control (CSUACFC) API verb. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both perform the same function.

- “Example: ILE C program for setting the environment ID on your Coprocessor” on page 61
- “Example: ILE RPG program for setting the environment ID on your Coprocessor” on page 63

Your Cryptographic Coprocessor copies the EID into every PKA key token that your Coprocessor creates. The EID helps the Coprocessor identify keys that it created as opposed to keys that another Coprocessor created.

The Clock

The Coprocessor uses its clock-calendar to record time and date and to determine whether a profile can log on. The default time is Greenwich Mean Time (GMT). Because of its function, you should set the clock inside your Coprocessor before removing the default role’s capability of setting it.

The easiest and fastest way to set the clock is to use the Cryptographic Coprocessor configuration web-based utility found off of the iSeries Tasks page at <http://server-name:2001>. The utility includes the Basic configuration wizard that is used when the Coprocessor is in an un-initialized state. If the Coprocessor already has been initialized, then use click on **Manage configuration** and then click on **Attributes** to set the clock.

If you would prefer to write your own application to set the clock, you can do so by using the Cryptographic_Facility_Control (CSUACFC) API verb. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both perform the same function.

- “Example: ILE C program for setting the clock on your Coprocessor” on page 65
- “Example: ILE RPG program for setting the clock on your Coprocessor” on page 68

Example: ILE C program for setting the environment ID on your Coprocessor: Change this program example to suit your needs for setting the environment ID on your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Set the environment ID on the card, based on a          */
/* 16-byte sample value defined in this program.          */
/*                                                       */
/*                                                       */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999                */
/*                                                       */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot    */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are    */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF    */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files.                              */
/*                                                       */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide              */
/* (SC31-8609) publication.                               */
/*                                                       */
/* Parameters:                                           */
/* none.                                                 */
/*                                                       */
/* Example:                                             */
/* CALL PGM(SETVID)                                     */
/*                                                       */
/* Note: This program assumes the device to use is      */
/* already identified either by defaulting to the CRP01  */
/* device or by being explicitly named using the        */
/* Cryptographic_Resource_Allocate verb. Also this      */
/* device must be varied on and you must be authorized  */
/* to use this device description.                       */
/*                                                       */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA)                                     */
/* CRTCMOD MODULE(SETVID) SRCFILE(SAMPLE)               */
/* CRTPGM PGM(SETVID) MODULE(SETVID)                   */
/* BNSRVPGM(QCCA/CSUACFC)                               */
/*                                                       */
/* Note: Authority to the CSUACFC service program in the */
/* QCCA library is assumed.                             */
/*                                                       */
/* The Common Cryptographic Architecture (CCA) verb used is */
/* Cryptographic_Facilites_Control (CSUACFC).           */
/*                                                       */
/*-----*/

#include "csucincl.h" /* header file for CCA Cryptographic */
/* Service Provider for iSeries */

```

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR    -1
#define OK       0
#define WARNING  4

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters */
    /*-----*/

    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 2;
    char exit_data[4];
    char rule_array[2][8];
    long rule_array_count = 2;

    /*-----*/
    /* fields unique to this sample program */
    /*-----*/

    long verb_data_length;
    char * verb_data = "SOME ID data 16@";

    /* set keywords in the rule array */
    memcpy(rule_array, "ADAPTER1SET-EID ", 16);

    verb_data_length = 16;

    /* invoke the verb to set the environment ID */

    CSUACFC(&return_code,
            &reason_code,
            &exit_data_length,
            exit_data,
            &rule_array_count,
            (char *)rule_array,
            &verb_data_length,
            verb_data);

    if ( (return_code == OK) | (return_code == WARNING) )
    {
        printf("Environment ID was successfully set with ");
        printf("return/reason codes %1d/%1d\n\n", return_code, reason_code);
        return(OK);
    }
    else
    {
        printf("An error occurred while setting the environment ID.\n");
        printf("Return/reason codes %1d/%1d\n\n", return_code, reason_code);
    }
}

```

```

return(ERROR);
}
}

```

Example: ILE RPG program for setting the environment ID on your Coprocessor: Change this program example to suit your needs for setting the environment ID on your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* SETEID
D*
D* Set the environment ID on the card, based on a
D* 16-byte sample value defined in this program.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: None
D*
D* Example:
D* CALL PGM(SETSEID)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(SETSEID) SRCFILE(SAMPLE)
D* CRTPGM PGM(SETSEID) MODULE(SETSEID)
D* BNDSRVPGM(QCCA/CSUACFC)
D*
D* Note: Authority to the CSUACFC service program in the
D* QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Facility_Control (CSUACFC)
D*
D*****
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S         9B 0
D*          ** Exit data
DEXITDATA S           4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY S           16
D*          ** Verb data length

```

```

DVERBDATALEN      S          9B 0
D*                ** Verb data
DVERBDATA         S          16  INZ('Card ID 01234567')
D*
D*
D*****
D* Prototype for Cryptographic_Facility_Control (CSUACFC)
D*****
DCSUACFC          PR
DRETCODE          9B 0
DRSNCODE          9B 0
DEXTDTALEN       9B 0
DEXTDTA           4
DRARRAYCT        9B 0
DRARRAY           16
DVRBDTALEN       9B 0
DVRBDTA          16
D*
D*-----
D*                ** Declares for sending messages to the
D*                ** job log using the QMHSNDPM API
D*-----
DMSG              S          75  DIM(2) CTDATA PERRCD(1)
DMSGLENGTH        S          9B 0 INZ(75)
D
DMSGTEXT          1          80
DFAILRET          41         44
DFAILRNC          46         49
DMESSAGEID        S          7  INZ(' ')
DMESSAGEFILE      S          21 INZ(' ')
DMSGKEY           S          4  INZ(' ')
DMSGTYPE          S          10 INZ('*INFO ')
DSTACKENTRY       S          10 INZ('* ')
DSTACKCOUNTER     S          9B 0 INZ(2)
DERRCODE          DS
DBYTESIN          1          4B 0 INZ(0)
DBYTESOUT         5          8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM
C*
C*-----
C* Set the keyword in the rule array
C*-----
C          MOVE      'ADAPTER1'  RULEARRAY
C          MOVE      'SET-EID '  RULEARRAY
C          Z-ADD     2           RULEARRAYCNT
C*-----
C* Set the verb data length to 16
C*-----
C          Z-ADD     16          VERBDATALEN
C*****
C* Call Cryptographic Facility Control SAPI
C*****
C          CALLP     CSUACFC      (RETURNCODE:
C                                REASONCODE:
C                                EXITDATALEN:
C                                EXITDATA:
C                                RULEARRAYCNT:
C                                RULEARRAY:
C                                VERBDATALEN:
C                                VERBDATA)
C*-----
C* Check the return code *
C*-----
C          RETURNCODE  IFGT      4
C*
C*-----

```

```

C*          * Send error message *
C*          *-----*
C              MOVE      MSG(1)      MSGTEXT
C              MOVE      RETURNCODE  FAILRETC
C              MOVE      REASONCODE  FAILRSNC
C              EXSR      SNDMSG
C*
C              ELSE
C*          *-----*
C*          * Send success message *
C*          *-----*
C              MOVE      MSG(2)      MSGTEXT
C              EXSR      SNDMSG
C*
C              ENDIF
C*
C              SETON                                     LR
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C                  CALL      'QMHSNDPM'
C                  PARM      MESSAGEID
C                  PARM      MESSAGEFILE
C                  PARM      MSGTEXT
C                  PARM      MSGLENGTH
C                  PARM      MSGTYPE
C                  PARM      STACKENTRY
C                  PARM      STACKCOUNTER
C                  PARM      MSGKEY
C                  PARM      ERRCODE
C                  ENDSR

```

**
CSUACFC failed with return/reason codes 9999/9999.
The Environment ID was successfully set.

Example: ILE C program for setting the clock on your Coprocessor: Change this program example to suit your needs for setting the clock on your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Set the clock on the card, based on a string from */
/* the command line. The command line string must be of */
/* form YYYYMMDDHHMSSWW, where WW is the day of week (01 */
/* means Sunday and 07 means Saturday). */
/* */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: */

```

```

/* char * new time 16 characters */
/* */
/* Example: */
/* CALL PGM(SETCLOCK) PARM('1999021011375204') */
/* */
/* */
/* Note: This program assumes the device to use is */
/* already identified either by defaulting to the CRP01 */
/* device or by being explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/* */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(SETCLOCK) SRCFILE(SAMPLE) */
/* CRTPGM PGM(SETCLOCK) MODULE(SETCLOCK) */
/* BNSRVPGM(QCCA/CSUACFC) */
/* */
/* Note: Authority to the CSUACFC service program in the */
/* QCCA library is assumed. */
/* */
/* The Common Cryptographic Architecture (CCA) verb used is */
/* Cryptographic_Facilities_Control (CSUACFC). */
/* */
/*-----*/

#include "csucincl.h" /* header file for CCA Cryptographic */
/* Service Provider for iSeries */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR -1
#define OK 0
#define WARNING 4

void help(void)
{
    printf("\n\nThis program loads the time and date into the card.\n");
    printf("It requires a single command line parameter containing the \n");
    printf("new date and time in the form YYYYMMDDHHMMSSWW, where WW is the\n");
    printf("day of the week, 01 meaning Sunday and 07 meaning Saturday.\n\n");
}

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters */
    /*-----*/

    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 2;
    char exit_data[4];

```



```

char rule_array[2][8];
long rule_array_count = 2;

/*-----*/
/* fields unique to this sample program */
/*-----*/

long verb_data_length;
char * verb_data;

if (argc != 2)
{
    help();

    return(ERROR);
}

if (strlen(argv[1]) != 16)
{
    printf("Your input string is not the right length.");

    help();

    return(ERROR);
}

/* set keywords in the rule array */
memcpy(rule_array,"ADAPTER1SETCLOCK",16);

verb_data_length = 16;

/* copy keyboard input for new time */
verb_data = argv[1];

/* Set the clock to the time the user gave us */
CSUACFC( &return_code,
         &reason_code,
         &exit_data_length,
         exit_data,
         &rule_array_count,
         (char *)rule_array,
         &verb_data_length,
         verb_data);

if ( (return_code == OK) | (return_code == WARNING) )
{
    printf("Clock was successfully set.\nReturn/");

    printf("reason codes %ld/%ld\n\n", return_code, reason_code);

    return(OK);
}

else
{
    printf("An error occurred while setting the clock.\nReturn");

    printf("/reason codes %ld/%ld\n\n", return_code, reason_code);

    return(ERROR);
}
}

```

Example: ILE RPG program for setting the clock on your Coprocessor: Change this program example to suit your needs for setting the clock on your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* SETCLOCK
D*
D* Set the clock on the card, based on a string from
D* the command line. The command line string must be of
D* form YYYYMMDDHHMSSWW, where WW is the day of week (01
D* means Sunday and 07 means Saturday).
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters:
D* char * new time 16 characters
D*
D* Example:
D* CALL PGM(SETCLOCK) PARM('2000061011375204')
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(SETCLOCK) SRCFILE(SAMPLE)
D* CRTPGM PGM(SETCLOCK) MODULE(SETCLOCK)
D* BNDSRVPGM(QCCA/CSUACFC)
D*
D* Note: Authority to the CSUACFC service program in the
D* QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Facilty_Control (CSUACFC)
D*
D*****
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S         9B 0
D*          ** Exit data
DEXITDATA S           4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY S           16
D*          ** Verb data length
DVERBDATALEN S         9B 0
D*          ** Verb data

```

```

DVERBDATA      S          16
D*
D*****
D* Prototype for Cryptographic_Facility_Control (CSUACFQ)
D*****
DCSUACFC      PR
DRETCODE      9B 0
DRSNCODE      9B 0
DEXTDTALEN    9B 0
DEXTDTA       4
DRARRAYCT     9B 0
DRARRAY       16
DVRBDTALEN    9B 0
DVRBDTA       16
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG          S          75  DIM(6) CTDATA PERRCD(1)
DMSGLENGTH    S          9B 0 INZ(75)
D             DS
DMSGTEXT      1          80
DFAILRETC     41        44
DFAILRSNC     46        49
DMESSAGEID    S          7  INZ(' ')
DMESSAGEFILE  S          21 INZ(' ')
DMSGKEY       S          4  INZ(' ')
DMSGTYPE      S          10 INZ('*INFO ')
DSTACKENTRY   S          10 INZ('* ')
DSTACKCOUNTER S          9B 0 INZ(2)
DERRCODE      DS
DBYTESIN      1          4B 0 INZ(0)
DBYTESOUT     5          8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* * * * *
C  *ENTRY      PLIST
C              PARM          VERBDATA
C* * * * *
C*-----
C* Check the number of parameters passed in *
C*-----
C              IF          (%PARMS < 1)
C* * * * *
C* * Send message describing the format of the parameter *
C* * * * *
C              MOVEL      MSG(3)      MSGTEXT
C              EXSR       SNDMSG
C              MOVEL      MSG(4)      MSGTEXT
C              EXSR       SNDMSG
C              MOVEL      MSG(5)      MSGTEXT
C              EXSR       SNDMSG
C              MOVEL      MSG(6)      MSGTEXT
C              EXSR       SNDMSG
C              RETURN
C              ENDIF
C* * * * *
C*-----
C* Set the keyword in the rule array *
C*-----
C              MOVEL      'ADAPTER1'  RULEARRAY
C              MOVE      'SETCLOCK'   RULEARRAY
C              Z-ADD      2           RULEARRAYCNT
C*-----
C* Set the verb data length to 16 *

```

```

C*-----*
C          Z-ADD      16          VERBDATALEN
C*****
C* Call Cryptographic Facility Control SAPI */
C*****
C          CALLP      CSUACFC      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     VERBDATALEN:
C                                     VERBDATA)
C*-----*
C* Check the return code *
C*-----*
C          RETURNCODE  IFGT      4
C*          *-----*
C*          * Send error message *
C*          *-----*
C          MOVE      MSG(1)      MSGTEXT
C          MOVE      RETURNCODE  FAILRETC
C          MOVE      REASONCODE  FAILRSNC
C          EXSR      SNDMSG
C*
C          ELSE
C*          *-----*
C*          * Send success message *
C*          *-----*
C          MOVE      MSG(2)      MSGTEXT
C          EXSR      SNDMSG
C*
C          ENDIF
C*
C          SETON
C*
C*****
C* Subroutine to send a message
C*****
C          SNDMSG      BEGSR
C          CALL      'QMHSNDPM'
C          PARM      MESSAGEID
C          PARM      MESSAGEFILE
C          PARM      MSGTEXT
C          PARM      MSGLENGTH
C          PARM      MSGTYPE
C          PARM      STACKENTRY
C          PARM      STACKCOUNTER
C          PARM      MSGKEY
C          PARM      ERRCODE
C          ENDSR

```

LR

**
CSUACFC failed with return/reason codes 9999/9999.
The request completed successfully.
This program loads the time and date into the card.
It requires a single command line parameter containing the
new date and time in the form YYYYMMDDHHMSSWW, where WW is the
day of the week, 01 meaning Sunday and 07 meaning Saturday.

Load a function control vector

After “Create and define roles and profiles” on page 23, you must load a function control vector (FCV) for your Cryptographic Coprocessor. Without it, your Coprocessor will be unable to perform any cryptographic operations.

A function control vector is a digitally signed value stored in a file provided by IBM. When you install one of the 5722-ACx Cryptographic Access Provider products, the file is stored in the root file system

with a path of /QIBM/ProdData/CAP/FCV.CRT. This value enables the cryptographic application within the Coprocessor to yield a level of cryptographic service consistent with applicable import and export regulations.

The easiest and fastest way to load the FCV is to use the Cryptographic Coprocessor configuration web-based utility found off of the iSeries Tasks page at <http://server-name:2001>. The utility includes the Basic configuration wizard that is used when the Coprocessor is in an un-initialized state. If the Coprocessor has already been initialized, then click on **Manage configuration** and then click on **Attributes** to load the FCV.

If you would prefer to write your own application to load the FCV, you can do so by using the Cryptographic_Facility_Control (CSUACFC) API verb. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both perform the same function.

- “Example: ILE C program for loading a function control vector for your Cryptographic Coprocessor”
- “Example: ILE RPG program for loading a function control vector for your Coprocessor” on page 74

Two other example programs are provided that show how to clear the function control vector. One of them is written in ILE C, while the other is written in ILE RPG.

- “Example: ILE C program for clearing a function control vector from your Coprocessor” on page 78
- “Example: ILE RPG program for clearing a function control vector from your Coprocessor” on page 79

After you load a function control vector for your Coprocessor, you can load and set a master key using “Load and set a master key” on page 82 to use to encrypt keys.

Example: ILE C program for loading a function control vector for your Cryptographic Coprocessor:
Change this program example to suit your needs for loading a function control vector for your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```
/*-----*/
/* Load the Function Control Vector into the card. */
/* The Function Control Vector enables the cryptographic */
/* functions of the card and is shipped with the */
/* Cryptographic Access Provider products. */
/*
/* COPYRIGHT 5769-SS1 (c) IBM Corp 1999 */
/*
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function*/
/* of these programs. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* EXPRESSLY DISCLAIMED. IBM provides no program services for*/
/* these programs and files. */
/*
/* Note: The Function Control Vector is stored in an IFS */
/* file owned by the system. The format of this */
/* vector is described in an appendix of the */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/*
/* Parameters: */
/* none. */
/*
/* Example: */
/* CALL PGM(LOAD_FCV) */
/*
```

```

/* Note: This program assumes the device you want to load is */
/* already identified either by defaulting to the CRP01 */
/* device or has been explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/* */
/* Use the following commands to compile this program: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(LOAD_FCV) SRCFILE(SAMPLE) SYSIFCOPT(*IFSIO) */
/* CRTPGM PGM(LOAD_FCV) MODULE(LOAD_FCV) + */
/* BNSDRVPGM(QCCA/CSUACFC) */
/* */
/* Note: Authority to the CSUACFC service program in the */
/* QCCA library is assumed. */
/* */
/* Common Cryptographic Architecture (CCA) verbs used: */
/* Cryptographic_Facility_Control (CSUACFC) */
/* */
/*-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <decimal.h>
#include "csucincl.h" /* header file for CCA Cryptographic
                      Service Provider for iSeries */

/*-----*/
/* function to translate ASCII to EBCDIC and/or EBCDIC to ASCII */
/*-----*/

#pragma linkage(QDCXLATE, OS, nowiden)
void QDCXLATE(decimal(5,0)*,
              char *,
              char *,
              char *);

int main(void)
{

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR -1
#define OK 0

/*-----*/
/* standard CCA parameters */
/*-----*/

long return_code;
long reason_code;
long exit_data_length;
char exit_data[2];
char rule_array[4][8];
long rule_array_count;

/*-----*/
/* fields unique to this sample program */
/*-----*/

long verb_data_length;
char *verb_data;
char buffer[1000];

```

```

char description[81];
decimal(5,0) descr_length = 80;
int num_bytes;
FILE *fcv;

/*-----*/
/* retrieve FCV from IBM supplied file */
/*-----*/
fcv = fopen("/QIBM/ProdData/CAP/FCV.CRT", "rb");
if (fcv==NULL)
{
    printf("Function Control Vector file not available\n\n");
    return ERROR;          /* File not found or not authorized */
}

num_bytes = fread(buffer,1,1000,fcv);
fclose(fcv);

if (num_bytes != 802)
{
    printf("Function Control Vector file has wrong size\n\n");
    return ERROR;          /* Incorrect number of bytes read */
}

/*-----*/
/* extract fields in FCV needed by card */
/* Note: use offsets and lengths from CCA publication listed earlier */
/*-----*/

memcpy(description, &buffer[390],80);
description[80] = 0;
QDCXLATE(&descr_length, description, "QEBCDIC ", "QSYS ");
printf("Loading Function Control Vector: %s\n",description);

verb_data_length = 204;
verb_data = &buffer[470];

rule_array_count = 2;
memcpy((char*)rule_array,"ADAPTER1LOAD-FCV",16);

/*-----*/
/* Load the card with the FCV just retrieved */
/*-----*/
CSUACFC(&return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (char*)rule_array,
        &verb_data_length,
        verb_data);

if (return_code != 0)
{
    printf("Function Control Vector rejected for reason %d/%d\n\n",
           return_code, reason_code);
    return ERROR;          /* Operation failed. */
}
else
{
    printf("Loading Function Control Vector succeeded\n\n");
    printf("SAPI returned %ld/%ld\n\n", return_code, reason_code);
    return OK;
}
}

```

Example: ILE RPG program for loading a function control vector for your Coprocessor: Change this program example to suit your needs for loading a function control vector for your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* LOAD_FCV
D*
D* Load the Function Control Vector into the card.
D* The Function Control Vector enables the cryptographic
D* functions of the card and is shipped with the
D* Cryptographic Access Provider products.
D*
D* The Function Control Vector is contained within a stream
D* file. Before compiling and running this program, you
D* must copy the contents of the stream file to a database
D* member. An example of how to do this is shown in the
D* instructions below for compiling and running this program.
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: None
D*
D* Example:
D* CALL PGM(LOAD_FCV)
D*
D* Use these commands to compile this program on iSeries:
D*
D* CRTRPGMOD MODULE(LOAD_FCV) SRCFILE(SAMPLE)
D*
D* CRTPGM PGM(LOAD_FCV) MODULE(LOAD_FCV)
D* BNDSRVPGM(QCCA/CSUACFC)
D*
D* Note: Authority to the CSUACFC service program in the
D* QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Facilty_Control (CSUACFC)
D*
D*****
D*-----
D* Declare variables used by CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S          9B 0
D*          ** Exit data
DEXITDATA S            4
D*          ** Rule array count

```



```

DRULEARRAYCNT      S          9B 0
D*                ** Rule array
DRULEARRAY         S          16
D*                ** Verb data length
DVERBDATALEN      S          9B 0 INZ(204)
D*                ** Verb data
DVERBDATA         S          204
D*-----
D* Declare variables for working with files
D*-----
D*                ** File descriptor
D*                S          9B 0
D*                ** File path
D*                S          80    INZ('/QIBM/ProdData/CAP/FCV.CRT')
D*                ** Open Flag - Open for Read only
D*                S          10I 0 INZ(1)
D*                ** Structure of Function control vector file
D*                DS
D*                S          802
D*                S          391 470
D*                S          471 674
D*                ** Length of data read from file
D*                S          9B 0
D*                ** Declares for calling QDCXLATE API
D*                S          10    INZ('QEBCDIC ')
D*                S          10    INZ('QSYS   ')
D*                S          5P 0 INZ(80)
D*                ** Index into a string
D*                S          5B 0
D*                ** Variable to hold temporary character value
D*                S          1
D*
D*****
D* Prototype for Cryptographic_Facility_Control (CSUACFC)
D*****
DCSUACFC          PR
DRETCODE          9B 0
DRSNCODE          9B 0
DEXTDTALEN       9B 0
DEXTDTA          4
DRARRAYCT        9B 0
DRARRAY          16
DVRBDTALEN       9B 0
DVRBDTA          204
D*
D*****
D* Prototype for open()
D*****
D* value returned = file descriptor (OK), -1 (error)
Dopen            PR          9B 0 EXTPROC('open')
D* path name of file to be opened.
D                128    OPTIONS(*VARSIZE)
D* Open flags
D                9B 0 VALUE
D* (OPTIONAL) mode - access rights
D                10U 0 VALUE OPTIONS(*NOPASS)
D* (OPTIONAL) codepage
D                10U 0 VALUE OPTIONS(*NOPASS)
D*
D*****
D* Prototype for read()
D*****
D* value returned = number of bytes actually read, or -1
Dread           PR          9B 0 EXTPROC('read')
D* File descriptor returned from open()
D                9B 0 VALUE
D* Input buffer

```

```

D          2500  OPTIONS(*VARSIZE)
D*   Length of data to be read
D          9B 0 VALUE
D*
D*****
D* Prototype for close()
D*****
D*   value returned = 0 (OK), or -1
Dclose      PR          9B 0 EXTPROC('close')
D*   File descriptor returned from open()
D          9B 0 VALUE
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG        S          80  DIM(4) CTDATA PERRCD(1)
DMSGLENGTH  S          9B 0 INZ(80)
D          DS
DMSGTEXT    1          80
DFAILRETC   41         44
DFAILRNC    46         49
DMESSAGEID  S          7  INZ('      ')
DMESSAGEFILE S         21  INZ('      ')
DMSGKEY     S          4  INZ('      ')
DMSGTYPE    S         10  INZ('*INFO ')
DSTACKENTRY S         10  INZ('*      ')
DSTACKCOUNTER S        9B 0 INZ(2)
DERRCODE    DS
DBYTESIN    1          4B 0 INZ(0)
DBYTESOUT   5          8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* *
C*-----
C* Open the FCV file *
C*-----
C* *-----*
C* ** Null terminate path name *
C* *-----*
C          EVAL      %SUBST(PATH:27:1) = X'00'
C* *-----*
C* * Open the file *
C* *-----*
C          EVAL      FILED = open(PATH: OFLAGR)
C* *-----*
C* * Check if open worked *
C* *-----*
C  FILED      IFEQ      -1
C* *-----*
C* * Open failed, send an error message *
C* *-----*
C          MOVEL     MSG(1)      MSGTEXT
C          EXSR      SNDMSG
C          RETURN
C*
C          ENDIF
C* *-----*
C* * Open worked, read the FCV, and close the file *
C* *-----*
C          Z-ADD     802          INLEN
C          EVAL      INLEN = read(FILED: FLDDTA: INLEN)
C          CALLP     close      (FILED)
C*
C* *-----*
C* * Check if read operation was OK *

```

```

C*  *-----*
C  INLEN      IFEQ      -1
C              MOVEL    MSG(2)      MSGTEXT
C              EXSR     SNDMSG
C              RETURN
C              ENDIF
C*
C*-----*
C* Copy the FCV to the verb data parameter.      *
C*-----*
C              MOVEL    FNCCTLVCT    VERBDATA
C*-----*
C* Convert description to EBCDIC and display it   *
C*-----*
C              CALL     'QDCXLATE'
C              PARM
C              PARM     DESCLEN
C              PARM     DESCR
C              PARM     XLTTBL
C              PARM     TBLLIB
C              MOVEL    DESCR      MSGTEXT
C              Z-ADD    80         INDEX
C*-----*
C* Replace trailing null characters in description *
C* with space characters.                        *
C*-----*
C              SETOFF
C              DOU      *IN50
C              EVAL     CHAR = %SUBST(MSGTEXT:INDEX:1)
C  CHAR        IFNE     X'00'
C              SETON
C              ELSE
C              EVAL     %SUBST(MSGTEXT:INDEX:1) = ' '
C              SUB      1         INDEX
C  INDEX        IFEQ     0
C              SETON
C              ENDIF
C              ENDIF
C              ENDDO
C              EXSR     SNDMSG
C*-----*
C* Set the keywords in the rule array            *
C*-----*
C              MOVEL    'ADAPTER1'    RULEARRAY
C              MOVE     'LOAD-FCV'    RULEARRAY
C              Z-ADD    2              RULEARRAYCNT
C*****
C* Call Cryptographic Facility Control SAPI     */
C*****
C              CALLP    CSUACFC      (RETURNCODE:
C              REASONCODE:
C              EXITDATALEN:
C              EXITDATA:
C              RULEARRAYCNT:
C              RULEARRAY:
C              VERBDATALEN:
C              VERBDATA)
C* *-----*
C* * Check the return code *
C* *-----*
C  RETURNCODE  IFGT     0
C* *-----*
C* * Send failure message *
C* *-----*
C              MOVEL    MSG(3)      MSGTEXT
C              MOVE     RETURNCODE  FAILRETC
C              MOVE     REASONCODE  FAILRSNC
C              EXSR     SNDMSG

```

```

C*
C          ELSE
C*
C*  *-----*
C*  *  Send success message *
C*  *-----*
C          MOVEL    MSG(4)      MSGTEXT
C          EXSR     SNDMSG
C          ENDIF
C*
C          SETON                                LR
C*
C*****
C* Subroutine to send a message
C*****
C  SNDMSG    BEGSR
C            CALL      'QMHSNDPM'
C            PARM      MESSAGEID
C            PARM      MESSAGEFILE
C            PARM      MSGTEXT
C            PARM      MSGLENGTH
C            PARM      MSGTYPE
C            PARM      STACKENTRY
C            PARM      STACKCOUNTER
C            PARM      MSGKEY
C            PARM      ERRCODE
C            ENDSR

```

```

**
Error trying to open FCV file.
Error reading data from FCV file.
CSUACFC failed with return/reason codes 9999/9999.
The Function Control Vector was successfully loaded.

```

Example: ILE C program for clearing a function control vector from your Coprocessor: Change this program example to suit your needs for clearing a function control vector from your Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Clear the Function Control Vector from the card. */
/* The Function Control Vector enables the cryptographic */
/* functions of the card. Clearing it from the */
/* disabled the cryptographic functions. */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 2000 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or */
/* functions of these program. All programs contained */
/* herein are provided to you "AS IS". THE IMPLIED */
/* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A */
/* PARTICULAR PURPOSE ARE ARE EXPRESSLY DISCLAIMED. IBM */
/* provides no program services for these programs and files.*/
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: */
/* none. */
/* */
/* Example: */
/* CALL PGM(CLEARFCV)
/*

```

```

/*                                                                    */
/* Use the following command to compile this program:                  */
/* CRTCMOD MODULE(CLEARFCV) SRCFILE(SAMPLE)                          */
/* CRTPGM  PGM(CLEARFCV) MODULE(CLEARFCV)                            */
/*          BNDSRVPGM(QCCA/CSUACFC)                                   */
/*                                                                    */
/* Common Cryptographic Architecture (CCA) verbs used:              */
/* - Cryptographic_Facility_Control (CSUACFC)                       */
/*                                                                    */
/*-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "csucincl.h"

void main(void)
{
    long return_code;
    long reason_code;
    long exit_data_length;
    char exit_data[2];
    char rule_array[4][8];
    long rule_array_count;
    long verb_data_length;
    char *verb_data;
    char buffer[4];

/*-----*/
/* No verb data is needed for this option.                            */
/*-----*/
    verb_data_length = 0;
    verb_data = buffer;

/*-----*/
/* Rule array has two elements or rule array keywords                 */
/*-----*/
    rule_array_count = 2;
    memcpy((char*)rule_array,"ADAPTER1CLR-FCV ",16);

/*-----*/
/* Clear the Function control vector from the card                   */
/*-----*/
    CSUACFC(&return_code,
            &reason_code,
            &exit_data_length,
            exit_data,
            &rule_array_count,
            (char*)rule_array,
            &verb_data_length,
            verb_data);

    if (return_code != 0)
        printf("Operation failed: return code %d : reason code %d \n",
              return_code, reason_code);
    else
        printf("FCV is successfullly cleared\n");
}

```

Example: ILE RPG program for clearing a function control vector from your Coprocessor: Change this program example to suit your needs for clearing a function control vector from your Coprocessor.

Note: Read the "Code disclaimer information" on page 276 for important legal information.

```

D*****
D* CLEARFCV
D*
D* Clear the Function Control Vector from the card.
D* The Function Control Vector enables the cryptographic
D* functions of the card. Clearing it from the
D* disabled the cryptographic functions.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: None
D*
D* Example:
D* CALL PGM(CLEARFCV)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(CLEARFCV) SRCFILE(SAMPLE)
D* CRTPGM PGM(CLEARFCV) MODULE(CLEARFCV)
D* BNDSRVPGM(QCCA/CSUACFC)
D*
D* Note: Authority to the CSUACFC service program in the
D* QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Facilty_Control (CSUACFC)
D*
D*****
D*-----
D* Declare variables used on CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S         9B 0
D*          ** Exit data
DEXITDATA S           4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY S           16
D*          ** Verb data length
DVERBDATALEN S         9B 0
D*          ** Verb data
DVERBDATA S           16
D*
D*
D*****
D* Prototype for Cryptographic_Facilty_Control (CSUACFC)
D*****

```

```

DCSUACFC          PR
DRETCODE          9B 0
DRSNCODE          9B 0
DEXTDTALEN       9B 0
DEXTDTA          4
DRARRAYCT        9B 0
DRARRAY          16
DVRBDTALEN       9B 0
DVRBDTA          10
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG             S          75  DIM(2) CTDATA PERRCD(1)
DMSGLENGTH       S          9B 0 INZ(75)
D
DMSGTEXT         DS          1    75
DFAILRETC        41        44
DFAILRSNC        46        49
D*          ** Variables required for the QMHSNDPM API
DMESSAGEID       S          7    INZ(' ')
DMESSAGEFILE     S          21   INZ(' ')
DMSGKEY          S          4    INZ(' ')
DMSGTYPE         S          10   INZ('*INFO ')
DSTACKENTRY     S          10   INZ('* ')
DSTACKCOUNTER   S          9B 0 INZ(2)
DERRCODE         DS
DBYTESIN         1         4B 0 INZ(0)
DBYTESOUT        5         8B 0 INZ(0)
D*
C*****
C* START OF PROGRAM *
C* *
C*-----*
C* Set the keyword in the rule array *
C*-----*
C          MOVE    'ADAPTER1'  RULEARRAY
C          MOVE    'CLR-FCV '  RULEARRAY
C          Z-ADD   2           RULEARRAYCNT
C*-----*
C* Set the verb data length to 0 *
C*-----*
C          Z-ADD   0           VERBDATALEN
C*-----*
C* Call Cryptographic Facility Control SAPI
C*-----*
C          CALLP   CSUACFC      (RETURNCODE:
C                               REASONCODE:
C                               EXITDATALEN:
C                               EXITDATA:
C                               RULEARRAYCNT:
C                               RULEARRAY:
C                               VERBDATALEN:
C                               VERBDATA)
C*-----*
C* Check the return code
C*-----*
C          RETURNCODE  IFGT    0
C*          *-----*
C*          * Send a failure message *
C*          *-----*
C          MOVE    MSG(1)      MSGTEXT
C          MOVE    RETURNCODE  FAILRETC
C          MOVE    REASONCODE  FAILRSNC
C          EXSR    SNDMSG
C*

```

```

C           ELSE
C*          *-----*
C*          * Send a Success message *
C*          *-----*
C           MOVE      MSG(2)      MSGTEXT
C           EXSR      SNDMSG
C*
C           ENDIF
C*
C           SETON                               LR
C*
C*****
C* Subroutine to send a message
C*****
C   SNDMSG      BEGSR
C               CALL      'QMHSNDPM'
C               PARM      MESSAGEID
C               PARM      MESSAGEFILE
C               PARM      MSGTEXT
C               PARM      MSGLENGTH
C               PARM      MSGTYPE
C               PARM      STACKENTRY
C               PARM      STACKCOUNTER
C               PARM      MSGKEY
C               PARM      ERRCODE
C           ENDSR
C*

```

```

**
CSUACFC failed with return/reason codes 9999/9999'
The request completed successfully

```

Load and set a master key

After “Load a function control vector” on page 70, you can load and set a master key. The Coprocessor uses the master key to encrypt all operational keys. The master key is a special key-encrypting key stored in the clear (not encrypted) within the Coprocessor secure module. Your Coprocessor uses the master key to encrypt other keys so that you can store those keys outside of your Coprocessor. The master key is a 168-bit key formed from at least two 168-bit parts exclusive ORed together.

Loading a master key

There are three registers for your master keys: New, Current, and Old. The new master key register is used to hold a pending master key while it is being built. It is not used to encrypt any keys. The Current master key register holds the master key that is currently being used to encrypt newly generated/imported/re-enciphered keys. The old master key register holds the previous master key. It is used to recover keys after a master key change has occurred. When you load a master key, the Coprocessor places it into the New master key register. It remains there until you set the master key.

Choose one of these three ways to create and load a master key, based on your security needs:

- Load the first key parts and the subsequent key parts separately to maintain split knowledge of the key as a whole. This is the least secure method, but you can increase security by giving each key part to a separate individual.
- Use random key generation, which will remove any human knowledge of the key. This is the most secure method for loading a master key, but you will need to clone this randomly generated master key into a second Cryptographic Coprocessor in order to have a copy of it.
- Use a pre-existing master key by cloning it from another Coprocessor.

Setting a master key

Setting the master key causes the key in the Current master key register to move to the Old master key register. Then, the master key in the New master key register moves to the Current master key register.

Note: It is vital for retrieval of data encrypted by the master key that you have a backup copy of the master key at all times. For example write it on a piece of paper, and make sure that you store the backup copy with appropriate security precautions. Or, clone the master key to another Coprocessor.

The easiest and fastest way to load and set master keys is to use the Cryptographic Coprocessor configuration web-based utility found off of the iSeries Tasks page at <http://server-name:2001>. The utility includes the Basic configuration wizard that is used when the Coprocessor is in an un-initialized state. If the Cryptographic Coprocessor already has been initialized, then click on **Manage configuration** and then click on **Master keys** to load and set master keys.

If you would prefer to write your own application to load and set master keys, you can do so by using the Master_Key_Process (CSNBMKP) API verb. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both perform the same function.

- “Example: ILE C program for loading a master key into your Cryptographic Coprocessor”
- “Example: ILE RPG program for loading a master key into your Cryptographic Coprocessor” on page 85

Note: If you choose to use one of the program examples provided, change it to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

Re-encrypting keys

When you set a master key, you should re-encrypt all keys that were encrypted under the former master key to avoid losing access to them. You must do this before you change and set the master key.

You can re-encrypt keys in key store by using the Cryptographic Coprocessor configuration web-based utility found off of the iSeries Tasks page at <http://server-name:2001>. The Cryptographic Coprocessor must have already been initialized. Click on “Manage configuration” and then click on either “DES keys” to re-encrypt DES keys, or “PKA keys” to re-encrypt PKA keys.

If you have keys that are not in key store or if you would prefer to write your own application to re-encrypt keys, you can do so by using the Key_Token_Change (CSNBKTC) or PKA_Key_Token_Change (CSNDKTC) API verbs. An example program is provided for your consideration.

- “Example: ILE C program for re-encrypting keys for your Cryptographic Coprocessor” on page 89

Note: If you choose to use the program example provided, change it to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

Example: ILE C program for loading a master key into your Cryptographic Coprocessor: Change this program example to suit your needs for loading a new master key into your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```
/*-----*/
/* Load a new master key on the card. */
/* */
/* */
/* COPYRIGHT 5769-SS1, 5722-SS1 (C) IBM CORP. 1999, 2000 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
```

```

/* provided to you "AS IS". THE IMPLIED WARRANTIES OF          */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE    */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files.                                     */
/*                                                             */
/*                                                             */
/* Parameters:                                                 */
/* OPTION      (FIRST, MIDDLE, LAST, CLEAR, SET)              */
/* KEYPART     (24 bytes entered in hex -> X'01F7C4....')    */
/*             Required for FIRST, MIDDLE, and LAST           */
/* Example:                                                 */
/* CALL PGM(LOAD_KM)                                         */
/*   (FIRST X'0123456789ABCDEFEDCBA98765432100123456789ABCDEF') */
/* Note: This program assumes the device to use is          */
/*       already identified either by defaulting to the CRP01 */
/*       device or by being explicitly named using the       */
/*       Cryptographic_Resource_Allocate verb. Also this    */
/*       device must be varied on and you must be authorized */
/*       to use this device description.                     */
/* Use these commands to compile this program on iSeries:    */
/* ADDLIB LIB(QCCA)                                          */
/* CRTCMOD MODULE(LOAD_KM) SRCFILE(SAMPLE)                  */
/* CRTPGM PGM(LOAD_KM) MODULE(LOAD_KM)                      */
/*       BNDSRVPGM(QCCA/CSNBKMP QCCA/CSNBRNG)                */
/* Note: Authority to the CSNBKMP and CSNBRNG service programs */
/*       in the QCCA library is assumed.                     */
/* The main Common Cryptographic Architecture (CCA) verb used */
/* is Master_Key_Process (CSNBKMP).                          */
/*-----*/

#include "csucincl.h"    /* header file for CCA Cryptographic */
                        /* Service Provider for iSeries    */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR    -1
#define OK       0
#define WARNING  4

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters */
    /*-----*/
    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 2;
    char exit_data[4];
    char rule_array[2][8];
    long rule_array_count = 1;

    /*-----*/

```

```

/* parameters unique to this program */
/*-----*/
char keypart[24];          /* Dummy parm for SET and CLEAR */

/*-----*/
/* Process the parameters */
/*-----*/
if (argc < 2)
{
    printf("Option parameter must be specified.\n");
    return(ERROR);
}

if (argc < 3 && memcmp(argv[1],"CLEAR",5) != 0 &&
    memcmp(argv[1],"SET",3) != 0)
{
    printf("KeyPart parameter must be specified.\n");
    return(ERROR);
}

/*-----*/
/* Set the keywords in the rule array */
/*-----*/
memset(rule_array,' ',8);
memcpy(rule_array,argv[1],
        (strlen(argv[1]) > 8) ? 8 : strlen(argv[1]));

/*-----*/
/* Call Master Key Process SAPI */
/*-----*/
CSNBMKP( &return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char *)rule_array,
        (argc == 3) ? argv[2] : keypart);

/*-----*/
/* Check the return code and display the results */
/*-----*/
if ( (return_code == OK) | (return_code == WARNING) )
{
    printf("Request was successful with return/reason codes: %d/%d \n",
        return_code, reason_code);
    return(OK);
}
else
{
    printf("Request failed with return/reason codes: %d/%d \n",
        return_code, reason_code);
    return(ERROR);
}
}

```

Example: ILE RPG program for loading a master key into your Cryptographic Coprocessor: Change this program example to suit your needs for loading a new master key into your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* LOAD_KM
D*
D* Load a new master key on the card.

```

D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters:
D* OPTION (FIRST, MIDDLE, LAST, CLEAR, SET)
D* KEYPART (24 bytes entered in hex -> X'01F7C4....')
D* Required for FIRST, MIDDLE, and LAST
D*
D* The master key is loaded in 3 or more parts. Specify FIRST
D* when loading the first part, MIDDLE when loading all parts
D* between the first and the last, and LAST when loading the final
D* part of the master key.
D*
D* As the master key parts are entered, they are Exclusively OR'ed
D* with the current contents of the master key register. After the
D* last master key, if the contents do not have odd parity in every
D* byte, a non-zero return/reason code will be returned. In order
D* to ensure that the final result has odd parity, each key part
D* should have odd parity in every byte. This is assuming that there
D* is an odd number of key parts. (If there is an even number of
D* key parts, then one of the key parts should have even parity).
D*
D* A byte has odd parity if it contains:
D* an odd parity nibble : 1, 2, 4, 7, 8, B, D, or E AND
D* an even parity nibble: 0, 3, 5, 6, 9, A, C, or F.
D*
D* For example 32, A4, 1F, and 75 are odd parity bytes because
D* they contain both an odd parity and an even parity
D* nibble.
D*
D* 05, 12, 6C, and E7 are even parity bytes because
D* they contain either two even parity nibbles or
D* two odd parity nibbles.
D*
D* The New master key register must be empty before the first part
D* of a master key can be entered. Use CLEAR to ensure that the
D* New master key register is empty before loading the master key
D* parts.
D*
D* After loading the master key, use SET to move the master key from
D* the New-master-key register to the Current-master-key register.
D* Cryptographic keys are encrypted under the master key in the
D* the Current-master-key register.
D*
D* Example:
D* CALL PGM(LOAD_KM) (CLEAR)
D*
D* CALL PGM(LOAD_KM)
D* (FIRST X'0123456789ABCDEFEDCBA98765432100123456789ABCDEF')
D*
D* CALL PGM(LOAD_KM)

```

D*      (MIDDLE X'1032A873458010F7EF3438373132F1F2F4F8B3CDCDCDCEF1')
D*
D*      CALL PGM(LOAD_KM)
D*      (LAST X'2040806789ABCDEFFEDC3434346432100123456789FEDCBA')
D*
D*      CALL PGM(LOAD_KM) (SET)
D*
D*
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(LOAD_KM) SRCFILE(SAMPLE)
D* CRTPGM PGM(LOAD_KM) MODULE(LOAD_KM)
D*      BNDSRVPGM(QCCA/CSNBKMP)
D*
D* Note: Authority to the CSNBKMP service program in the
D*      QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Master_Key_Process (CSNBKMP)
D*
D*****
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*      ** Return code
DRETURNCODE      S          9B 0
D*      ** Reason code
DREASONCODE      S          9B 0
D*      ** Exit data length
DEXITDATALEN     S          9B 0
D*      ** Exit data
DEXITDATA        S           4
D*      ** Rule array count
DRULEARRAYCNT   S          9B 0
D*      ** Rule array
DRULEARRAY       S          16
D*      ** Option (Rule Array Keyword)
DOPTION          S           8
D*      ** Master key part parameter on program
DMASTERKEYPART  S          24
D*      ** Master key part parameter on CSNBKMP
DKEYPART         S          24  INZ(*ALLX'00')
D*
D*****
D* Prototype for Master_Key_Process (CSNBKMP)
D*****
DCSNBMKP        PR
DRETCODE         9B 0
DRSNCODE         9B 0
DEXTDTALEN       9B 0
DEXTDTA          4
DRARRAYCT        9B 0
DRARRAY          16
DMSTRKEY         24  OPTIONS(*NOPASS)
D*
D*-----
D*      ** Declares for sending messages to the
D*      ** job log using the QMHSDPM API
D*-----
DMSG             S          75  DIM(2) CTDATA PERRCD(1)
DMSGLENGTH       S          9B 0  INZ(75)
D                DS
DMSGTEXT          1          75
DFAILRET         41         44
DFAILRSNC        46         49
DMESSAGEID       S           7  INZ(' ')
DMESSAGEFILE     S          21  INZ(' ')

```

```

DMSGKEY          S          4    INZ('  ')
DMSGTYPE        S          10   INZ('*INFO ')
DSTACKENTRY     S          10   INZ('*  ')
DSTACKCOUNTER   S          9B 0  INZ(2)
DERRCODE        DS
DBYTESIN        1          4B 0  INZ(0)
DBYTESOUT       5          8B 0  INZ(0)
D*
C*****
C* START OF PROGRAM *
C* *
C   *ENTRY      PLIST
C               PARM                OPTION
C               PARM                MASTERKEYPART
C* *
C*-----*
C* Set the keyword in the rule array *
C*-----*
C               MOVEL    OPTION      RULEARRAY
C               Z-ADD    1           RULEARRAYCNT
C* *
C*-----*
C* Check for FIRST, MIDDLE, or LAST *
C*-----*
C   OPTION      IFEQ      'FIRST'
C   OPTION      OREQ      'MIDDLE'
C   OPTION      OREQ      'LAST'
C* *-----*
C* * Copy keypart parameter *
C* *-----*
C               MOVEL    MASTERKEYPART  KEYPART
C               ENDIF
C* *
C*-----*
C* Call Master Key Process SAPI *
C*-----*
C               CALLP    CSNBMP        (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     KEYPART)
C*-----*
C* Check the return code *
C*-----*
C   RETURNCODE  IFGT      0
C* *-----*
C* * Send error message *
C* *-----*
C               MOVE     MSG(1)        MSGTEXT
C               MOVE     RETURNCODE    FAILRETC
C               MOVE     REASONCODE    FAILRSNC
C               EXSR     SNDMSG
C* *
C               ELSE
C* *-----*
C* * Send success message *
C* *-----*
C               MOVE     MSG(2)        MSGTEXT
C               EXSR     SNDMSG
C* *
C               ENDIF
C* *
C               SETON
C* *
C*****

```

LR

```

C* Subroutine to send a message
C*****
C   SNDMSG      BEGSR
C               CALL      'QMHSNDPM'
C               PARM      MESSAGEID
C               PARM      MESSAGEFILE
C               PARM      MSGTEXT
C               PARM      MSGLENGTH
C               PARM      MSGTYPE
C               PARM      STACKENTRY
C               PARM      STACKCOUNTER
C               PARM      MSGKEY
C               PARM      ERRCODE
C               ENDSR
C*

```

```

**
CSNBMPK failed with return/reason codes 9999/9999
The request completed successfully

```

Example: ILE C program for re-encrypting keys for your Cryptographic Coprocessor: Change this program example to suit your needs for re-encrypting keys for your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Description: Re-enciphers key store files using the current */
/*              master key.                                  */
/*              */
/* COPYRIGHT    5769-SS1 (c) IBM Corp 1999                */
/*              */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot    */
/* guarantee or imply reliability, serviceability, or function */
/* of these programs. All programs contained herein are   */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF    */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files.                               */
/*              */
/* Parameters:                                           */
/* char * keysto_type, choices are "DES" or "PKA"        */
/*              (If omitted, the default is "PKA".)      */
/* Examples:                                           */
/* CALL PGM(REN_KEYSTO) PARM(DES)                       */
/* CALL PGM(REN_KEYSTO)                                  */
/*              */
/* Note: The CCA verbs used in the this program are more fully */
/* described in the IBM CCA Basic Services Reference */
/* and Guide (SC31-8609) publication.                   */
/*              */
/* Note: This program assumes the card you want to use is */
/* already identified either by defaulting to the CRP01 */
/* device or has been explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description.                      */
/*              */
/* This program also assumes the key store file you will */
/* use is already identified either by being specified on */
/* the cryptographic device or has been explicitly named */
/* using the Key_Store_Designate verb. Also you must be */
/* authorized to update records in this file.           */
/*              */
/* Use the following commands to compile this program:  */
/* ADDLIB LIB(QCCA)                                     */
/* CRTCMOD MODULE(REN_KEYSTO) SRCFILE(SAMPLE)          */

```

```

/* CRTPGM PGM(REN_KEYSTO) MODULE(REN_KEYSTO) */
/* BNSDRVPGM(QCCA/CSNBKTC QCCĀ/CSNBKRL) */
/* QCCA/CSNDKTC QCCA/CSNDKRL) */
/* */
/* Note: authority to the CSNDKTC, CSNDKRL, CSNBKTC, and CSNBKRL */
/* service programs in the QCCA library is assumed. */
/* */
/* Common Cryptographic Architecture (CCA) verbs used: */
/* PKA_Key_Token_Change (CSNDKTC) */
/* DES_Key_Token_Change (CSNBKTC) */
/* PKA_Key_Record_List (CSNDKRL) */
/* DES_Key_Record_List (CSNBKRL) */
/*-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "csucinl.h" /* header file for CCA Cryptographic
                    Service Provider for iSeries */

/* Define the acceptable file types */
#define PKA 1
#define DES 0

int re_encipher(FILE *key_rec, long rec_length, int key_type);

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard return codes */
    /*-----*/

#define ERROR -1
#define OK 0

    /*-----*/
    /* standard CCA parameters */
    /*-----*/

    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 0;
    char exit_data[2];
    long rule_array_count = 0;
    char rule_array[1][8];

    /*-----*/
    /* fields unique to this sample program */
    /*-----*/
    char key_label[65] =
        "*****";
    long data_set_name_length = 0;
    char data_set_name[65];
    char security_server_name[9] = " ";

    FILE *kr1;
    int keysto_type = PKA;
    /*-----*/
    /* Check whether the user requested to re-encipher a DES or */
    /* a PKA keystore file. Default to PKA if key file type is */
    /* not specified. */
    /*-----*/
    if (argc >= 2)
    {

```



```

if ((strcmp(argv[1],"DES")==0)
{
    printf("\nDES ");
    keysto_type = DES;
}
else if ((strcmp(argv[1],"PKA")==0))
    printf("\nPKA ");
else
{
    printf("\nKeystore type parm incorrectly specified.\n");
    printf("Acceptable choices are PKA or DES.\n");
    printf("The default is PKA.\n");
    return ERROR;
}
}
else
{
    printf("\nPKA ");
}

    if (keysto_type == DES)
    {

/*-----*/
/* Invoke the verb to create a DES Key Record List      */
/*-----*/
CSNBKRL( &return_code,
    &reason_code,
    &exit_data_length,
    exit_data,
    key_label,
    &data_set_name_length,
    data_set_name,
    security_server_name);
    }
    else
    {
/*-----*/
/* Invoke the verb to create a PKA Key Record List      */
/*-----*/
CSNDKRL( &return_code,
    &reason_code,
    &exit_data_length,
    exit_data,
    &rule_array_count,
    (char *) rule_array,
    key_label,
    &data_set_name_length,
    data_set_name,
    security_server_name);
    }

    if ((return_code != 0) || (reason_code != 0))
    {
printf("Key Record List generation was unsuccessful. ");
printf("Return/reason code = %d/%d\n",return_code, reason_code);
    }
    else
    {
printf("Key Record List generation was successful. ");
printf("Return/reason codes = %d/%d\n",return_code, reason_code);
data_set_name[data_set_name_length] = '\0';
printf("data_set_name = %s\n",data_set_name);

/* Open the Key Record List file. */
kr1 = fopen(data_set_name, "rb");

```

```

if (kr1 == NULL) /* Open failed. */
{
    printf("The open of the Key Record List file failed\n");
    return ERROR;
}
else /* Open was successful. */
{
    char header1[77];
    int num_rec, i;
    long rec_length, offset_rec1;

    /* Read the first part of the KRL header. */
    fread(header1,1,77,kr1);

    /* Get the number of key records in the file. */
    num_rec = atoi(&header1[50]);
    printf("Number of key records = %d\n",num_rec);

    /* Get the length for the key records. */
    rec_length = atoi(&header1[58]);

    /* Get the offset for the first key record. */
    offset_rec1 = atoi(&header1[62]);

    /* Set the file pointer to the first key record. */
    fseek(kr1, offset_rec1, SEEK_SET);

    /* Loop through the entries in the KRL and re-encipher. */
    for (i = 1; i <= num_rec; i++)
    {
int result;
result = re_encipher(kr1, rec_length, keysto_type);
if (result !=0)
{
    fclose(kr1);
    return ERROR;
}
    }
    printf("Key store file re-enciphered successfully.\n\n");
    fclose(kr1);
    return OK;
}
}
} /* end of main() */

```

```

int re_encipher(FILE *key_rec, long rec_length, int key_type)
{
    /*-----*/
    /* standard CCA parameters */
    /*-----*/

    long return_code;
    long reason_code;
    long exit_data_length = 0;
    char exit_data[2];
    long rule_array_count = 1;
    char rule_array[1][8];

    /*-----*/
    /* fields unique to this function */
    /*-----*/
    long key_identifier_length = 64;
    char key_identifier[64];

```

```

char key_record[154];

fread(key_record, 1, rec_length, key_rec);
memcpy(key_identifier, &key_record[3], 64);
memcpy(rule_array, "RTCMK",8);

if (key_type == DES)
{
CSNBKTC(&return_code,
&reason_code,
&exit_data_length,
exit_data,
&rule_array_count,
(char *) rule_array,
key_identifier);
}
else if (key_type == PKA)
{
CSNDKTC(&return_code,
&reason_code,
&exit_data_length,
exit_data,
&rule_array_count,
(char *) rule_array,
&key_identifier_length,
key_identifier);
}
else
{
printf("re_encipher() called with an invalid key type.\n");
return ERROR;
}

printf("Re-enciphering for key_label = %.64s",key_identifier);
printf("completed with return/reason codes of ");
printf("%d/%d\n",return_code,reason_code);
return return_code;
}

}/* end of re_encipher() */

```

Configure the Cryptographic Coprocessor for use with DCM and SSL

The following section lists the steps needed to make the Cryptographic Coprocessor ready for use with SSL.

Using your Coprocessor with DCM and SSL

To install the Cryptographic Coprocessor and prerequisite software, you must do the following:

- Install the Coprocessor in your server.
 - For feature 4806, install your Cryptographic Coprocessor, as instructed in the PCI Cryptographic Coprocessor Card Instructions that are shipped with your Cryptographic Coprocessor.
- Install i5/OS Option 35 CCA CSP.
- Install the 5722-AC3 Cryptographic Access Provider 128-bit licensed program product.
- For Feature 4806, install the 5722-CY1 Cryptographic Device Manager product.
- Set i5/OS object authorities for “Secure access” on page 17.
- Use your web browser to go to the iSeries Tasks page at <http://server-name:2001>.
- Configure the Coprocessor by following the steps in “Configure the Cryptographic Coprocessor” on page 20.

The Cryptographic Coprocessor is now ready to be used to create private keys for SSL certificates.

- Use DCM to create a certificate, specifying that the private key be generated by the hardware.

- Use DCM to receive the signed certificate.

See Manage public Internet certificates for SSL communications sessions for more information on these last two steps.

Note: If you plan to use multiple cards for SSL, see “Manage multiple Cryptographic Coprocessors” on page 175 and “Clone master keys” on page 185.

Configure the Cryptographic Coprocessor for use with i5/OS applications

The following section lists the steps needed to make Cryptographic Coprocessors ready for use with an i5/OS or OS/400 application. See Cryptographic hardware scenario: Write a i5/OS or OS/400 application to use the Cryptographic Coprocessor for an example usage of the Cryptographic Coprocessor in an i5/OS or OS/400 application.

Using the Cryptographic Coprocessor for OS/400 applications

To install the Cryptographic Coprocessor and prerequisite software, you must do the following:

- Install the Coprocessor in your server.
For feature 4806, install your Cryptographic Coprocessor, as instructed in the PCI Cryptographic Coprocessor Card Instructions that are shipped with your Cryptographic Coprocessor.
- Install i5/OS or OS/400 Option 35 CCA CSP.
- Install either the 5722-AC3 Cryptographic Access Provider 128-bit or the 5722-AC2 Cryptographic Access Provider 56-bit licensed program product.
- For Feature 4806, install the 5722-CY1 Cryptographic Device Manager product.
- Set i5/OS object authorities for “Secure access” on page 17.
- Use your web browser to go to the iSeries Tasks page at <http://server-name:2001>.
- Configure the Coprocessor by following the steps in “Configure the Cryptographic Coprocessor” on page 20.
- Write your application to use the Cryptographic Coprocessor.

Note: If you plan to use multiple cards for your i5/OS or OS/400 applications, see “Manage multiple Cryptographic Coprocessors” on page 175.

Migrate to the Cryptographic Coprocessor

If you have worked with iSeries cryptography before, you may have a requirement to migrate from a previous iSeries cryptography product to the 4764 or 4758 Cryptographic Coprocessor.

Note: If you are replacing your 4758 Cryptographic Coprocessor with the newer 4764 Cryptographic Coprocessor, then ensure that the roles profiles and access controls for the 4764 Coprocessor are setup similar to those used with the 4758 Coprocessor. The CCA APIs and key store files used with the 4758 are compatible with those of the 4764 Coprocessor.

You may have cryptographic cross-domain files from Cryptographic Support for iSeries (5769-CR1). Or you may have key store files from the IBM CCA Services for iSeries (5799-FRF) product. If this is the case, you can migrate their contents to your new Cryptographic Coprocessor. There is an example migration program available for each cryptographic product:

- **Cryptographic Support for AS/400 (5769-CR1 or 5722-CR1).** Cryptographic Support is a software-only product that encrypts cross-domain keys under a host master key. Cryptographic Support then stores the cross-domain keys in a file. You can migrate cross-domain key files from Cryptographic Support for AS/400 to your Cryptographic Coprocessor using “Migrate Cryptographic Support for AS/400 cross-domain key files” on page 95.
- **IBM CCA Services for iSeries (5799-FRF) PRPQ.** This product provides cryptographic function on cryptographic hardware by using Data Encryption Standard (DES). The CCA Services PRPQ requires

that you have a cryptographic processor, hardware feature number 2620 or 2628, installed on your server. You can migrate key store files from the IBM CCA Services to your Cryptographic Coprocessor using “Migrate key store files from the IBM CCA Services for OS/400 PRPQ” on page 110.

Migrate Cryptographic Support for AS/400 cross-domain key files

If you have worked with cryptography before on your server, you may have cryptographic cross-domain files from Cryptographic Support for iSeries (5769–CR1). You can migrate existing cross-domain keys to your new cryptographic coprocessor.

The Cryptographic Support for iSeries product (5769–CR1 or 5722–CR1) encrypts its cross-domain keys under the host master key and stores them in a file. Common Cryptographic Architecture (CCA) cannot use them in this form, but you can migrate them from the Cryptographic Support product for the CCA to use with your Coprocessor. You must consider a number of things before completing this task:

- **Encryption of cross-domain keys by cross-domain keys.** Cryptographic Support for iSeries supports importing clear key values for cross-domain keys and encrypting data keys under cross-domain keys. However, it does not support encrypting cross-domain keys under cross-domain keys, nor does it support returning the clear key value of any cross-domain key. Because of this, migrating cross-domain keys is considerably more involved than just performing an export and import operation.
- **Single-length keys versus double-length keys.** All keys in Cryptographic Support for iSeries are single-length keys. In CCA, all key-encrypting keys and PIN keys are double-length keys. Although the key lengths are different, you can build a double-length key from a single-length key and have that double-length key behave like the single-length key. If both halves of a double-length key are the same, the result of any encryption operation will be the same as if a single-length key was used. Therefore, when you migrate keys from Cryptographic Support for iSeries to CCA, you will need to copy the key value of the cross-domain key into both halves of the key value for a CCA key.
- **CCA control vectors versus master key variants.** In CCA, when a key is said to be encrypted under a key-encrypting key, it is really encrypted under a key that is formed by an exclusive OR operation of the key-encrypting key and a control vector. For Cryptographic Support, cross-domain keys are encrypted under one of three different master key variants. A master key variant is the result of the exclusive OR operation of the host master key with either 8 bytes of hexadecimal 22, 44, or 88. Both control vectors and master key variants provide key separation and thereby restrict keys to their intended use. In CCA, the value of the control vector determines its use. In Cryptographic Support how a key is used determines which master key variant will be used to decrypt it. In both cases, any attempt to use the key for other than its intended use will result in an error. Although control vectors and master key variants may work similarly, the values used to form master key variants are not the same as control vectors.
- **Asymmetry of CCA control vectors for double-length keys.** Double-length keys behave like single-length keys only when both halves of the double-length key are identical. Control vectors for double-length keys are asymmetric. Any double-length key that is exclusive ORed with a control vector will not result in a key with identical halves. This double-length key will not behave like a single length key.

You can choose one of two methods for migrating the keys.

Method 1 (Recommended)

This method provides some solutions to the considerations listed above and is the recommended method to use.

To migrate the cross-domain keys from Cryptographic Support to CCA, you will need to use a key-encrypting key that is common to both. You can use the Cryptographic Support host master key as the common key between Cryptographic Support and CCA (in CCA, the host master key is known as the master key). Import the Cryptographic Support host master key clear value into CCA as an IMPORTER key-encrypting key. Because you enter the host master key in two separate parts, you should consider importing it into CCA as two parts using the Key_Part_Import (CSNBKPI) CCA API. If you had dual

| responsibility for the Cryptographic Support host master key, you should maintain this dual
| responsibility for this key-encrypting key. Alternatively, if you know both parts of the host master key,
| you could also perform an exclusive OR of the two parts and import the key in just one part. The
| program example uses this method of importing the host master key. You may want to consider
| importing the host master key in a completely separate process instead of combining it with the
| migration of all cross-domain keys like the program example does.

| There are three types of cross-domain keys:

- | • Receiving cross-domain keys
- | • Sending cross-domain keys
- | • PIN cross-domain keys

| The CCA equivalent of receiving cross-domain keys are IMPORTER key-encrypting keys. Both are used
| for receiving or importing an encrypted key.


| Sending-cross-domain keys are used for both a) encrypting data keys, which can then be sent to another
| system, and b) translating encrypted personal identification numbers (PIN). CCA has stricter key
| separation than the Cryptographic Support product, so you cannot generate or import a key that
| provides both functions. If the key is used as both an EXPORTER key-encrypting key and an OPINENC
| (outbound PIN encrypting) key, you need to import sending-cross-domain keys twice into two different
| keys with two different key types.

| You may use PIN-cross-domain keys for generating PINs and verifying PINs. CCA separates these two
| usage's into PINGEN (PIN generation) and PINVER (PIN verification) keys. If the key is used for both
| generating and verifying PINs, you need to import PIN-cross-domain keys twice as well.

| While the host master key encrypts data keys, different master key variants encrypt cross-domain keys.

- | • Master key variant 1 encrypts sending cross-domain keys. Variant 1 is the result of an exclusive-OR
| operation of the host master key with 8 bytes of hexadecimal 88.
- | • Master key variant 2 encrypts receiving cross-domain keys. Variant 2 is the result of an exclusive-OR
| operation of the host master key and 8 bytes of hexadecimal 22.
- | • Master key variant 3 encrypts PIN cross-domain keys. Variant 3 is the result of an exclusive-OR
| operation of the host master key and 8 bytes of hexadecimal 44.

| **Note:** If you were to only import the clear key value of the host master key into CCA, you would not be
| able to migrate any keys. You need to factor in which master key variant encrypts the key in order to
| migrate it.

| The 8 byte values for creating master key variants are analogous to control vectors. The process of
| migrating keys can be thought of as changing control vectors on a key. The IBM PCI Cryptographic
| Coprocessor CCA Basic Services Reference and Guide  describes a method for such a process. The
| method is the pre-exclusive-OR technique. If the clear key value of a key-encrypting key (the host master
| key in this case) is exclusive-ORed with control vector information before importing the key, you can
| effectively change the control vector for any key that this key-encrypting key imports.

| The pre-exclusive-OR technique works well if you are working with single-length keys. For double-length
| keys, the technique must be changed because the control vector for the right half of a CCA key is
| different than the control vector for the left half. To overcome this difference, import the key twice, as
| follows:

- | 1. Create a 16 byte value such that each 8 byte half is identical to the left half of the control vector of the
| key you want to import. Use this 16 byte value in the pre-exclusive-OR technique to create an
| importer key-encrypting key that you can refer to as the "left-importer." Only the left half of keys that
| are imported using this key-encrypting key will be valid.

2. Create another 16 byte value such that each 8 byte half is identical to the right half of the control vector of the key you want to import. Use this 16 byte value in the pre-exclusive-OR technique to create an importer key-encrypting key. Using this importer key-encrypting key, only the right half of the keys that are imported will be valid
3. Import the cross-domain twice:
 - a. First use the key-encrypting key created in step 1 and save the left half of the result.
 - b. Then use the key-encrypting key created in step 2 and save the right half of the result.
4. In the final step, concatenate the left half of the result from step A with the right half of the result from step B. Place the combined results in a new key token.

You now have a CCA double-length key that behaves like the cross-domain key from the Cryptographic Support for iSeries product.

“Using IMPORTER key-encrypting keys” on page 109 summarizes all of the importer key-encrypting keys that are needed to import all of the cross-domain keys. It also describes how to create the importer key-encrypting keys.

Method 2

Note: You should only use this method if you feel comfortable with the security of your system and environment. This method is easier than the recommended method, but it presents a greater security risk for your cross-domain key files, since the cross-domain keys will be in clear form in application storage.

1. Import the host master key into CCA as a data key by using the Clear_Key_Import (CSNBCKI) CCA API. Remember to perform an exclusive OR operation on the key with the values needed to produce data keys equivalent to the master key variants as follows:
 - Master key variant 1 encrypts sending cross-domain keys. Variant 1 is the result of an exclusive-OR operation of the host master key with 8 bytes of hexadecimal 88.
 - Master key variant 2 encrypts receiving cross-domain keys. Variant 2 is the result of an exclusive-OR operation of the host master key and 8 bytes of hexadecimal 22.
 - Master key variant 3 encrypts PIN cross-domain keys. Variant 3 is the result of an exclusive-OR operation of the host master key and 8 bytes of hexadecimal 44.

You will have 3 different data keys after this step.

2. Use the Decrypt (CSNBDEC) CCA API to decrypt the cross-domain keys to return the clear key values. Use the correct data key to decrypt it.
3. Use the Key_Part_Import (CSNBKPI) CCA API to import the clear key into CCA.

You should not consider this method to be secure. All of the keys will have been in clear form in application storage at some time during this method.

Congratulations! You are now qualified to write a program to migrate cross-domain keys, or you can change the following program example to suit your needs for migrating Cryptographic Support for iSeries cross-domain key files to your Cryptographic Coprocessor.



```

/*****/
/* This program migrates keys stored in the file QACRKTBL in library */
/* QUSRSYS to key storage for Option 35 - CCA Cryptographic Service */
/* Provider. The QACRKTBL file contains cross domain keys that are */
/* used for the Cryptographic Support licensed program, 5769-CR1. */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */

```



```

| /* tested under all conditions. IBM, therefore, cannot */
| /* guarantee or imply reliability, serviceability, or function */
| /* of these program. All programs contained herein are */
| /* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
| /* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
| /* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
| /* these programs and files. */
| /* */
| /* */
| /* The keys are migrated by the following steps: */
| /* */
| /* 1 - The master key used for 5769-CR1 passed as a parameter. */
| /* 2 - Build importer keys using the master key, 8 bytes of a mask */
| /* to create a variant, and a control vector. */
| /* 3 - The file QACRKTBL is opened for input. */
| /* 4 - A record is read. */
| /* 5 - Import the key using the pre-exclusive OR process. CCA uses */
| /* control vectors while non-CCA implementations don't. 5769-CR1*/
| /* creates master key variants similar to what 4700 finance */
| /* controllers do. Since the control vector and master key */
| /* variant material affect how the key is enciphered, the pre- */
| /* exclusive OR process "fixes" the importer key so that it can */
| /* correctly import a key. */
| /* - *SND keys are imported twice as an EXPORTER and OPINENC keys. */
| /* - *PIN keys are imported twice as a PINGEN and IPINENC keys. */
| /* - *RCV keys are imported as a IMPORTER key. */
| /* 6- A key record is created with a similar name as in QACRKTBL. */
| /* For key names longer than 8 characters, a '.' will be */
| /* inserted between the 8th and 9th characters. Also a 1 byte */
| /* extension is appended that describes the key type. */
| /* For example, MYKEY *RCV ----> MYKEY.R */
| /* MYKEK00001 *RCV ----> MYKEK000.01.R */
| /* */
| /* For *SND and *PIN keys, a second key record is also created. */
| /* For example, MYKEY *SND ----> MYKEY.S */
| /* MYKEY.O */
| /* MYPINKEY *PIN ----> MYPINKEY.P */
| /* MYPINKEY.I */
| /* */
| /* 7 - The key is written out to key store. */
| /* */
| /* 8 - Steps 4 through 7 are repeated until all keys have been */
| /* migrated. */
| /* */
| /* */
| /* Note: Input format is more fully described in Chapter 2 of */
| /* IBM CCA Basic Services Reference and Guide */
| /* (SC31-8609) publication. */
| /* */
| /* Parameters: */
| /* nonCCA master key - 8 bytes */
| /* */
| /* Example: */
| /* CALL PGM(MIGRATECR) PARM(X'1C23456789ABCDEF') */
| /* */
| /* Note: This program assumes the device to be used is */
| /* already identified either by defaulting to the CRP01 */
| /* device or by being explicitly named using the */
| /* Cryptographic_Resource_Allocate verb. Also this */
| /* device must be varied on and you must be authorized */
| /* to use this device description. */
| /* */
| /* */
| /* Use these commands to compile this program on iSeries: */

```



```

| /* ADDLIB LIB(QCCA) */
| /* CRTCMOD MODULE(MIGRATECR) SRCFILE(SAMPLE) */
| /* CRTPGM PGM(MIGRATECR) MODULE(MIGRATECR) */
| /* BNSDRVPGM(QCCA/CSNBKIM QCCA/CSNBKPI QCCA/CSNBKRC */
| /* QCCA/CSNBDEC QCCA/CSNBKRW) */
| /* */
| /* Note: Authority to the CSNBKIM, CSNBKPI, CSNBKRC, and CSNBKRW */
| /* service programs in library QCCA is assumed. */
| /* */
| /* */
| /* The Common Cryptographic Architecture (CCA) verbs used are: */
| /* */
| /* Key_Import (CSNBKIM) */
| /* Key_Part_Import (CSNBKPI) */
| /* Key_Record_Create (CSNBKRC) */
| /* Key_Record_Write (CSNBKRW) */
| /* */
| /* */
| /*****/
|
| /*****/
| /* Retrieve various structures/utilities that are used in program. */
| /*****/
|
| #include <stdio.h> /* Standard I/O header. */
| #include <stdlib.h> /* General utilities. */
| #include <stddef.h> /* Standard definitions. */
| #include <string.h> /* String handling utilities. */
| #include "miptrnam.h" /* MI templates for pointer */
| /* resolution instructions. */
| #include "csucincl.h" /* Header file for security API */
|
| /*****/
| /* Declare function prototype to build tokens to import keys */
| /*****/
| int buildImporter(char * token,
| char * clearkey,
| char * preXORcv,
| char * variant);
|
| /*****/
| /* Declare function prototype to import a non-CCA key and put it */
| /* into key store. */
| /*****/
| int importNonCCA(char * label,
| char * left_importer,
| char * right_importer,
| char * cv,
| char * encrypted_key);
|
| /*****/
| /* Declares for working with files */
| /*****/
| #include <xxfdbk.h> /* Feedback area structures. */
| #include <recio.h> /* Record I/O routines */
| _RFILE *dbfptr; /* Pointer to database file. */
| _RIOFB_T *db_fdbk; /* I/O Feedback - data base file */
| _XXOPFB_T *db_opfb;
|
| /*****/
| /* Define the record for cross domain key file QACRKTBL */
| /*****/
| struct
| {
| char label[10];
| char key_type;

```

```

|         char    key_value[8];
|     } key_rec;
|
| /*****
| /* Define the structure for key tokens
| */
| /*****
| typedef struct
| {
|     char    tokenType;
|     char    reserved1;
|     char    MasterKeyVerifPattern[2];
|     char    version;
|     char    reserved2;
|     char    flagByte1;
|     char    flagByte2;
|     char    reserved3[8];
|     char    leftHalfKey[8];
|     char    rightHalfKey[8];
|     char    controlVectorBase[8];
|     char    rightControlVector[8];
|     char    reserved4[12];
|     char    tvv[4];
| } key_token_T;
|
| /*****
| /* Declare control vectors used for building keys
| */
| /*****
| char        pingenc_cv[16] = { 0x00, 0x22, 0x7E, 0x00,
|                                0x03, 0x41, 0x00, 0x00,
|                                0x00, 0x22, 0x7E, 0x00,
|                                0x03, 0x21, 0x00, 0x00};
|
| char        ipinenc_cv[16] = { 0x00, 0x21, 0x5F, 0x00,
|                                0x03, 0x41, 0x00, 0x00,
|                                0x00, 0x21, 0x5F, 0x00,
|                                0x03, 0x21, 0x00, 0x00};
|
| char        opinenc_cv[16] = { 0x00, 0x24, 0x77, 0x00,
|                                0x03, 0x41, 0x00, 0x00,
|                                0x00, 0x24, 0x77, 0x00,
|                                0x03, 0x21, 0x00, 0x00};
|
| char        importer_cv[16] = { 0x00, 0x42, 0x7D, 0x00,
|                                0x03, 0x41, 0x00, 0x00,
|                                0x00, 0x42, 0x7D, 0x00,
|                                0x03, 0x21, 0x00, 0x00};
|
| char        exporter_cv[16] = { 0x00, 0x41, 0x7D, 0x00,
|                                0x03, 0x41, 0x00, 0x00,
|                                0x00, 0x41, 0x7D, 0x00,
|                                0x03, 0x21, 0x00, 0x00};
|
| char        importer_cv_part[16] = { 0x00, 0x42, 0x7D, 0x00,
|                                    0x03, 0x48, 0x00, 0x00,
|                                    0x00, 0x42, 0x7D, 0x00,
|                                    0x03, 0x28, 0x00, 0x00};
|
| char        exporter_cv_part[16] = { 0x00, 0x41, 0x7D, 0x00,
|                                    0x03, 0x48, 0x00, 0x00,
|                                    0x00, 0x41, 0x7D, 0x00,
|                                    0x03, 0x28, 0x00, 0x00};
|
| /*****
| /* Start of mainline code.
| */
| /*****
| int main(int argc, char *argv[])
| {

```

```

| long          i,j,k;          /* Indexes for loops          */
| char          key_label[64];   /* label of new key          */
| char          key_label1[64];  /* label of new key          */
|
| /*****
| /* Declare importer keys - two keys are needed for each type */
| /*****/
| char          EXPORTER_importerL[64];
| char          EXPORTER_importerR[64];
| char          OPINENC_importerL[64];
| char          OPINENC_importerR[64];
| char          IMPORTER_importerL[64];
| char          IMPORTER_importerR[64];
| char          PINGEN_importerL[64];
| char          PINGEN_importerR[64];
| char          IPINENC_importerL[64];
| char          IPINENC_importerR[64];
|
| /*****
| /* Declare variables to hold bit strings to generate master key */
| /* variants.                                                    */
| /*****/
| char          variant1[16];
| char          variant2[16];
| char          variant3[16];
|
| /*****
| /* Build the key tokens for each of the importer keys using    */
| /* Key-Token_Build. Each key is built by using a variant, a control */
| /* vector, and the clear key. Master key variant 1 is the result of */
| /* an exclusive OR of the master key with hex '8888888888888888',    */
| /* Master key variant 2 is the result of an exclusive OR of the    */
| /* master key with hex '2222222222222222', and Master key variant 3  */
| /* is the result of an exclusive OR of the master key with hex    */
| /* '4444444444444444'. During the import operation, the control    */
| /* vector is exclusive OR'ed with the importer key. The effect of    */
| /* the control vector is overcome by including the control vector as */
| /* key part. Then when the import operation is done, the exclusive  */
| /* OR operation will result in the original key. For double keys,    */
| /* the left and right half of the control vector is not the same and */
| /* therefore, XORing with the control vector will not result in the  */
| /* original key - only one half of it will be valid. So two keys are*/
| /* needed - one for each half.                                       */
| /*****/
|     memset(variant1, 0x88, 16);
|     memset(variant2, 0x22, 16);
|     memset(variant3, 0x44, 16);
|
|     if (buildImporter(EXPORTER_importerL, argv[1],
|                       exporter_cv, variant1)    ||
|
|         buildImporter(EXPORTER_importerR, argv[1],
|                       &exporter_cv[8], variant1) ||
|
|         buildImporter(IMPORTER_importerL, argv[1],
|                       importer_cv, variant2)    ||
|
|         buildImporter(IMPORTER_importerR, argv[1],
|                       &importer_cv[8], variant2) ||
|
|         buildImporter(PINGEN_importerL, argv[1],
|                       pingenc_cv, variant3)     ||
|
|         buildImporter(PINGEN_importerR, argv[1],
|                       &pingenc_cv[8], variant3) ||

```

```

    buildImporter(IPINENC_importerL, argv[1],
                  ipinenc_cv, variant3)      ||

    buildImporter(IPINENC_importerR, argv[1],
                  &ipinenc_cv[8], variant3) ||

    buildImporter(OPINENC_importerL, argv[1],
                  opinenc_cv, variant1)     ||

    buildImporter(OPINENC_importerR, argv[1],
                  &opinenc_cv[8], variant1)

    {
        printf("An error occured creating the importer keys\n");
        return;
    }

/*****
/* Open database file.
*****/

/* Open the input file.
/* If the file pointer,
/* dbfptr is not NULL,
/* then the file was
/* successfully opened.
if (( dbfptr = _Ropen("QUSRSYS/QACRKTBL", "rr riofb=n")
    != NULL)
{
    db_opfb = _Ropnfbk( dbfptr );
/* Get pointer to the
/* File open feedback
/* area.

    j = db_opfb->num_records;
/* Save number of records*/

/*****
/* Read keys and migrate to key storage.
*****/
for (i=1; i<=j; i++)
{
    db_fdbk = _Rreadn(dbfptr, &key_rec,
                    sizeof(key_rec), __DFT);

/*****
/* Generate a key label for the imported keys.
/* The key label will be similar to the label that was used for
/* the QACRKTBL file. If the label is longer than 8 characters,
/* then a period '.' will be inserted at position 8 to make it
/* conform to label naming conventions for CCA. Also one
/* one character will be added to the end to indicate what type
/* of key. 5769-CR1 does not require unique key names across all
/* key types. CCA requires unique labels for all keys.
*****/
    memset((char *)key_label, ' ',64);
/* Initialize key label
/* to all blanks.

    /* Copy first bytes of label
    memcpy((char *)key_label,(char *)key_rec.label,8);

    /* If label is longer than 8 characters, add a second element*/
    if (key_rec.label[8] != ' ')
    {
        key_label[8] = '.';
        key_label[9] = key_rec.label[8];
        key_label[10] = key_rec.label[9];

```

```

    }

    /* *SND keys and *PIN keys need to be imported twice so      */
    /* make a second label                                       */
    if (key_rec.key_type != 'R')
        memcpy((char *)key_label1,(char *)key_label,64);

    /* Add keytype to label name. Search until a space is found */
    /* and if less than 8, add the 1 character keytype. If it   */
    /* is greater than 8, add a second element with the keytype */
    /* 'R' is *RCV key, 'S' is *SND key, 'P' is *PIN key,      */
    /* 'I' is an IPINENC key and 'O' is OPINENC key            */
    for (k=1; k<=11; k++)
    {
        if (key_label[k] == ' ')
        {
            if (k != 8)
            {
                key_label[k] = key_rec.key_type;

                /* If this is a *SND or *PIN key, update the keytype */
                /* in the second label as well                          */
                if (key_rec.key_type != 'R')
                {
                    memcpy((char *)key_label1,(char *)key_label,64);
                    if (key_rec.key_type == 'S')
                        key_label1[k] = 'O';
                    else
                        key_label1[k] = 'I';
                }
            }
            else
            {
                key_label[8] = '.';
                key_label[9] = key_rec.key_type;

                /* If this is a *SND or *PIN key, update the keytype */
                /* in the second label as well                          */
                if (key_rec.key_type != 'R')
                {
                    memcpy((char *)key_label1,(char *)key_label,64);
                    if (key_rec.key_type == 'S')
                        key_label1[9] = 'O';
                    else
                        key_label1[9] = 'I';
                }
            }
        }
        k = 11;
    }
}

/*****
/* Check for the type of key that was in the QACRKTBL file      */
/* - S for SENDER key will become two keys - EXPORTER and OPINENC*/
/* - R for RECEIVER key will become IMPORTER key                */
/* - P for PIN will become two keys - PINGEN and IPINENC        */
/* Set the key id to the key token that contains the key under  */
/* which the key in QACRKTBL is enciphered.                    */
/* Set the key_type SAPI parameter for the Secure_Key_Import verb*/
*****/
if (key_rec.key_type == 'S')
{
    /* Import the exporter key      */
    if(importNonCCA(key_label,
                    EXPORTER_importerL,
                    EXPORTER_importerR,

```

```

                exporter_cv,
                key_rec.key_value))
        {
            printf("An error ocured importing an exporter key\n");
            break;
        }

        /* Import the OPINENC key      */
        if (importNonCCA(key_labell,
            OPINENC_importerL,
            OPINENC_importerR,
            opinenc_cv,
            key_rec.key_value))
        {
            printf("An error ocured importing an opinenc key\n");
            break;
        }
    }
else
if (key_rec.key_type == 'R')
    {
        /* Import the importer key      */
        if (importNonCCA(key_label,
            IMPORTER_importerL,
            IMPORTER_importerR,
            importer_cv,
            key_rec.key_value))
        {
            printf("An error ocured importing an importer key\n");
            break;
        }
    }
else
    {
        /* Import the PINGEN key      */
        if(importNonCCA(key_label,
            PINGEN_importerL,
            PINGEN_importerR,
            pingenc_cv,
            key_rec.key_value))
        {
            printf("An error ocured importing a PINGEN key\n");
            break;
        }

        /* Import the IPINENC key      */
        if(importNonCCA(key_labell,
            IPINENC_importerL,
            IPINENC_importerR,
            ipinenc_cv,
            key_rec.key_value))
        {
            printf("An error ocured importing an ipinenc key\n");
            break;
        }
    }
    }

        /* End loop repeating for each record */

/*****
/* Close database file.
/*****
if (dbfptr != NULL)
    _Rclose(dbfptr);
    /* Close the file.
    /* End if file open leg

```



```

|     CSNBKPI( &return_code, &reason_code, &exit_data_length,
|             (char *) exit_data,
|             (long *) &rule_array_count,
|             (char *) rule_array,
|             (char *) ctl_vector,
|             (char *) token);
|
|     if (return_code > 4)
|     {
|         printf("Key_Part_Import failed with return/reason codes \
|                %d/%d \n",return_code, reason_code);
|         return 1;
|     }
|
| /*****
| /* Import the variant as a key part using Key_Part_Import      */
| /*****
|     memcpy(rule_array, "MIDDLE ", 8);
|     CSNBKPI( &return_code, &reason_code, &exit_data_length,
|             (char *) exit_data,
|             (long *) &rule_array_count,
|             (char *) rule_array,
|             (char *) variant,
|             (char *) token);
|
|     if (return_code > 4)
|     {
|         printf("Key_Part_Import failed with return/reason codes \
|                %d/%d \n",return_code, reason_code);
|         return 1;
|     }
|
| /*****
| /* Import the clear key as a key part using Key_Part_Import   */
| /*****
|     memcpy(keyvalue, clearkey, 8);
|     memcpy(&keyvalue[8], clearkey, 8); /* Make key double length*/
|     memcpy(rule_array, "LAST ", 8);
|     CSNBKPI( &return_code, &reason_code, &exit_data_length,
|             (char *) exit_data,
|             (long *) &rule_array_count,
|             (char *) rule_array,
|             (char *) keyvalue,
|             (char *) token);
|
|     if (return_code > 4)
|     {
|         printf("Key_Part_Import failed with return/reason codes \
|                %d/%d \n",return_code, reason_code);
|         return 1;
|     }
|
|     return 0;
| }
|
| /*****
| /* importNonCCA imports a double length key into CCA from the  */
| /* non-CCA implementation                                     */
| /*****
| int importNonCCA(char * label,
|                 char * left_importer,
|                 char * right_importer,
|                 char * cv,
|                 char * encrypted_key)
| {
| /*****

```



```

| /* Declare variables used by the SAPIs */
| /*****/
| long      return_code, reason_code;
| char      exit_data[4];
| long      exit_data_length;
| long      rule_array_count;
| char      rule_array[24];
| char      keytoken[64];
| char      externalkey[64];
| char      keyvalue[16];
| char      keytype[8];
| char      *importer;
| char      mkvp[2];
| key_token_T *token_ptr;
| int       tvv, tvv_part;
| char      *tvv_pos;
|
| /*****/
| /* Build an external key token to IMPORT from */
| /*****/
|     memset((void *)externalkey, '\00', 64);
|     token_ptr = (key_token_T *)externalkey;
|     token_ptr->tokenType = 0x02;           /* 02 is external token */
|     token_ptr->version = 0x00;           /* Version 0 token */
|     token_ptr->flagByte1 = 0xC0;        /* High order bit is 1 so */
|                                           /* key is present. The */
|                                           /* 40 bit means that CV */
|                                           /* is present */
|
|     memcpy(token_ptr->controlVectorBase, cv, 16); /* Copy control
|                                                    vector into token */
|     memcpy(token_ptr->leftHalfKey, encrypted_key, 8); /* Copy key
|                                                    into left half */
|     memcpy(token_ptr->rightHalfKey, encrypted_key, 8); /* Copy key
|                                                    into right half */
|
| /*****/
| /* Calculate the TVV by adding every 4 bytes */
| /*****/
|     tvv_pos = externalkey;
|     tvv = 0;
|     while (tvv_pos < (externalkey + 60))
|     {
|         memcpy((void*)&tvv_part, tvv_pos, 4);
|         tvv += tvv_part;
|         tvv_pos += 4;
|     }
|     memcpy(token_ptr->tvv, (void*)&tvv, 4);
|
| /*****/
| /* Import the left half of the key using Key_Import and */
| /* the importer built with left half of the control vector */
| /*****/
|     exit_data_length = 0;
|     memcpy(keytype, "TOKEN ", 8);
|     memset((void *)keytoken, '\00', 64);
|     CSNBKIM( &return_code, &reason_code, &exit_data_length,
|             (char *) exit_data,
|             (char *) keytype,
|             (char *) externalkey,
|             (char *) left_importer,
|             (char *) keytoken);
|
|     if (return_code > 4)
|     {
|         printf("Key_Import failed with return/reason codes \

```

```

|                                     %d/%d \n",return_code, reason_code);
|     return 1;
| }
|
| /*****
| /* Save left half of key out of key token */
| *****/
|     memcpy(keyvalue, &keytoken[16], 8);
|
| /*****
| /* Import the right half of the key using Key_Import and */
| /* the importer built with right half of the control vector*/
| *****/
|     memcpy(keytype, "TOKEN ", 8);
|     memset((void *)keytoken,'\00',64);
|     CSNBKIM( &return_code, &reason_code, &exit_data_length,
|             (char *) exit_data,
|             (char *) keytype,
|             (char *) externalkey,
|             (char *) right_importer,
|             (char *) keytoken);
|
|     if (return_code > 4)
|     {
|         printf("Key_Import failed with return/reason codes \
|                                     %d/%d \n",return_code, reason_code);
|         return 1;
|     }
|
| /*****
| /* Save right half of key out of key token */
| *****/
|     memcpy(&keyvalue[8], &keytoken[24], 8);
|
| /*****
| /* Get master key verification pattern from the last key token built */
| *****/
|     mkvp[0] = keytoken[2];
|     mkvp[1] = keytoken[3];
|
| /*****
| /* Build an internal key token using both key halves just */
| /* imported and using the master key verification pattern */
| *****/
|     memset((void *)keytoken,'\00',64);
|     exit_data_length = 0;
|     token_ptr = (key_token_T *)keytoken;
|     token_ptr->tokenType = 0x01;           /* 01 is internal token */
|     token_ptr->version = 0x03;           /* Version 3 token */
|     token_ptr->flagByte1 = 0xC0;        /* High order bit is 1 so */
|                                         /* key is present. The */
|                                         /* 40 bit means that CV is */
|                                         /* present */
|
|                                         /* Set the first byte of */
|                                         /* Master key verification */
|                                         /* pattern. */
|     token_ptr->MasterKeyVerifPattern[0] = mkvp[0];
|                                         /* Set the second byte of */
|                                         /* Master key verification */
|                                         /* pattern. */
|     token_ptr->MasterKeyVerifPattern[1] = mkvp[1];
|
|                                         /* Copy control vector into*/

```

```

|                                     /* token          */
| memcpy(token_ptr->controlVectorBase, cv, 16);
| memcpy(token_ptr->leftHalfKey, keyvalue, 16); /*Copy key to token */
|
| /*****
| /* Calculate the TVV by adding every 4 bytes */
| /*****
| tvv_pos = externalkey;
| tvv = 0;
| while (tvv_pos < (externalkey + 60))
| {
|     memcpy((void*)&tvv_part,tvv_pos,4);
|     tvv += tvv_part;
|     tvv_pos += 4;
| }
| memcpy(token_ptr->tvv, (void*)&tvv, 4);
|
| /*****
| /* Create a Key Record in Key Store          */
| /*****
| exit_data_length = 0;
| CSNBKRC((long *) &return_code,
|         (long *) &reason_code,
|         (long *) &exit_data_length,
|         (char *) exit_data,
|         (char *) label);
|
| if (return_code > 4)
| {
|     printf("Key_Record Create failed with return/reason codes \
|           %d/%d \n",return_code, reason_code);
|     return 1;
| }
|
| /*****
| /* Write the record out to Key Store          */
| /*****
| CSNBKRW((long *) &return_code,
|         (long *) &reason_code,
|         (long *) &exit_data_length,
|         (char *) exit_data,
|         (char *) keytoken,
|         (char *) label);
|
| if (return_code > 4)
| {
|     printf("Key_Record Write failed with return/reason codes \
|           %d/%d \n",return_code, reason_code);
|     return 1;
| }
|
| return 0;
| }

```



Using IMPORTER key-encrypting keys: To import all types of cross-domain keys you will need the following IMPORTER key-encrypting keys:

1. A KEK for importing the left half of exporter keys.
Create this key using the clear host master key, the left half of an exporter key-encrypting key control vector, and 16 bytes of hex 88.
2. A KEK for importing the right half of exporter keys

- | Create this key using the clear host master key, the right half of an exporter key-encrypting key control vector, and 16 bytes of hex 88.
- | 3. A KEK for importing the left half of importer keys.
| Create this key using the clear host master key, the left half of an importer key-encrypting key control vector, and 16 bytes of hex 22.
- | 4. A KEK for importing the right half of importer keys.
| Create this key using the clear host master key, the right half of an importer key-encrypting key control vector, and 16 bytes of hex 22.
- | 5. A KEK for importing the left half of OPINENC keys.
| Create this key using the clear host master key, the left half of an OPINENC key control vector, and 16 bytes of hex 88.
- | 6. A KEK for importing the right half of OPINENC keys.
| Create this key using the clear host master key, the right half of an OPINENC key control vector, and 16 bytes of hex 88.
- | 7. A KEK for importing the left half of IPINENC keys.
| Create this key using the clear host master key, the left half of an IPINENC key control vector, and 16 bytes of hex 44.
- | 8. A KEK for importing the right half of IPINENC keys.
| Create this key using the clear host master key, the right half of an IPINENC key control vector, and 16 bytes of hex 44.
- | 9. A KEK for importing the left half of PINGEN keys.
| Create this key using the clear host master key, the left half of a PINGEN key control vector, and 16 bytes of hex 44.
- | 10. A KEK for importing the right half of PINGEN keys.
| Create this key using the clear host master key, the right half of a PINGEN key control vector, and 16 bytes of hex 44.
- | 11. A KEK for importing the left half of PINVER keys.
| Create this key using the clear host master key, the left half of a PINVER key control vector, and 16 bytes of hex 44.
- | 12. A KEK for importing the right half of PINVER keys.
| Create this key using the clear host master key, the right half of a PINVER key control vector, and 16 bytes of hex 44.

| **Migrate key store files from the IBM CCA Services for OS/400 PRPQ**

| If you currently use the Common Cryptographic Architecture (CCA) Services for iSeries (5799-FRF), you can migrate the keys in the key store file so that your Cryptographic Coprocessor can use them. The Coprocessor uses the migrated keys with the CCA Cryptographic Service Provider (CCA CSP, which is packaged as OS/400 Option 35).

| **Note:** You cannot migrate all keys because the CCA Services supports a wider range of key types than the Cryptographic Coprocessor. For example, you cannot migrate keys that have had the prohibit-export bit in the control vector set. Also, you cannot migrate any of the PKA keys in the CCA Services for iSeries because CCA Services provides public key algorithm (PKA) support that is significantly different than that in the Cryptographic Coprocessor.

| You will need to write two programs to migrate your Data Encryption Standard (DES) keys. Or, there are two program examples, “Example: EXPORTing keys” on page 111 and “Example: IMPORTing keys” on page 116, which you can change and run to migrate the key store files. The CCA defines the format of the external DES key tokens and therefore is the same for both products.

| Use the EXPORT program in conjunction with the IMPORT program. This will migrate DES keys from
| the IBM CCA Services for iSeries to your Cryptographic Coprocessor and CCA CSP. You should run the
| EXPORT program first to generate a file that contains the necessary key information in a secure,
| exportable form. You should then transfer the file to the target server. You can then run the IMPORT
| program to import the keys from the file into a key storage file that you have created. The key storage
| file to which you want to import the keys must already exist before you run the program.

| To change the program examples, follow these steps.

- | 1. Import the same clear key value for a key-encrypting key into both products. For the CCA Services,
| the key-encrypting key must be an EXPORTER, and for CCA CSP it must be an IMPORTER.
- | 2. Run the Key_Export (CSNBKEX) CCA API in the CCA Services for **each key** you want to migrate.
| This causes the program example to call an API.
- | 3. Import the outputted external key token into CCA CSP and your Cryptographic Coprocessor by using
| the Key_Import (CSNBKIM) CCA API. Remember to change the program to do this for **each key**.

| Once you change the program to address each key, you can run the program. Remember to run EXPORT
| first and then IMPORT.

| **Note:** If you choose to use the program examples provided, change them to suit your specific needs. For
| security reasons, IBM recommends that you individualize these program examples rather than
| using the default values provided.

| **Example: EXPORTing keys:** This is step one. Change this program example to suit your needs for
| migrating the key store files. Once you run this program, use "Example: IMPORTing keys" on page 116
| to complete the migration process.

```
| /*-----*/  
| /* Description: One of two programs used to migrate DES keys */  
| /* from a key store file used with the 2620 to a */  
| /* key store file for use with the card. */  
| /* */  
| /* Note: This program is intended to be used in conjunction with */  
| /* IMPORT_TSS to migrate DES keys from 2620 to card. */  
| /* EXPORT_TSS should be run first to generate a file */  
| /* containing the needed key information in a secure, */  
| /* exportable form. The file should then be transferred */  
| /* to the target system. IMPORT_TSS can then be run using */  
| /* the file to import the keys into a previously created */  
| /* key storage file. */  
| /* */  
| /* COPYRIGHT 5769-SS1 (c) IBM Corp 1999 */  
| /* */  
| /* This material contains programming source code for your */  
| /* consideration. These examples have not been thoroughly */  
| /* tested under all conditions. IBM, therefore, cannot */  
| /* guarantee or imply reliability, serviceability, or function */  
| /* of these programs. All programs contained herein are */  
| /* provided to you "AS IS". THE IMPLIED WARRANTIES OF */  
| /* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */  
| /* EXPRESSLY DISCLAIMED. IBM provides no program services for */  
| /* these programs and files. */  
| /* */  
| /* Parameters: File to contain exported key information */  
| /* */  
| /* Examples: */  
| /* CALL PGM(EXPORT_TSS) PARM('File_for_Exported_Keys') */  
| /* */  
| /* Use the following commands to compile this program: */  
| /* ADDLIB LIB(QTSS) */  
| /* CRTCMOD MODULE(EXPORT_TSS) SRCFILE(SAMPLE) */
```

```

| /* CRTPGM PGM(EXPORT_TSS) MODULE(EXPORT_TSS) */
| /* */
| /* Note: authority to the functions CSNBKEX, CSNBKPI, CSNBKRL, */
| /* and CSNBKTB is assumed */
| /* */
| /* Common Cryptographic Architecture (CCA) verbs used: */
| /* Key_Export CSNBKEX */
| /* Key_Part_Import CSNBKPI */
| /* Key_Record_List CSNBKRL */
| /* Key_Token_Build CSNBKTB */
| /*-----*/
|
| #include <stdlib.h>
| #include <stdio.h>
| #include <string.h>
| #include "MIPTRNAM.H" /* needed to resolve function ptrs */
| #include "csucinl.h" /* header file for CCA Cryptographic
| Service Provider for iSeries */
|
| int main(int argc, char *argv[])
| {
|
| /*-----*/
| /* standard return codes */
| /*-----*/
| #define ERROR -1
| #define OK 0
|
| /*-----*/
| /* Declare function pointers (see csucinl.h) */
| /*-----*/
| T_CSNBKEX *CSNBKEX;
| T_CSNBKRL *CSNBKRL;
| T_CSNBKPI *CSNBKPI;
| T_CSNBKTB *CSNBKTB;
|
| /*-----*/
| /* standard CCA parameters */
| /*-----*/
|
| long return_code;
| long reason_code;
| long exit_data_length = 0;
| char exit_data[2];
| long rule_array_count = 0;
| char rule_array[2][8];
|
| /*-----*/
| /* additional parameters needed for CSNBKRL */
| /*-----*/
| char key_label[64];
| long data_set_name_length = 0;
| char data_set_name[65];
| char security_server_name[9] = " ";
|
| /*-----*/
| /* additional parameters needed for CSNBKEX */
| /*-----*/
| char key_type[8];
| char source_key_identifier[64];
| char exporter_key_identifier[64];
| char target_key_token[64];
|
| /*-----*/
| /* additional parameters needed for CSNBKTB */
| /*-----*/

```

```

| /*-----*/
| char key_token[64];
| char key_value[64];
| long master_key_verification_pattern = 0;
| long reserved_int;
| char reserved_str[8];
| char control_vector[16];
|
| /*-----*/
| /* additional parameters needed for CSNBKPI */
| /*-----*/
| char key_part[16];
| char key_identifier[64];
|
| /*-----*/
| /* Other variables */
| /*-----*/
| char header1[77];
| long num_rec, i;
| long num_successful = 0;
| long rec_length, offset_rec1;
|
| char key_record[154];
|
| FILE *kr1, *export_file;
|
| /* Check input parm */
| if (argc < 2)
| {
|     printf("File for storing the exported key data not specified.\n");
|     return ERROR;
| }
|
| /*-----*/
| /* Resolve function pointers */
| /*-----*/
| _lib_qualify(CSNBKEX,QTSS)
| _lib_qualify(CSNBKRL,QTSS)
| _lib_qualify(CSNBKPI,QTSS)
| _lib_qualify(CSNBKTB,QTSS)
|
| memset(key_label, ' ',64);
| memcpy(key_label, "*****",9);
|
| /*-----*/
| /* Invoke the verb to create a DES Key Record List */
| /*-----*/
| CSNBKRL( &return_code,
|         &reason_code,
|         &exit_data_length,
|         exit_data,
|         key_label,
|         &data_set_name_length,
|         data_set_name,
|         security_server_name);
|
| if ((return_code != 0) || (reason_code != 0))
| {
|     printf("Key Record List generation was unsuccessful. ");
|     printf("Return/reason code = %d/%d\n",return_code, reason_code);
|     return ERROR;
| }

```

```

printf("Key Record List generation was successful. ");
printf("Return/reason codes = %d/%d\n",return_code, reason_code);
data_set_name[data_set_name_length] = '\0';
printf("data_set_name = %s\n\n",data_set_name);

/* Generate a clear key for export use. */
/* The same key will be used for import. */
memcpy(key_type,"EXPORTER",8);
rule_array_count = 2;
memcpy(rule_array[0],"INTERNAL",8);
memcpy(rule_array[1],"KEY-PART",8);

CSNBKTB( &return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        key_token,
        key_type,
        &rule_array_count,
        (char *) rule_array,
        key_value,
        &master_key_verification_pattern,
        &reserved_int,
        reserved_str,
        control_vector,
        reserved_str,
        &reserved_int,
        reserved_str,
        reserved_str);

if (return_code != 0) {
    printf("Building of the export key failed.\n");
    printf("Key Token Build failed.");
    printf("Return/reason codes = %d/%d\n",return_code, reason_code);
    return ERROR;
}

/* Import the key parts to be used. */
rule_array_count = 1;
memcpy(rule_array[0],"FIRST  ",8);
memset(key_part,'\x01',16);

for(i=1;i<=2;i++) {
    CSNBKPI( &return_code,
            &reason_code,
            &exit_data_length,
            (char *) exit_data,
            &rule_array_count,
            (char *) rule_array,
            key_part,
            key_token);

    if (return_code != 0) {
        printf("Building of the export key failed.\n");
        printf("Key Part Import failed.");
        printf("Return/reason codes = %d/%d\n",return_code, reason_code);
        return ERROR;
    }

    memcpy(rule_array[0],"LAST  ",8);
    /* Set key part to the clear key to be used. */
    /* Note: It may not be desirable to hard-code this. */
    memcpy(key_part,"Clear.KEY.hErE!!",16);
}
}

```



```

|
| /* Export key built successfully. */
| /* Open the Key Record List file. */
| krl = fopen(data_set_name, "rb");
|
| if (krl == NULL)
| { /* Open failed. */
|   printf("The open of the Key Record List file failed.\n");
|   return ERROR;
| }
|
|
| /* Key record list open was successful. */
| /* Open the file to save key info. */
| export_file = fopen(argv[1], "wb");
| if (export_file == NULL)
| {
|   printf("Opening of key export file failed.\n");
|   fclose(krl);
|   return ERROR;
| }
|
| /* Write num_successful to the export file to hold a place for it. */
| fwrite(&num_successful,sizeof(long),1,export_file);
|
| /* Read the first part of the KRL header. */
| fread(header1,1,77,krl);
|
| /* Get the number of key records in the file. */
| num_rec = atoi(&header1[50]);
| printf("Number of key records = %d\n",num_rec);
|
| /* Get the length for the key records. */
| rec_length = atol(&header1[58]);
|
| /* Get the offset for the first key record. */
| offset_rec1 = atol(&header1[62]);
|
| /* Set the file pointer to the first key record. */
| fseek(krl, offset_rec1, SEEK_SET);
|
| /* Set the key type to TOKEN. */
| memcpy(key_type,"TOKEN  ",8);
|
| /* Loop through the entries in the KRL and EXPORT. */
| for (i = 1; i <= num_rec; i++)
| {
|   fread(key_record, 1, rec_length, krl);
|   memcpy(source_key_identifier, &key_record[3], 64);
|
|   CSNBKEX(&return_code,
|           &reason_code,
|           &exit_data_length,
|           exit_data,
|           key_type,
|           source_key_identifier,
|           key_token,
|           /* exporter_key_identifier, */
|           target_key_token);
|
|   printf("Exporting of key = %.64s",source_key_identifier);
|   printf("completed with return/reason codes of ");
|   printf("%d/%d\n",return_code,reason_code);
|
|   if (return_code == 0)
|   {

```

```

|         ++num_successful;
|         fwrite(source_key_identifer, 1, 64, export_file);
|         fwrite(target_key_token, 1, 64, export_file);
|     }
|
| } /* end of for loop */
|
| printf("Key store file exported successfully.\n");
| printf("%d key(s) successfully exported.\n\n",num_successful);
|
| /* Write out the number of exported keys and close the file. */
| fseek(export_file,0,SEEK_SET);
| fwrite(&num_successful,sizeof(long),1,export_file);
|
| /* Close the files and return. */
| fclose(kr1);
| fclose(export_file);
| return OK;
| }

```

Example: IMPORTing keys: This is step two. If you have not already done so, run the “Example: EXPORTing keys” on page 111 program to begin the migration process. Then change this program example to suit your needs for completing the migration of the key store files.

```

| /*-----*/
| /* Description: One of two programs used to migrate DES keys */
| /* from a key store file used with the 2620 to a */
| /* key store file for use with the card. */
| /* */
| /* Note: This program is intended to be used in conjunction with */
| /* EXPORT_TSS to migrate DES keys from 2620. */
| /* EXPORT_TSS should be run first to generate a file */
| /* containing the needed key information in a secure, */
| /* exportable form. The file should then be transferred */
| /* to the target system. IMPORT_TSS can then be run using */
| /* the file to import the keys into a previously created */
| /* key storage file. */
| /* */
| /* */
| /* COPYRIGHT 5769-SS1 (c) IBM Corp 1999 */
| /* */
| /* This material contains programming source code for your */
| /* consideration. These examples have not been thoroughly */
| /* tested under all conditions. IBM, therefore, cannot */
| /* guarantee or imply reliability, serviceability, or function */
| /* of these programs. All programs contained herein are */
| /* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
| /* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
| /* EXPRESSLY DISCLAIMED. IBM provides no program services for */
| /* these programs and files. */
| /* */
| /* Parameters: File containing exported key information */
| /* */
| /* Examples: */
| /* CALL PGM(IMPORT_TSS) PARM('Exported_Key_File') */
| /* */
| /* Note: The CCA verbs used in the this program are more fully */
| /* described in the IBM CCA Basic Services Reference */
| /* and Guide (SC31-8609) publication. */
| /* */
| /* Note: This program assumes the card you want to use is */
| /* already identified either by defaulting to the CRP01 */
| /* device or has been explicitly named using the */
| /* Cryptographic_Resource_Allocate verb. Also this */
| /* device must be varied on and you must be authorized */
| /* to use this device description. */
| /* */

```

```

| /*      This program also assumes the key store file you will  */
| /*      use is already identified either by being specified on */
| /*      the cryptographic device or has been explicitly named  */
| /*      using the Key_Store_Designate verb. Also you must be  */
| /*      authorized to update records in this file.            */
| /*                                                            */
| /* Use the following commands to compile this program:        */
| /*  ADDLIB LIB(QCCA)                                          */
| /*  CRTMOD MODULE(IMPORT_TSS) SRCFILE(SAMPLE)                */
| /*  CRTPGM  PGM(IMPORT_TSS) MODULE(IMPORT_TSS)              */
| /*          BNDSRVPGM(QCCA/CSNBKRC QCCA/CSNBKIM QCCA/CSNBKPI) */
| /*                                                            */
| /* Note: authority to the CSNBKIM, CSNBKPI, and CSNBKRC      */
| /*       service programs in the QCCA library is assumed.    */
| /*                                                            */
| /* Common Cryptographic Architecture (CCA) verbs used:      */
| /*  Key_Import          CSNBKIM                               */
| /*  Key_Record_Create   CSNBKRC                               */
| /*  Key_Part_Import     CSNBKPI                               */
| /*-----*/
|
| #include <stdlib.h>
| #include <stdio.h>
| #include <string.h>
| #include "csucincl.h"      /* header file for CCA Cryptographic
|                             Service Provider for iSeries  */
|
| /*-----*/
| /* Structure defining the DES key token for internal keys. This */
| /* structure is used in the creation of the importer key-      */
| /* encrypting key. For more information on the fields in this  */
| /* structure, see the IBM CCA Basic Services Reference and */
| /* Guide (SC31-8609-01), Appendix B and C.                    */
| /*-----*/
| struct DES_key_token {
|     char    type;      /* Set to 0x01 for 'internal' */
|     char    resv1;     /* Reserved (set to binary zero) */
|     char    mkvp[2];   /* Master Key Verification Pattern */
|     char    version;   /* Version. Will be set to 0x03. */
|     char    resv2;     /* Reserved (set to binary zero) */
|     char    flag;      /* Flag */
|     char    resv3;     /* Reserved (set to binary zero) */
|     char    resv4[8];  /* Reserved (set to binary zero) */
|     char    key1[8];   /* Single length encrypted key or
|                          left half of double length
|                          encrypted key. */
|     char    key2[8];   /* Null or right half of double
|                          length encrypted key */
|     int     cvb1[2];   /* Control-vector base */
|     int     cvb2[2];   /* Null or control vector base for
|                          the 2nd eight-byte portion of a
|                          16-byte key */
|     char    resv5[12]; /* Reserved (set to binary zero) */
|     int     tvv;      /* Token-validation value */
| };
|
| int main(int argc, char *argv[])
| {
|
|     /*-----*/
|     /* standard return codes */
|     /*-----*/
|
| #define ERROR -1
| #define OK    0

```

```

/*-----*/
/* standard CCA parameters */
/*-----*/

long return_code;
long reason_code;
long exit_data_length = 0;
char exit_data[2];
long rule_array_count = 0;
char rule_array[2][8];

/*-----*/
/* additional parameters required for CSNBKRC and CSNBKIM */
/*-----*/
char import_key_label[64];
char import_key_token[64];

/*-----*/
/* additional parameters required for CSNBKPI */
/*-----*/
struct DES_key_token importer_kt;

char importer_key_token[64];
char key_type[8];
char key_part[16];

/*-----*/
/* Other variables */
/*-----*/
long num_rec = 0, i;
long num_imported = 0;

FILE *import_file;

printf("\n\n");
/* Check input parm */
if (argc < 2)
{
    printf("File containing the exported key data not specified.\n");
    return ERROR;
}

/* Generate a clear key for import use. */

/* Initialize the importer key token. */
memset(&importer_kt,0x00,sizeof(struct DES_key_token));
importer_kt.type = 0x01;
importer_kt.version = 0x03;
importer_kt.flag = 0x40; /* Indicates control vector is present */
importer_kt.cvb1[0] = 0x00427d00;
importer_kt.cvb1[1] = 0x03480000;
importer_kt.cvb2[0] = 0x00427d00;
importer_kt.cvb2[1] = 0x03280000;
importer_kt.tvv = 0xaf53a00;

/* Initialize parameters for the first pass */
rule_array_count = 1;
memcpy(rule_array[0],"FIRST ",8);
memset(key_part,0x01,16);

for(i=1;i<=2;i++) {
    CSNBKPI( &return_code,
            &reason_code,
            &exit_data_length,
            (char *) exit_data,

```

```

        &rule_array_count,
        (char *) rule_array,
        key_part,
        (char *) &importer_kt);

    if (return_code != 0) {
        printf("Building of the importer key failed.\n");
        printf("Key Part Import failed.");
        printf("Return/reason codes = %d/%d\n",return_code, reason_code);
        return ERROR;
    }
    else if ( i == 1) {
        /* Init variables for the final pass */
        memcpy(rule_array[0],"LAST  ",8);
        /* Set key part to the clear key to be used. */
        memcpy(key_part,"CLEAR.KEY.hErE!!",16);
    }
}

/* Import key built successfully. */
printf("Importer key built successfully.\n\n");

/* Open the Exported Key file. */
import_file = fopen(argv[1], "rb");

if (import_file == NULL)
{ /* Open failed. */
    printf("The open of the Exported Key file failed\n");
    return ERROR;
}

/* Import Key file open was successful. */
fread(&num_rec,sizeof(num_rec),1,import_file);

/* Loop through the entries in the import file and create key records. */
for (i = 1; i <= num_rec; i++)
{

    fread(import_key_label, 1, 64, import_file);
    fread(import_key_token, 1, 64, import_file);

    printf("Importing DES key:\n");
    printf("    \".64s\"\n",import_key_label);

    /* Create a key record. */
    CSNBKRC(&return_code,
            &reason_code,
            &exit_data_length,
            exit_data,
            import_key_label);

    if (return_code != 0)
    {
        printf("    Key record creation failed. ");
        printf("Return/reason codes = %d/%d\n\n",return_code,reason_code);
        continue;
    }

    /* Else, key record created successfully so import the key. */
    memcpy(key_type,"TOKEN  ",8);

    CSNBKIM( &return_code,
            &reason_code,
            &exit_data_length,
            exit_data,
            key_type,
            import_key_token,

```

```

|         (char *) &importer_kt,
|         import_key_label);
|
|     if (return_code != 0)
|     {
|         printf("    Key import failed. ");
|         printf("Return/reason codes = %d/%d\n\n",return_code,reason_code);
|         continue;
|     }
|
|     /* else, Key import was a success. */
|     printf("    Key imported successfully. ");
|     printf("Return/reason codes = %d/%d\n\n",return_code,reason_code);
|     ++num_imported;
| } /* end of for loop */
|
| printf("\nCompleted key import procedure.\n");
| printf("%d of %d key(s) successfully imported.\n\n",num_imported,num_rec);
| fclose(import_file);
| return OK;
|
| }

```

Manage the Cryptographic Coprocessor

This section is mainly for i5/OS or OS/400 application use of the Cryptographic Coprocessor. If you are using multiple Coprocessors with SSL, see “Manage multiple Cryptographic Coprocessors” on page 175 and “Clone master keys” on page 185.

After you set up your Cryptographic Coprocessor, you can begin writing programs to make use of your Cryptographic Coprocessor’s cryptographic functions. You can use programs to perform these tasks:

- “Log on or off of the Cryptographic Coprocessor” to work with role-restricted APIs.
- “Query status or request information” on page 130.
- “Initialize a key store file” on page 136 if you plan to keep records of your DES and PKA keys.
- “Create DES and PKA keys” on page 141 and storing them in a DES key store.
- “Encrypt or decrypt a file” on page 148.
- “Work with PINs” on page 154.
- “Generate and verify a digital signature” on page 166.
- “Manage multiple Cryptographic Coprocessors” on page 175.
- “Clone master keys” on page 185 when using multiple Cryptographic Coprocessors.

Note: Many of the pages in this section include one or more program examples. Change these programs to suit your specific needs. Some require that you change only one or two parameters while others require more extensive changes. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

Log on or off of the Cryptographic Coprocessor

Logging on

You need to log on only if you wish to use an API that uses an access control point that is not enabled in the default role. Log on with a profile that uses a role that has the access control point you want to use enabled.

After you log on to your Cryptographic Coprocessor, you can run programs to utilize the cryptographic functions for your Cryptographic Coprocessor. You can log on by writing an application that uses the Logon_Control (CSUALCT) API verb. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both perform the same function.

- “Example: ILE C program for logging on to your Cryptographic Coprocessor” on page 121

- “Example: ILE RPG program for logging on to your Cryptographic Coprocessor” on page 123

Logging off

When you have finished with your Cryptographic Coprocessor, you should log off of your Cryptographic Coprocessor. You can log off by writing an application that uses the Logon_Control (CSUALCT) API verb. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both perform the same function.

- “Example: ILE C program for logging off of your Cryptographic Coprocessor” on page 126
- “Example: ILE RPG program for logging off of your Cryptographic Coprocessor” on page 128

Note: If you choose to use the program examples provided, change them to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

Example: ILE C program for logging on to your Cryptographic Coprocessor: Change this program example to suit your needs for logging on to your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Log on to the card using your profile and passphrase. */
/* */
/* */
/* COPYRIGHT 5769-SS1, 5722-SS1 (C) IBM CORP. 1999, 2000 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these programs. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* */
/* Note: This verb is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: */
/* none. */
/* */
/* Example: */
/* CALL PGM(LOGON) */
/* */
/* Note: This program assumes the card with the profile is */
/* already identified either by defaulting to the CRP01 */
/* device or by being explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/* */
/* */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(LOGON) SRCFILE(SAMPLE) */
/* CRTPGM PGM(LOGON) MODULE(LOGON) BNDSRVPGM(QCCA/CSUALCT) */
/* */
/* Note: Authority to the CSUALCT service program in the */
/* QCCA library is assumed. */
/* */

```

```

/* The Common Cryptographic Architecture (CCA) verb used is      */
/* Logon_Control (CSUALCT).                                       */
/*                                                                 */
/*-----*/

#include "csucincl.h"      /* header file for CCA Cryptographic */
                          /* Service Provider for iSeries     */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*-----*/
/* standard return codes                                         */
/*-----*/

#define ERROR    -1
#define OK       0
#define WARNING  4

int main(int argc, char *argv[])

{
    /*-----*/
    /* standard CCA parameters                                   */
    /*-----*/

    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 2;
    char exit_data[4];
    char rule_array[2][8];
    long rule_array_count = 2;

    /*-----*/
    /* fields unique to this sample program                     */
    /*-----*/

    char profile[8];
    long auth_parm_length;
    char auth_parm[4];
    long auth_data_length;
    char auth_data[256];

    /* set rule array keywords                                   */
    memcpy(rule_array, "LOGON  PPHRASE ", 16);

    /* Check for correct number of parameters                   */
    if (argc < 3)
    {
        printf("Usage: CALL LOGON ( profile 'pass phrase')\n");
        return(ERROR);
    }

    /* Set profile and pad out with blanks                       */
    memset(profile, ' ', 8);
    if (strlen(argv[1]) > 8)
    {
        printf("Profile is limited to 8 characters.\n");
        return(ERROR);
    }
    memcpy(profile, argv[1], strlen(argv[1]));

    /* Authentication parm length must be 0 for logon           */
    auth_parm_length = 0;

```



```

/* Authentication data length is length of the pass-phrase          */
auth_data_length = strlen(argv[2]);

/* invoke verb to log on to the card                                */

CSUALCT( &return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (char *)rule_array,
        profile,
        &auth_parm_length,
        auth_parm,
        &auth_data_length,
        argv[2]);

if (return_code != OK)
{
    printf("Log on failed with return/reason codes %ld/%ld\n",
          return_code, reason_code);
}
else
    printf("Logon was successful\n");
}

```

Example: ILE RPG program for logging on to your Cryptographic Coprocessor: Change this program example to suit your needs for logging on to your Cryptographic Coprocessor.

Note: Read the "Code disclaimer information" on page 276 for important legal information.

```

D*****
D* LOGON
D*
D* Log on to the Cryptographic Coprocessor.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: Profile
D*             Pass-phrase
D*
D* Example:
D* CALL PGM(LOGON) PARM(PROFILE PASSPRHASE)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(LOGON) SRCFILE(SAMPLE)
D* CRTPGM PGM(LOGON) MODULE(LOGON)

```

```

D*          BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSUALCT service program in the
D*       QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Facilty_Control (CSUACFC)
D*
D* This program assumes the card with the profile is
D* already identified either by defaulting to the CRP01
D* device or by being explicitly named using the
D* Cryptographic_Resource_Allocate verb. Also this
D* device must be varied on and you must be authorized
D* to use this device description.
D*****
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE  S          9B 0
D*          ** Reason code
DREASONCODE  S          9B 0
D*          ** Exit data length
DEXITDATALEN S          9B 0
D*          ** Exit data
DEXITDATA    S          4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY   S          16
D*          ** Userid parm
DUSERID      S          8
D*          ** Authentication parameter length
DAUTHPARMLEN S        9B 0 INZ(0)
D*          ** Authentication parameter
DAUTHPARAM   S          10
D*          ** Authentication data length
DAUTHDATALEN S        9B 0 INZ(0)
D*          ** Authentication data
DAUTHDATA    S          50
D*
D*****
D* Prototype for Logon Control (CSUALCT)
D*****
DCSUALCT     PR
DRETCODE     9B 0
DRSNCODE     9B 0
DEXTDTALEN   9B 0
DEXTDTA      4
DRARRAYCT    9B 0
DRARRAY      16
DUSR         8
DATHPRMLEN   9B 0
DATHPRM      10
DATHDTALEN   9B 0
DATHDTA      50
D*
D*****
D* Declares for sending messages to job log
D*****
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG         S          75  DIM(2) CTDATA PERRCD(1)
MSGLENGTH    S          9B 0 INZ(75)
D            DS

```

```

DMSGTEXT          1    75
DFAILRETC         41   44
DFAILRSNC         46   49
DMESSAGEID        S      7  INZ(' ')
DMESSAGEFILE      S     21  INZ(' ')
DMSGKEY           S      4  INZ(' ')
DMSGTYPE          S     10  INZ('*INFO ')
DSTACKENTRY       S     10  INZ('* ')
DSTACKCOUNTER     S     9B 0 INZ(2)
DERRCODE          DS
DBYTESIN          1     4B 0 INZ(0)
DBYTESOUT         5     8B 0 INZ(0)
D*
C*****
C* START OF PROGRAM *
C* *
C*-----*
C  *ENTRY      PLIST
C              PARM          USERID
C              PARM          AUTHDATA
C*-----*
C* Set the keywords in the rule array *
C*-----*
C              MOVE      'LOGON '  RULEARRAY
C              MOVE      'PPHRASE '  RULEARRAY
C              Z-ADD      2          RULEARRAYCNT
C*-----*
C* Get the length of the passphrase *
C*-----*
C              EVAL      AUTHDATALEN = %LEN(%TRIM(AUTHDATA))
C*
C*****
C* Call Logon Control SAPI *
C*****
C              CALLP      CSUALCT      (RETURNCODE:
C                                  REASONCODE:
C                                  EXITDATALEN:
C                                  EXITDATA:
C                                  RULEARRAYCNT:
C                                  RULEARRAY:
C                                  USERID:
C                                  AUTHPARMLEN:
C                                  AUTHPARAM:
C                                  AUTHDATALEN:
C                                  AUTHDATA)
C*-----*
C* Check the return code *
C*-----*
C      RETURNCODE  IFGT      0
C*
C*      *-----*
C*      * Send error message *
C*      *-----*
C              MOVE      MSG(1)      MSGTEXT
C              MOVE      RETURNCODE  FAILRETC
C              MOVE      REASONCODE  FAILRSNC
C              EXSR      SNDMSG
C*
C              ELSE
C*      *-----*
C*      * Send success message *
C*      *-----*
C              MOVE      MSG(2)      MSGTEXT
C              EXSR      SNDMSG
C*
C              ENDIF
C*
C              SETON

```

LR

```

C*
C*****
C* Subroutine to send a message
C*****
C   SNDMSG      BEGSR
C               CALL      'QMHSNDPM'
C               PARM
C               PARM      MESSAGEID
C               PARM      MESSAGEFILE
C               PARM      MSGTEXT
C               PARM      MSGLENGTH
C               PARM      MSGTYPE
C               PARM      STACKENTRY
C               PARM      STACKCOUNTER
C               PARM      MSGKEY
C               PARM      ERRCODE
C
C               ENDSR
C*

```

```

**
CSUALCT failed with return/reason codes 9999/9999'
The request completed successfully

```

Example: ILE C program for logging off of your Cryptographic Coprocessor: Change this program example to suit your needs for logging off of your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Log off the Cryptographic CoProcessor          */
/*                                               */
/*                                               */
/* COPYRIGHT 5769-SS1, 5722-SS1 (C) IBM CORP. 1999, 2000 */
/*                                               */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these programs. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/*                                               */
/* Note: This verb is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide          */
/* (SC31-8609) publication. */
/* Parameters: */
/* none. */
/* Example: */
/* CALL PGM(LOGOFF) */
/* Note: This program assumes the card with the profile is */
/* already identified either by defaulting to the CRP01 */
/* device or by being explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(LOGOFF) SRCFILE(SAMPLE) */
/* CRTPGM PGM(LOGOFF) MODULE(LOGOFF) BNDSRVPGM(QCCA/CSUALCT) */

```

```

/* Note: Authority to the CSUALCT service program in the          */
/*       QCCA library is assumed.                                */
/*                                                                 */
/* The Common Cryptographic Architecture (CCA) verb used is      */
/* Logon_Control (CSUALCT).                                       */
/*                                                                 */
/*-----*/

#include "csucincl.h"      /* header file for CCA Cryptographic */
                          /* Service Provider for iSeries     */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*-----*/
/* standard return codes                                         */
/*-----*/

#define ERROR    -1
#define OK       0

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters                                    */
    /*-----*/
    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 2;
    char exit_data[4];
    char rule_array[2][8];
    long rule_array_count = 1;

    /*-----*/
    /* fields unique to this sample program                       */
    /*-----*/
    char profile[8];
    long auth_parm_length;
    char * auth_parm = " ";
    long auth_data_length = 256;
    char auth_data[300];

    /* set rule array keywords to log off                         */
    memcpy(rule_array,"LOGOFF ",8);

    rule_array_count = 1;

    /* Both Authentication parm and data lengths must be 0      */
    auth_parm_length = 0;
    auth_data_length = 0;

    /* Invoke verb to log off the Cryptographic CoProcessor     */
    CSUALCT( &return_code,
             &reason_code,
             &exit_data_length,
             exit_data,
             &rule_array_count,
             (char *)rule_array,
             profile,
             &auth_parm_length,
             auth_parm,
             &auth_data_length,
             auth_data);
}

```

```

if (return_code != OK)
{
    printf("Log off failed with return/reason codes %ld/%ld\n\n",
        return_code, reason_code);
    return(ERROR);
}
else
{
    printf("Log off successful\n");
    return(OK);
}
}

```

Example: ILE RPG program for logging off of your Cryptographic Coprocessor: Change this program example to suit your needs for logging off of your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* LOGOFF
D*
D* Log off from the Cryptographic Coprocessor.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: None
D*
D* Example:
D* CALL PGM(LOGOFF)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(LOGOFF) SRCFILE(SAMPLE)
D* CRTPGM PGM(LOGOFF) MODULE(LOGOFF)
D* BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSUALCT service program in the
D* QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Facilty_Control (CSUACFC)
D*
D* This program assumes the card with the profile is
D* already identified either by defaulting to the CRP01
D* device or by being explicitly named using the
D* Cryptographic_Resource_Allocate verb. Also this
D* device must be varied on and you must be authorized
D* to use this device description.
D*****
D*-----
D* Declare variables for CCA SAPI calls

```

```

D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S         9B 0
D*          ** Exit data
DEXITDATA   S           4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY  S          16
D*          ** Userid parm
DUSERID     S           8
D*          ** Authentication parameter length
DAUTHPARMLEN S        9B 0 INZ(0)
D*          ** Authentication parameter
DAUTHPARAM  S           8
D*          ** Authentication data length
DAUTHDATALEN S        9B 0 INZ(0)
D*          ** Authentication data
DAUTHDATA   S           8
D*
D*****
D* Prototype for Logon Control (CSUALCT)
D*****
DCSUALCT    PR
DRETCODE    9B 0
DRSNCODE    9B 0
DEXTDTALEN  9B 0
DEXTDTA     4
DRARRAYCT   9B 0
DRARRAY     16
DUSR        8
DATHPRMLEN  9B 0
DATHPRM     8
DATHDTALEN  9B 0
DATHDTA     8
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG        S          75  DIM(2) CTDATA PERRCD(1)
DMSGLENGTH  S          9B 0 INZ(75)
D           DS
DMSGTEXT    1          75
DFAILRETC   41         44
DFAILRSNC   46         49
DMESSAGEID  S          7  INZ(' ')
DMESSAGEFILE S         21 INZ(' ')
DMSGKEY     S          4  INZ(' ')
DMSGTYPE    S         10 INZ('*INFO ')
DSTACKENTRY S         10 INZ('* ')
DSTACKCOUNTER S        9B 0 INZ(2)
DERRCODE    DS
DBYTESIN    1          4B 0 INZ(0)
DBYTESOUT   5          8B 0 INZ(0)
D*
C*****
C* START OF PROGRAM *
C* *
C*-----*
C* Set the keywords in the rule array *
C*-----*
C          MOVEL 'LOGOFF ' RULEARRAY
C          Z-ADD 1 RULEARRAYCNT

```

```

C*
C*****
C* Call Logon Control SAPI
C*****
C          CALLP      CSUALCT      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     USERID:
C                                     AUTHPARMLEN:
C                                     AUTHPARAM:
C                                     AUTHDATALEN:
C                                     AUTHDATA)
C*-----*
C* Check the return code *
C*-----*
C      RETURNCODE      IFGT      0
C*          *-----*
C*          * Send error message *
C*          *-----*
C              MOVE      MSG(1)      MSGTEXT
C              MOVE      RETURNCODE  FAILRETC
C              MOVE      REASONCODE  FAILRSNC
C              EXSR      SNDMSG
C*
C              ELSE
C*          *-----*
C*          * Send success message *
C*          *-----*
C              MOVE      MSG(2)      MSGTEXT
C              EXSR      SNDMSG
C*
C              ENDIF
C*
C              SETON                                  LR
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C              CALL      'QMHSNDPM'
C              PARM      MESSAGEID
C              PARM      MESSAGEFILE
C              PARM      MSGTEXT
C              PARM      MSGLENGTH
C              PARM      MSGTYPE
C              PARM      STACKENTRY
C              PARM      STACKCOUNTER
C              PARM      MSGKEY
C              PARM      ERRCODE
C              ENDSR
C*


```

**
CSUALCT failed with return/reason codes 9999/9999'
The request completed successfully

Query status or request information

You can query your Cryptographic Coprocessor to determine characteristics such as which algorithms are enabled, the key lengths it supports, the status of the master key, the status of cloning, and the clock setting. The easiest and fastest way to query the Cryptographic Coprocessor is to use the Cryptographic Coprocessor configuration web-based utility. Click on **Display configuration** and then select a device, then select items you want to display.

If you would prefer to write your own application to query the Coprocessor, you can do so by using the Cryptographic_Facility_Query (CSUACFQ) API verb. Two example programs are provided for your consideration. "Example: Querying the status of your Cryptographic Coprocessor" uses the STATEID and TIMEDATE keywords, while "Example: Requesting information from your Cryptographic Coprocessor" on page 133 prompts the user for the second required keyword.

The IBM PCI Cryptographic Coprocessor CCA Basic Services Reference and Guide  describes the Cryptographic_Facility_Query (CSUACFQ) security application programming interface, the types of information that you can request, and the format of the information that is returned.

Example: Querying the status of your Cryptographic Coprocessor: Change this program example to suit your needs for querying the status of your Cryptographic Coprocessor.

```

/*-----*/
/* Query the card for status or other information. */
/* This sample program uses the STATEID and TIMEDATE keywords. */
/* */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* */
/* Note: This verb is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: */
/* none. */
/* */
/* Example: */
/* CALL PGM(QUERY) */
/* */
/* Note: This program assumes the device to use is */
/* already identified either by defaulting to the CRP01 */
/* device or by being explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/* */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(QUERY) SRCFILE(SAMPLE) */
/* CRTPGM PGM(QUERY) MODULE(QUERY) BNDSRVPGM(QCCA/CSUACFQ) */
/* */
/* Note: Authority to the CSUACFQ service program in the */
/* QCCA library is assumed. */
/* */
/* The Common Cryptographic Architecture (CCA) verb used is */
/* Cryptographic_Facility_Query (CSUACFQ). */
/* */
/*-----*/

#include "csucincl.h" /* header file for CCA Cryptographic */
/* Service Provider for iSeries */

```

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR      -1
#define OK         0
#define WARNING    4

#define IDSIZE     16 /* number of bytes in environment ID */
#define TIMEDATESIZE 24 /* number of bytes in time and date */

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters */
    /*-----*/

    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 2;
    char exit_data[4];
    char rule_array[2][8];
    long rule_array_count = 2;
    char rule_array2[3][8];

    /*-----*/
    /* fields unique to this sample program */
    /*-----*/

    long verb_data_length = 0; /* currently not used by this verb */
    char * verb_data = " ";

    /* set keywords in the rule array */

    memcpy(rule_array,"ADAPTER1STATEID ",16);

    /* get the environment ID from the card */

    CSUACFQ( &return_code,
            &reason_code,
            &exit_data_length,
            exit_data,
            &rule_array_count,
            (char *)rule_array,
            &verb_data_length,
            verb_data);

    if ( (return_code == OK) | (return_code == WARNING) )
    {
        printf("Environment ID was successfully returned.\n");
        printf("Return/reason codes ");
        printf("%d/%d\n\n", return_code, reason_code);
        printf("ID = %.16s\n", rule_array);
    }

    else
    {

```

```

printf("An error occurred while getting the environment ID.\n");
printf("Return/reason codes ");
printf("%1d/%1d\n\n", return_code, reason_code);
/* return(ERROR) */;
}

/* set count to number of bytes of returned data */
rule_array_count = 2;

return_code = 0;
reason_code = 0;

/* set keywords in the rule array */
memcpy(rule_array2,"ADAPTER1TIMEDATE",16);

/* get the time from the card */

CSUACFQ( &return_code,
         &reason_code,
         &exit_data_length,
         exit_data,
         &rule_array_count,
         (char *)rule_array2,
         &verb_data_length,
         verb_data);

if ( (return_code == OK) | (return_code == WARNING) )
{
printf("Time and date was successfully returned.\n");
printf("Return/reason codes ");
printf("%1d/%1d\n\n", return_code, reason_code);

printf("DATE = %.8s\n", rule_array2);
printf("TIME = %.8s\n", &rule_array2[1]);
printf("DAY of WEEK = %.8s\n", &rule_array2[2]);
}

else
{
printf("An error occurred while getting the time and date.\n");
printf("Return/reason codes ");
printf("%1d/%1d\n\n", return_code, reason_code);

return(ERROR);
}
}

```

Example: Requesting information from your Cryptographic Coprocessor: Change this program example to suit your needs for requesting information from your Cryptographic Coprocessor.

```

/*-----*/
/* Query the card for status or other information. */
/* This sample program prompts the user for the second required */
/* keyword. (ADAPTER1 keyword is assumed.) */
/* */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999 */
/* */

```

```

/* This material contains programming source code for your          */
/* consideration. These examples have not been thoroughly          */
/* tested under all conditions. IBM, therefore, cannot            */
/* guarantee or imply reliability, serviceability, or function    */
/* of these program. All programs contained herein are            */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF             */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE      */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files.                                       */
/*                                                                    */
/*                                                                    */
/* Note: This verb is more fully described in Chapter 2 of        */
/* IBM CCA Basic Services Reference and Guide                    */
/* (SC31-8609) publication.                                       */
/*                                                                    */
/* Parameters:                                                    */
/* char * keyword2 upto 8 bytes                                   */
/*                                                                    */
/* Example:                                                       */
/* CALL PGM(CFQ) TIMEDATE                                         */
/*                                                                    */
/* Note: This program assumes the device to use is              */
/* already identified either by defaulting to the CRP01          */
/* device or by being explicitly named using the                 */
/* Cryptographic_Resource_Allocate verb. Also this              */
/* device must be varied on and you must be authorized          */
/* to use this device description.                                */
/*                                                                    */
/* Use these commands to compile this program on iSeries:       */
/* ADDLIB LIB(QCCA)                                              */
/* CRTCMOD MODULE(CFQ) SRCFILE(SAMPLE)                          */
/* CRTPGM PGM(CFQ) MODULE(CFQ) BNDSRVPGM(QCCA/CSUACFQ)          */
/*                                                                    */
/* Note: Authority to the CSUACFQ service program in the        */
/* QCCA library is assumed.                                       */
/*                                                                    */
/* The Common Cryptographic Architecture (CCA) verb used is     */
/* Cryptographic_Facility_Query (CSUACFQ).                      */
/*                                                                    */
/*-----*/

#include "csucincl.h"      /* header file for CCA Cryptographic */
                          /* Service Provider for iSeries      */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*-----*/
/* standard return codes                                         */
/*-----*/

#define ERROR      -1
#define OK         0
#define WARNING    4

int main(int argc, char *argv[])
{
/*-----*/
/* standard CCA parameters                                       */
/*-----*/

    long return_code = 0;
    long reason_code = 0;

```

```

long exit_data_length = 2;
char exit_data[4];
char rule_array[18][8];
long rule_array_count = 2;

/*-----*/
/* fields unique to this sample program */
/*-----*/

long verb_data_length = 0; /* currently not used by this verb */
char * verb_data = " ";

int i;

/* check the keyboard input */

if (argc != 2)
{
printf("You did not enter the keyword parameter.\n");
printf("Enter one of the following: STATCCA, STATCARD, ");
printf("STATDIAG, STATEXPT, STATMOFN, STATEID, TIMEDATE\n");
return(ERROR);
}

if ( ( strlen(argv[1]) > 8 ) | ( strlen(argv[1]) < 7 ) )
{
printf("Your input string is not the right length.\n");
printf("Input keyword must be 7 or 8 characters.\n");

printf("Enter one of the following: STATCCA, STATCARD, ");
printf("STATDIAG, STATEXPT, STATMOFN, STATEID, TIMEDATE\n");
return(ERROR);
}

/* set keywords in the rule array */

memcpy(rule_array,"ADAPTER1",16);
memcpy(&rule_array[1], argv[1], strlen(argv[1]));

/* get the requested data from the card */

CSUACFQ( &return_code,
&reason_code,
&exit_data_length,
exit_data,
&rule_array_count,
(char *)rule_array,
&verb_data_length,
verb_data);

if ( ( return_code == OK ) | ( return_code == WARNING ) )
{
printf("Requested data was successfully returned.\n");
printf("Return/reason codes ");
printf("%d/%d\n\n", return_code, reason_code);
printf("%s data = ", argv[1]);

```

```

for (i = 0; i < 8 * rule_array_count; i++)
    printf("%c", rule_array[i / 8][i % 8]);
printf("\n");
}

else
{
printf("An error occurred while getting the requested data.\n");

printf("You requested %s\n", argv[1]);

printf("Return/reason codes ");

printf("%ld/%ld\n\n", return_code, reason_code);

return(ERROR);
}
}

```

Initialize a key store file

A key store file is a database file that stores operational keys, i.e. keys encrypted under the master key. You can initialize two different types of key stores for your Cryptographic Coprocessor. The Cryptographic Coprocessor uses one type to store PKA keys and the other to store DES keys. You need to initialize a key store file if you plan to store keys in it or if you plan to use retain keys on hardware.

The CCA CSP creates a DB2[®] key store file, if one does not already exist. If a key store file already exists, the CCA CSP deletes the file and recreates a new one.

To initialize a key store, you can use the Cryptographic Coprocessor configuration utility. Click on **Manage configuration** and then click on either **DES keys** or **PKA keys** depending upon what key store file you wish to initialize. With the utility, you can only initialize a file if it does not already exist.

If you would rather write your own application to initialize a key store file, you can do so by using the KeyStore_Initialize (CSNBKSI) API verb. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both perform the same function.

- “Example: ILE C program for initializing a key store for your Cryptographic Coprocessor”
- “Example: ILE RPG program for initializing a key store for your Cryptographic Coprocessor” on page 138

Note: If you choose to use one of the program examples provided, change it to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

After you create a key store for your Cryptographic Coprocessor, you can generate DES and PKA keys using “Create DES and PKA keys” on page 141 to store in your key store files.

Example: ILE C program for initializing a key store for your Cryptographic Coprocessor: Change this program example to suit your needs for initializing a key store for your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Create key store files for PKA keys.          */
/*                                              */
/* COPYRIGHT      5769-SS1 (c) IBM Corp 1999, 2000 */
/*                                              */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */

```

```

/* guarantee or imply reliability, serviceability, or function */
/* of these programs. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* Parameters: */
/* Qualified File Name */
/* */
/* Examples: */
/* CALL PGM(INZPKEYST) PARM('QGPL/PKAFILE') */
/* */
/* Use the following commands to compile this program: */
/* ADDLIB LIB(QCCA) */
/* CRTCPGM MODULE(INZPKEYST) SRCFILE(SAMPLE) */
/* CRTCPGM PGM(INZPKEYST) MODULE(INZPKEYST) + */
/* BNDSPGM(QCCA/CSNBKSI) */
/* */
/* Note: authority to the CSNBKSI service program in the */
/* QCCA library is assumed. */
/* */
/* Common Cryptographic Architecture (CCA) verbs used: */
/* Keystore_Initialize (CSNBKSI) */
/* */
/*-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "csucincl.h" /* header file for CCA Cryptographic
Service Provider for iSeries */

int main(int argc, char *argv[])
{
/*-----*/
/* standard return codes */
/*-----*/

#define ERROR -1
#define OK 0

/*-----*/
/* standard CCA parameters */
/*-----*/

long return_code;
long reason_code;
long exit_data_length;
char exit_data[2];
char rule_array[4][8];
long rule_array_count;

/*-----*/
/* fields unique to this sample program */
/*-----*/
long file_name_length;
unsigned char description[4];
long description_length = 0;
unsigned char masterkey[8];

/*-----*/
/* Check if file name was passed */
/*-----*/
if(argc < 2)

```

```

    {
        printf("File name was not specified.\n");
        return ERROR;
    }

/*-----*/
/* fill in parameters for Keystore_Initialize */
/*-----*/
rule_array_count = 2;
memcpy((char*)rule_array,"CURRENT PKA      ",16);
file_name_length = strlen(argv[1]);

/*-----*/
/* Create key store file */
/*-----*/

    CSNBKSI(&return_code,
            &reason_code,
            &exit_data_length,
            exit_data,
            &rule_array_count,
            (char*)rule_array,
            &file_name_length,
            argv[1],
            &description_length,
            description,
            masterkey);

/*-----*/
/* Check the return code and display the result */
/*-----*/
    if (return_code != 0)
    {
        printf("Request failed with return/reason codes: %d/%d\n",
              return_code, reason_code);
        return ERROR;
    }
    else
    {
        printf("Key store file created\n");
        return OK;
    }
}

```

Example: ILE RPG program for initializing a key store for your Cryptographic Coprocessor: Change this program example to suit your needs for initializing a key store for your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* INZPKAST
D*
D* Create key store files for PKA keys.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*

```



```

D*
D* Note: Input format is more fully described in Chapter 2 of
D*     IBM CCA Basic Services Reference and Guide
D*     (SC31-8609) publication.
D*
D* Parameters: None
D*
D* Example:
D* CALL PGM(INZPKEYST) ('QGPL/PKAKEYS')
D*
D* Use these commands to compile this program on iSeries:
D* CRTPRGMOD MODULE(INZPKAST) SRCFILE(SAMPLE)
D* CRTPGM PGM(INZPKEYST) MODULE(INZPKEYST)
D*     BNDSRVPGM(QCCA/CSNBKSI)
D*
D* Note: Authority to the CSNBKSI service program in the
D*     QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Key_Store_Initialize (CSNBKSI)
D*
D*****
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S         9B 0
D*          ** Exit data
DEXITDATA S           4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY S          16
D*          ** File name length
DFILENAMELEN S         9B 0
D*          ** File name
DFILENAME S           21
D*          ** Description length
DDESCRIPLN S          9B 0
D*          ** Description
DDESCRIP S            16
D*          ** Master key part
DMASTERKEY S          24
D*
D*****
D* Prototype for Key_Store_Initialize (CSNBKSI)
D*****
DCSNBKSI PR
DRETCODE          9B 0
DRSNCODE          9B 0
DEXTDTALEN        9B 0
DEXTDTA           4
DRARRAYCT         9B 0
DRARRAY           16
DFILENMLN         9B 0
DFILENM           21
DDSCPLN           9B 0
DDSCRIP           16
DMSTRKY           24
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API

```

```

D*-----
DMSG          S          75    DIM(2) CTDATA PERRCD(1)
DMSGLENGTH    S          9B 0  INZ(75)
D             DS
DMSGTEXT      1          75
DFAILRETC     41          44
DFAILRSNC     46          49
DMESSAGEID    S          7     INZ(' ')
DMESSAGEFILE  S          21    INZ(' ')
DMSGKEY       S          4     INZ(' ')
DMSGTYPE      S          10    INZ('*INFO ')
DSTACKENTRY   S          10    INZ('* ')
DSTACKCOUNTER S          9B 0  INZ(2)
DERRCODE      DS
DBYTESIN      1          4B 0  INZ(0)
DBYTESOUT     5          8B 0  INZ(0)
D*
C*****
C* START OF PROGRAM *
C*****
C  *ENTRY      PLIST
C              PARM          FILENAME
C-----
C* Set the keyword in the rule array *
C-----
C              MOVE      'PKA '  RULEARRAY
C              MOVE      'CURRENT ' RULEARRAY
C              Z-ADD      2          RULEARRAYCNT
C-----
C* Set the description length *
C-----
C              Z-ADD      0          DESCRIPLEN
C-----
C* Find the file name length *
C-----
C              EVAL      FILENAMELEN = %LEN(%TRIM(FILENAME))
C*****
C* Call Key Store Initialize SAPI *
C*****
C              CALLP      CSNBKSI      (RETURNCODE:
C                                  REASONCODE:
C                                  EXITDATALEN:
C                                  EXITDATA:
C                                  RULEARRAYCNT:
C                                  RULEARRAY:
C                                  FILENAMELEN:
C                                  FILENAME:
C                                  DESCRIPLEN:
C                                  DESCRIP:
C                                  MASTERKEY)
C* *-----*
C* * Check the return code *
C* *-----*
C  RETURNCODE  IFGT      4
C* *-----*
C* * Send failure message *
C* *-----*
C              MOVE      MSG(1)      MSGTEXT
C              MOVE      RETURNCODE   FAILRETC
C              MOVE      REASONCODE   FAILRSNC
C              EXSR      SNDMSG
C              RETURN
C              ENDIF
C*
C* *-----*
C* * Send success message *
C* *-----*

```

```

C          MOVEL      MSG(2)      MSGTEXT
C          EXSR       SNDMSG
C*
C          SETON                               LR
C*
C*****
C* Subroutine to send a message
C*****
C          SNDMSG      BEGSR
C          CALL        'QMHSNDPM'
C          PARM        MESSAGEID
C          PARM        MESSAGEFILE
C          PARM        MSGTEXT
C          PARM        MSGLENGTH
C          PARM        MSGTYPE
C          PARM        STACKENTRY
C          PARM        STACKCOUNTER
C          PARM        MSGKEY
C          PARM        ERRCODE
C          ENDSR

```

**
CSNBKSI failed with return/reason codes 9999/9999.
The file was succesully initialized.

Create DES and PKA keys

You can use your Cryptographic Coprocessor to create two types of cryptographic keys.


- Data Encryption Standard (DES) keys base their content on a symmetric algorithm. This means that cryptography uses the same key value to encrypt and decrypt data. Use DES keys for “Encrypt or decrypt a file” on page 148, “Work with PINs” on page 154, and managing keys.

To create DES keys with your Cryptographic Coprocessor, write a program or change this program “Example: Creating a DES key with your Cryptographic Coprocessor.”

- Public key algorithm (PKA) keys base their content on an asymmetric algorithm, meaning that cryptography uses different keys for encryption and decryption. Use PKA keys for signing files using “Generate and verify a digital signature” on page 166 and for managing keys.

To create PKA keys with your Cryptographic Coprocessor, write a program or change this “Example: Creating a PKA key with your Cryptographic Coprocessor” on page 144.

Note: If you choose to use the program examples provided, change them to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

Store your DES and PKA keys in the key store file you created for them using “Initialize a key store file” on page 136. You can also store PKA keys in your Cryptographic Coprocessor. See the information at <http://www.ibm.com/security/cryptocards/html/library.shtml>  for more information on storing your keys in the hardware.

Example: Creating a DES key with your Cryptographic Coprocessor: Change this program example to suit your needs for creating a DES key with your Cryptographic Coprocessor.

```

/*-----*/
/* Generate DES keys in key store.                */
/*                                                */
/* COPYRIGHT      5769-SS1 (c) IBM Corp 1999      */
/*                                                */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these programs. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* EXPRESSLY DISCLAIMED. IBM provides no program services for */

```

```

/* these programs and files. */
/* */
/* Parameters: */
/* char * key label, 1 to 64 characters */
/* char * key store name, 1 to 21 characters in form 'lib/file' */
/*      (optional, see second note below) */
/* */
/* Examples: */
/* CALL PGM(KEYGEN) PARM('TEST.LABEL.1') */
/* */
/* CALL PGM(KEYGEN) PARM('MY.OWN.LABEL' 'QGPL/MYKEYSTORE') */
/* */
/* Note: This program assumes the device you want to use is */
/*       already identified either by defaulting to the CRP01 */
/*       device or has been explicitly named using the */
/*       Cryptographic_Resource_Allocate verb. Also this */
/*       device must be varied on and you must be authorized */
/*       to use this device description. */
/* */
/* If the key store name parameter is not provided, this */
/*       program assumes the key store file you will use is */
/*       already identified either by being specified on the */
/*       cryptographic device or has been previously named */
/*       using the Key_Store_Designate verb. Also you must be */
/*       authorized to add and update records in this file. */
/* */
/* Use the following commands to compile this program: */
/* ADDLIBLE LIB(QCCA) */
/* CRTCMOD MODULE(KEYGEN) SRCFILE(SAMPLE) */
/* CRTPGM  PGM(KEYGEN) MODULE(KEYGEN) + */
/*        BNSDRVPGM(QCCA/CSUAKSD QCCA/CSNBKRC QCCA/CSNBKGN) */
/* */
/* Note: authority to the CSUAKSD, CSNBKRC and CSNBKGN service */
/*       programs in the QCCA library is assumed. */
/* */
/* Common Cryptographic Architecture (CCA) verbs used: */
/* Key_Store_Designate (CSUAKSD) */
/* DES_Key_Record_Create (CSNBKRC) */
/* Key_Generate (CSNBKGN) */
/* */
/*-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "csucincl.h"          /* header file for CCA Cryptographic
                               Service Provider for iSeries */

int main(int argc, char *argv[])
{

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR -1
#define OK    0

/*-----*/
/* standard CCA parameters */
/*-----*/

    long return_code;
    long reason_code;
    long exit_data_length;
    char exit_data[2];
    long rule_array_count;

```

```

/*-----*/
/* fields unique to this sample program */
/*-----*/

long file_name_length;
char key_label[64];

/*-----*/
/* See if the user wants to specify which key store file to use */
/*-----*/

if(argc > 2)
{
    file_name_length = strlen(argv[2]);

    if((file_name_length > 0) &&
(file_name_length < 22))
    {
        rule_array_count = 1;

        CSUAKSD(&return_code,
&reason_code,
&exit_data_length,
exit_data,
&rule_array_count,
"DES ", /* rule_array, we are working with
DES keys in this sample program */
&file_name_length,
argv[2]); /* key store file name */

        if (return_code != 0)
        {
            printf("Key store designate failed for reason %d/%d\n\n",
return_code, reason_code);
            return ERROR;
        }
        else
        {
            printf("Key store designated\n");
            printf("SAPI returned %ld/%ld\n", return_code, reason_code);
        }
        else
        {
            printf("Key store file name is wrong length");
            return ERROR;
        }
    }
}
else; /* let key store file name default */

/*-----*/
/* Create a record in key store */
/*-----*/

memset(key_label, ' ', 64);
memcpy(key_label, argv[1], strlen(argv[1]));

CSNBKRC(&return_code,
&reason_code,
&exit_data_length,
exit_data,
key_label);

if (return_code != 0)
{
    printf("Record could not be added to key store for reason %d/%d\n\n",

```

```

        return_code, reason_code);
    return ERROR;
}
else
{
    printf("Record added to key store\n");
    printf("SAPI returned %ld/%ld\n", return_code, reason_code);
}

/*-----*/
/* Generate a key                                     */
/*-----*/

CSNBKGN(&return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        "OP ",          /* operational key is requested */
        "SINGLE ",      /* single length key requested */
        "DATA ",       /* Data encrypting key requested */
        " ",           /* second value must be blanks when
key form requests only one key */
        "\0",          /* key encrypting key is null for
operational keys */
        "\0",          /* key encrypting key is null since
only one key is being requested */
        key_label,     /* store generated key in key store*/
        "\0");        /* no second key is requested */

if (return_code != 0)
{
    printf("Key generation failed for reason %d/%d\n",
        return_code, reason_code);
    return ERROR;
}
else
{
    printf("Key generated and stored in key store\n");
    printf("SAPI returned %ld/%ld\n", return_code, reason_code);
    return OK;
}
}

```

Example: Creating a PKA key with your Cryptographic Coprocessor: Change this program example to suit your needs for creating a PKA key with your Cryptographic Coprocessor.

```

/*-----*/
/* Generate PKA keys in key store.                     */
/*                                                     */
/* COPYRIGHT      5769-SS1 (c) IBM Corp 1999         */
/*                                                     */
/* This material contains programming source code for your
/* consideration. These examples have not been thoroughly
/* tested under all conditions. IBM, therefore, cannot
/* guarantee or imply reliability, serviceability, or function
/* of these programs. All programs contained herein are
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
/* EXPRESSLY DISCLAIMED. IBM provides no program services for
/* these programs and files.
/*
/* Parameters:
/* char * key label, 1 to 64 characters
/*
/* Examples:
/* CALL PGM(PKAKEYGEN) PARM('TEST.LABEL.1')
/*

```

```

/*                                                                    */
/* Note: This program assumes the card you want to load is          */
/* already identified either by defaulting to the CRP01             */
/* device or has been explicitly named using the                   */
/* Cryptographic_Resource_Allocate verb. Also this                */
/* device must be varied on and you must be authorized            */
/* to use this device description.                                  */
/*                                                                    */
/* This program also assumes the key store file you will           */
/* use is already identified either by being specified on          */
/* the cryptographic device or has been explicitly named           */
/* using the Key_Store_Designate verb. Also you must be           */
/* authorized to add and update records in this file.              */
/*                                                                    */
/* Use the following commands to compile this program:            */
/* ADDLIB LIB(QCCA)                                                */
/* CRTCMOD MODULE(PKAKEYGEN) SRCFILE(SAMPLE)                       */
/* CRTPGM PGM(PKAKEYGEN) MODULE(PKAKEYGEN) +                       */
/* BNSRVPGM(QCCA/CSNDKRC QCCA/CSNDPKG)                             */
/*                                                                    */
/* Note: authority to the CSNDKRC and CSNDPKG service programs    */
/* in the QCCA library is assumed.                                  */
/*                                                                    */
/* Common Cryptographic Architecture (CCA) verbs used:           */
/* PKA_Key_Record_Create (CSNDKRC)                                 */
/* PKA_Key_Generate (CSNDPKG)                                     */
/*                                                                    */
/*-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "csucincl.h"          /* header file for CCA Cryptographic
                               Service Provider for iSeries */

int main(int argc, char *argv[])
{

/*-----*/
/* standard return codes                                          */
/*-----*/

#define ERROR -1
#define OK    0

/*-----*/
/* standard CCA parameters                                        */
/*-----*/

    long return_code;
    long reason_code;
    long exit_data_length;
    char exit_data[2];
    char rule_array[4][8];
    long rule_array_count;

/*-----*/
/* fields unique to this sample program                          */
/*-----*/

    char key_label[64];          /* identify record in key store to
                                 hold generated key */
    #pragma pack (1)

    typedef struct rsa_key_token_header_section {
        char token_identifier;
        char version;

```

```

    short key_token_struct_length;
    char reserved_1[4];
} rsa_key_token_header_section;

typedef struct rsa_private_key_1024_bit_section {
    char section_identifier;
    char version;
    short section_length;
    char hash_of_private_key[20];
    short reserved_1;
    short master_key_verification_pattern;
    char key_format_and_security;
    char reserved_2;
    char hash_of_key_name[20];
    char key_usage_flag;
    char rest_of_private_key[312];
} rsa_private_key_1024_bit_section;

typedef struct rsa_public_key_section {
    char section_idenfifer;
    char version;
    short section_length;
    short reserved_1;
    short exponent_field_length;
    short modulus_length;
    short modulus_length_in_bytes;
    char exponent;
} rsa_public_key_section;

struct {
    rsa_key_token_header_section    rsa_header;
    rsa_private_key_1024_bit_section rsa_private_key;
    rsa_public_key_section          rsa_public_key;
} key_token;

struct {
    short modlen;
    short modlenfld;
    short pubexplen;
    short prvexplen;
    long  pubexp;
} prvPubl;

#pragma pack ()

long key_struct_length;
long zero = 0;
long key_token_length;

long regen_data_length;
long generated_key_id_length;

/*-----*/
/* Create record in key store                               */
/*-----*/
rule_array_count = 0;
key_token_length = 0;
memset(key_label, ' ', 64);
memcpy(key_label, argv[1], strlen(argv[1]));

CSNDKRC(&return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        "\0",          /* rule_array          */

```



```

    key_label,
    &key_token_length,
    "\0");          /* key token          */

if (return_code != 0)
{
    printf("Record could not be added to key store for reason %d/%d\n",
        return_code, reason_code);
    return ERROR;
}
else
{
    printf("Record added to key store\n");
    printf("SAPI returned %ld/%ld\n", return_code, reason_code);
}

/*-----*/
/* Build a key token, needed to generate PKA key          */
/*-----*/
memset(&key_token, 0X00, sizeof(key_token));

key_token.rsa_header.token_identifer = 0X1E; /* external token */
key_token.rsa_header.key_token_struct_length = sizeof(key_token);

key_token.rsa_private_key.section_identifer =
    0X02;          /* RSA private key */
key_token.rsa_private_key.section_length =
    sizeof(rsa_private_key_1024_bit_section);
key_token.rsa_private_key.key_usage_flag = 0X80;

key_token.rsa_public_key.section_identifer = 0X04; /* RSA public key */
key_token.rsa_public_key.section_length =
    sizeof(rsa_public_key_section);
key_token.rsa_public_key.exponent_field_length = 1;
key_token.rsa_public_key.modulus_length = 512;
key_token.rsa_public_key.exponent = 0x03;

key_token_length = sizeof(key_token);

printf("Key token built\n");

/*-----*/
/* Generate a key          */
/*-----*/

rule_array_count = 1;
regen_data_length = 0;
/* key_token_length = 64; */
generated_key_id_length = 2500;

CSNDPKG(&return_code,
    &reason_code,
    &exit_data_length,
    exit_data,
    &rule_array_count,
    "MASTER ",          /* rule_array          */
    &regen_data_length,
    "\0",          /* regeneration_data, none needed */
    &key_token_length, /* skeleton_key_token_length */
    (char *)&key_token, /* skeleton_key_token built above */
    "\0",          /* transport_id, only needed for
                    XPORT keys          */
    &generated_key_id_length,
    key_label); /* generated_key_id, store generated
                key in key store          */

if (return_code != 0)

```

```

{
    printf("Key generation failed for reason %d/%d\n\n",
        return_code, reason_code);
    return ERROR;
}
else
{
    printf("Key generated and stored in key store\n");
    printf("SAPI returned %d/%d\n\n", return_code, reason_code);
    return OK;
}
}

```

Encrypt or decrypt a file

One of the more practical uses for your Cryptographic Coprocessor is encrypting and decrypting data files. You can use one of these cryptographic methods to protect a file:

- Treat the whole file as a string of bytes (which is the method the program example uses).
- Encrypt each record or part of each record.

Write your own program or change the techniques in this program “Example: Encrypting data with your Cryptographic Coprocessor” to protect data in many different formats, not just data files.

Example: Encrypting data with your Cryptographic Coprocessor: Change this program example to suit your needs for encrypting data with your Cryptographic Coprocessor.

```

/*-----*/
/*
/* Sample C program for enciphering data in a file.
/*
/* COPYRIGHT      5769-SS1 (c) IBM Corp 1999      */
/*
/* This material contains programming source code for your
/* consideration. These examples have not been thoroughly
/* tested under all conditions. IBM, therefore, cannot
/* guarantee or imply reliability, serviceability, or function
/* of these programs. All programs contained herein are
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
/* EXPRESSLY DISCLAIMED. IBM provides no program services for
/* these programs and files.
/*
/* Parameters:
/* char * key label, 1 to 64 characters
/* char * input file name, 1 to 21 characters (lib/file)
/* char * output file name, 1 to 21 characters (lib/file)
/*
/* Example:
/* CALL PGM(ENCFILE) PARM( 'MY.KEY.LABEL' 'QGPL/MYDATA' +
/*                          'QGPL/CRYPTDATA' )
/*
/* Note: This program assumes the device you want to use is
/* already identified either by defaulting to the CRP01
/* device or has been explicitly named using the
/* Cryptographic_Resource_Allocate verb. Also this
/* device must be varied on and you must be authorized
/* to use this device description.
/*
/* This program assumes the key store file you will use is
/* already identified either by being specified on the
/* cryptographic device or has been previously named
/* using the Key_Store_Designate verb. Also you must be
/* authorized to add and update records in this file.
/*
/* The output file should NOT have key fields since all
/* data in the file will be encrypted and therefore trying
/*

```

```

/*      to sort the data will be meaningless.          */
/*      (This is NOT checked by the program)          */
/*                                                    */
/* Use the following commands to compile this program: */
/* ADDLIB LIB(QCCA)                                  */
/* CRTCMOD MODULE(ENCFILE) SRCFILE(SAMPLE)          */
/* CRTPGM PGM(ENCFILE) MODULE(ENCFILE) +          */
/*      BNDSRVPGM(QCCA/CSNBENC)                    */
/*                                                    */
/* Note: authority to the CSNBENC service program in the */
/*      QCCA library is assumed.                    */
/*                                                    */
/* Common Cryptographic Architecture (CCA) verbs used: */
/* Encipher (CSNBENC)                               */
/*                                                    */
/*-----*/

/*-----*/
/* Retrieve various structures/utilities that are used in program. */
/*-----*/

#include <stdio.h>          /* Standard I/O header.          */
#include <stdlib.h>        /* General utilities.           */
#include <stddef.h>        /* Standard definitions.        */
#include <string.h>        /* String handling utilities.    */
#include "csucincl.h"      /* header file for CCA Cryptographic
                          Service Provider for iSeries */

/*-----*/
/* Declares for working with files.                  */
/*-----*/
#include <xxfdbk.h>        /* Feedback area structures.    */
#include <recio.h>        /* Record I/O routines          */
_RFILE          *dbfptr;  /* Pointer to database file.    */
_RFILE          *dbfptre; /* Pointer to database file.    */
_RIOFB_T        *db_fdbk; /* I/O Feedback - data base file */
_XXOPFB_T       *db_opfb;
_XXOPFB_T       *db_opfbe;

/*-----*/
/* Declares for working with user space objects.    */
/*-----*/
#include "qusptrus.h"
#include "qusrtus.h"
#include "qusdltus.h"
#define USSPC_ATTR          "PF          "
#define USSPC_INIT_VAL     0x40
#define USSPC_AUTH         "*EXCLUDE "
#define USSPC_TEXT         "Sample user space"
#define USSPC_REPLACE      "*YES          "

char    space_name[21] = "PLAINTXT QTEMP          "; /* Name of user
                                                    space for plain text */
char    cipher_name[21] = "CIPHER QTEMP          "; /* Name for user
                                                    space containing ciphertext */

struct {
    /* Error code structure required for */
    /* the User Space API's.           */
    int in_len;          /* the length of the error code.    */
    int out_len;         /* the length of the exception data. */
    char excp_id[7];     /* the Exception ID.                */
    char rev;            /* Reserved Field.                  */
    char excp_data[120]; /* the output data associated       */
} error_code;          /* the exception ID.                */

```

```

char      ext_atr[11] = USSPC_ATTR; /* Space attribute      */
char      initial_val = USSPC_INIT_VAL; /* Space initial value */
char      auth[11]   = USSPC_AUTH; /* Space authority     */
char      desc[51]   = USSPC_TEXT; /* Space text          */
char      replace[11] = USSPC_REPLACE; /*Space replace attribute*/

/*-----*/
/* Start of mainline code. */
/*-----*/
int main(int argc, char *argv[])
{

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR -1
#define OK    0

/*-----*/
/* standard CCA parameters */
/*-----*/

    long return_code;
    long reason_code;
    long exit_data_length;
    char exit_data[2];
    long rule_array_count;

    char      *user_space_ptr;
    char      *user_space;
    char      *cipher_spc;
    long      file_bytes;
    long      i;
    long      j;
    char      key_label[64];

    long      text_len, pad_character;
    char      initial_vector[8];
    char      chaining_vector[18];

/*-----*/
/* Open database files. */
/*-----*/

    if (argc < 4) /* were the correct number
                  of parameters passed? */

    {
        printf("This program needs 3 parameters - ");
        printf("key label, input file name, output file name\n");

        return ERROR;
    }

    else
    {

        file_bytes = 0; /* Set initial number of
                       bytes to encipher to 0 */

        /* Open the input file. If the file pointer, dbfptr is not
           NULL, then the file was successfully opened. */

```

```

if (( dbfptr = _Ropen(argv[2], "rr riofb=n"))
    != NULL)
{
/*-----*/
/* Determine the number of bytes that will be enciphered. */
/*-----*/
    db_opfb = _Ropnfbk( dbfptr ); /* Get pointer to the File
                                   open feedback area. */

    file_bytes = db_opfb->num_records *
                db_opfb->pgm_record_len
                + 1; /* 1 is added to prevent an
                       end of space error */

    j = db_opfb->num_records; /* Save number of records*/
/*-----*/
/* Create user space and get pointer to it. */
/*-----*/
    error_code.in_len = 136; /* Set length of error */
                             /* structure. */
    QUSDLTUS(space_name,&error_code); /* Delete the user space
                                       if it already exists. */

/* Create the plaintext user space object */
    QUSCRTUS(space_name,ext_atr,file_bytes,
             &initial_val,auth,
             desc, replace,&error_code);

    error_code.in_len = 48; /* Set length of error
                             structure */

    QUSPTRUS(space_name, /* Retrieve a pointer to */
             (void *)&user_space, /* the user space. */
             (char*)&error_code);

    user_space_ptr = user_space; /* Make copy of pointer */

    error_code.in_len = 136; /* Set length of error */
                             /* structure. */
    QUSDLTUS(cipher_name,&error_code); /* Delete cipher space
                                       if already exists. */

/* Create ciphertext user space object */
    QUSCRTUS(cipher_name,ext_atr,
             file_bytes,&initial_val,auth,
             desc, replace,&error_code);

    error_code.in_len = 48; /* Set length of error */
                             /* structure */
    QUSPTRUS(cipher_name, /* Retrieve pointer to */
             (void *)&cipher_spc, /* ciphertext user space */
             (char*)&error_code);

/*-----*/
/* Read file and fill space */
/*-----*/
    for (i=1; i<=j; i++) /* Repeat for each record */
    {
        /* Read a record and place in user space. */
        db_fdbk = _Rreadn(dbfptr, user_space_ptr,
                        db_opfb->pgm_record_len, __DFT);

        /* Move the user space ahead the length of a record */
        user_space_ptr = user_space_ptr +
            db_opfb->pgm_record_len;
    }
}

```

```

    }

    if (dbfptr != NULL)                /* Close the file. */
        _Rclose(dbfptr);
/*-----*/
/* Encrypt data in space                */
/*-----*/

    memset((char *)key_label,' ',64); /* Initialize key label
                                     to all blanks.      */
    memcpy((char *)key_label,        /* Copy key label parm */
           argv[1],strlen(argv[1]));

    text_len = file_bytes - 1;
    rule_array_count = 1;
    pad_character = 40;
    exit_data_length = 0;
    memset((char *)initial_vector,'\0',8);

    /* Encipher data in ciphertext user space */
    CSNBENC(&return_code,
            &reason_code,
            &exit_data_length,
            exit_data,
            key_label,
            &text_len,
            user_space,
            initial_vector,
            &rule_array_count,
            "CBC", /* rule_array */
            &pad_character,
            chaining_vector,
            cipher_spc );

    if (return_code == 0) {
/*-----*/
/* Open output file                */
/*-----*/
        if (( dbfptre = _Ropen(argv[3],
                               "wr riofb=n")) != NULL)
        {
            db_opfbe = _Ropnfbk( dbfptr ); /* Get pointer to
                                           the File open feedback
                                           area. */
            if(text_len % db_opfbe->pgm_record_len != 0)
            {
                printf("encrypted data will not fit into ");
                printf("an even number of records\n");

                if (dbfptre != NULL) /* Close the file. */
                    _Rclose(dbfptre);

                /*-----*/
                /* Delete both user spaces.                */
                /*-----*/
                error_code.in_len = 136; /* Set length of
                                         error structure. */
                QUSDLTUS(space_name,&error_code); /* Delete the
                                                  user space */
                QUSDLTUS(cipher_name,&error_code); /* Delete
                                                  ciphertext space */

                return ERROR;
            }
        }

/*-----*/
/* Write data from space to file.    */
/*-----*/

```

```

/*-----*/
        user_space_ptr = cipher_spc; /* Save pointer to
                                     cipher space. */

j = text_len / db_opfbe->pgm_record_len; /* find
how many records
are needed to store
result in output

        file */
        for (i=1; i<=j; i++) /* Repeat for each
                               record */
        {
            /* Write data to output file */
            db_fdbk = _Rwrite(dbfptre, user_space_ptr,
                             db_opfbe->pgm_record_len);

            /* Advance pointer ahead the length of a record */
            user_space_ptr = user_space_ptr +
                db_opfbe->pgm_record_len;
        }
        if (dbfptre != NULL) /* Close the file */
            _Rclose(dbfptre);

    } /* end of open open
      output file */
    else
    {
        printf("Output file %s could not be opened\n",
            argv[3]);

        /*-----*/
        /* Delete both user spaces. */
        /*-----*/
        error_code.in_len = 136; /* Set length of
                                  error structure. */
        QUSDLTUS(space_name,&error_code); /* Delete the
                                           user space */
        QUSDLTUS(cipher_name,&error_code); /* Delete
                                           ciphertext space */
        return ERROR;
    }

    } /* If return code = 0 */
    else
    {
        printf("Bad return/reason code : %d/%d \n",
            return_code,reason_code);
        /*-----*/
        /* Delete both user spaces. */
        /*-----*/
        error_code.in_len = 136; /* Set length of
                                  error structure. */
        QUSDLTUS(space_name,&error_code); /* Delete the
                                           user space */
        QUSDLTUS(cipher_name,&error_code); /* Delete
                                           ciphertext space */
        return ERROR;
    }

/*-----*/
/* Delete both user spaces. */
/*-----*/
error_code.in_len = 136; /* Set length of
                          error structure. */
QUSDLTUS(space_name,&error_code); /* Delete the user
space */
QUSDLTUS(cipher_name,&error_code); /* Delete ciphertext

```

```

                                space          */
        }                                /* End of open
                                input file     */
    else
    {
        printf("Input file %s could not be opened\n", argv[2]);
        return ERROR;
    }
}                                /* argv[] == null */
return OK;
}

```

Work with PINs

A financial institution uses personal identification numbers (PINs) to authorize personal financial transactions for its customers. A PIN is similar to a password except that a PIN consists of decimal digits and is normally a cryptographic function of an associated account number. You can use your Cryptographic Coprocessor to work with PINs.

To work with PINs, write a program or change this program "Example: Working with PINs on your Cryptographic Coprocessor."

Note: If you choose to use the program example provided, change it to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

Example: Working with PINs on your Cryptographic Coprocessor: Change this program example to suit your needs for working with PINs on your Cryptographic Coprocessor.

```

F*****
F* PINSAMPLE
F*
F* Sample program that shows the use of the appropriate
F* CCA Security API (SAPI) verbs for generating and verifying
F* PINS
F*
F* The keys are created by first building a key token
F* and then importing key parts using Key_Part_Import.
F* Four keys are created each with a different
F* key type - PINGEN, PINVER, IPINENC, and OPINENC. The
F* PINGEN key will be used to generate a Clear PIN with the
F* Clear_PIN_Generate verb. The OPINENC key will be used
F* to encrypt the PIN with the Clear_PIN_Encrypt verb.
F* The Encrypted_PIN_Verify with verify that the PIN is good
F* using the IPINENC key (to decrypt) and the PINVER key
F* to verify the PIN.
F*
F* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999
F*
F* This material contains programming source code for your
F* consideration. These example has not been thoroughly
F* tested under all conditions. IBM, therefore, cannot
F* guarantee or imply reliability, serviceability, or function
F* of these programs. All programs contained herein are
F* provided to you "AS IS". THE IMPLIED WARRANTIES OF
F* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
F* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
F* these programs and files.
F*
F*
F* Note: Input format is more fully described in Chapter 2 of
F* IBM CCA Basic Services Reference and Guide
F* (SC31-8609) publication.
F*
F* Parameters:

```



```

F* none.
F*
F* Example:
F* CALL PGM(PINSAMPLE)
F*
F* Use these commands to compile this program on iSeries:
F* CRTRPGMOD MODULE(PINSAMPLE) SRCFILE(SAMPLE)
F* CRTPGM PGM(PINSAMPLE) MODULE(PINSAMPLE)
F* BNDSRVPGM(QCCA/CSNBKPI QCCA/CSNBPGN +
F* QCCA/CSNBCPE QCCA/CSNBPVR)
F*
F* Note: Authority to the CSNBKPI, CSNBPGN, CSNBCPE, and
F* CSNBPVR service programs in the QCCA library is assumed.
F*
F* The Common Cryptographic Architecture (CCA) verbs used are
F* Key_Part_Import (CSNBKPI), Clear_PIN_Generate (CSNBPGN),
F* Clear_PIN_Encrypt (CSNBCPE), and Encrypted_PIN_Verify (CSNBPVR).
F*
F* Note: This program assumes the card you want to load is
F* already identified either by defaulting to the CRP01
F* device or has been explicitly named using the
F* Cryptographic_Resource_Allocate verb. Also this
F* device must be varied on and you must be authorized
F* to use this device description.
F*
F*****
F* Declare parameters that are common to all of the CCA verbs
F*
F*****
DRETURNCODE S 9B 0
DREASONCODE S 9B 0
DEXITDATALEN S 9B 0
DEXITDATA S 4
DRULEARRAYCNT S 9B 0
DRULEARRAY S 16
D*
D*****
D* Declare Key tokens used by this program
D*
D*****
DIPINKEY S 64
DOPINKEY S 64
DPINGENKEY S 64
DPINVERKEY S 64
DKEYTOKEN DS
DKEYFORM 1 1
DKEYVERSION 5 5
DKEYFLAG1 7 7
DKEYVALUE 17 32
DKEYCV 33 48
DKEYTVV 61 64B 0
DTOKENPART1 1 16
DTOKENPART2 17 32
DTOKENPART3 33 48
DTOKENPART4 49 64
DKEYTVV1 1 4B 0
DKEYTVV2 5 8B 0
DKEYTVV3 9 12B 0
DKEYTVV4 13 16B 0
DKEYTVV5 17 20B 0
DKEYTVV6 21 24B 0
DKEYTVV7 25 28B 0
DKEYTVV8 29 32B 0
DKEYTVV9 33 36B 0
DKEYTVV10 37 40B 0
DKEYTVV11 41 44B 0
DKEYTVV12 45 48B 0

```

```

DKEYTVV13          49    52B 0
DKEYTVV14          53    56B 0
DKEYTVV15          57    60B 0
D*
D*****
D* Declare parameters unique to Key_Part_Import
D*
D*****
DCLEARKEY          S          16
D*
D*****
D* Declare parameters unique to Clear_PIN_Generate,
D* Clear_PIN_Encrypt, and Encrypted_PIN_Verify
D*****
DPINLEN            S          9B 0
DPINCKL            S          9B 0
DSEQUNUMBER        S          9B 0
DCPIN              S          16
DEPIN              S          16
DPAN               S          12
DDATAARRAY         DS
DDECTABLE          1          16
DVALDATA           17         32
DCLRPIN            33         48
DPROFILE           DS
DPINFORMAT         1          8
DFORMATCONTROL     9          16
DPADDIGIT          17         24
D*
D*****
D* Declare variables used for creating a control vector and
D* clear key.
D*****
DBLDKEY            DS
DLEFTHALF          1          8
DLEFTHALFA         1          4B 0
DLEFTHALFB         5          8B 0
DRIGHTHALF         9          16
D*
D*
D*****
D* Prototype for Key Part Import (CSNBKPI)
D*****
DCSNBKPI           PR
DRETCODE           9B 0
DRSNCODE           9B 0
DEXTDTALEN         9B 0
DEXTDTA            4
DRARRAYCT          9B 0
DRARRAY            16
DCLRKEY            16
DIMPKEY            64
D*
D*****
D* Prototype for Clear PIN Generate (CSNBPGN)
D*****
DCSNBPGN           PR
DRETCODE           9B 0
DRSNCODE           9B 0
DEXTDTALEN         9B 0
DEXTDTA            4
DPINGEN            64
DRARRAYCT          9B 0
DRARRAY            16
DPINL              9B 0
DPINCHKLEN         9B 0
DDTAARRY           48

```

```

DRESULT                                16
D*
D*****
D* Prototype for Clear PIN Encrypt (CSNBCPE)
D*****
DCSNBCPE                                PR
DRETCODE                                9B 0
DRSNCODE                                9B 0
DEXTDTALEN                              9B 0
DEXTDTA                                  4
DPINENC                                  64
DRARRAYCT                                9B 0
DRARRAY                                  16
DCLRPIN                                  16
DPINPROFILE                              24
DPANDATA                                  12
DSEQN                                     9B 0
DEPINBLCK                                 8
D*
D*****
D* Prototype for Encrypted PIN Verify (CSNBPVR)
D*****
DCSNBPVR                                PR
DRETCODE                                9B 0
DRSNCODE                                9B 0
DEXTDTALEN                              9B 0
DEXTDTA                                  4
DPINENC                                  64
DPINVER                                  64
DPINPROFILE                              24
DPANDATA                                  12
DEPINBLCK                                 8
DRARRAYCT                                9B 0
DRARRAY                                  16
DCHECKLEN                                9B 0
DDTAARRAY                                24
D*
D*****
D* Declares for sending messages to job log
D*****
DFAILMESSAGE    S           50
DGOODMESSAGE    S           50
DFAILMSG        DS
DFAILMSGTEXT    1           50
DFAILRETC       41         44
DFAILRSNC       46         49
DRETSTRUCT      DS
DRETCODE        1          4I 0
DSLASH          5           5   INZ('/')
DRSNCODE        6           9I 0
DFAILMSGLENGTH S           9B 0 INZ(49)
DGOODMSGLENGTH S           9B 0 INZ(29)
DMESSAGEID      S           7   INZ(' ')
DMESSAGEFILE    S          21   INZ(' ')
DMSGKEY         S           4   INZ(' ')
DMSGTYPE        S          10   INZ('*INFO ')
DSTACKENTRY     S          10   INZ('* ')
DSTACKCOUNTER   S          9B 0 INZ(2)
DERRCODE        DS
DBYTESIN        1          4B 0 INZ(0)
DBYTESOUT       5          8B 0 INZ(0)
C                EVAL      FAILMESSAGE = '***** failed with return+
C                                     /reason codes 9999/9999'
C                EVAL      GOODMESSAGE = 'PIN Validation was successful'
C*****
C* START OF PROGRAM *
C* *

```

```

C*****
C* Build a PINGEN key token
C*
C*****
C* Zero out the key token to start with
C*
C          Z-ADD      0          KEYTVV1
C          Z-ADD      0          KEYTVV2
C          Z-ADD      0          KEYTVV3
C          Z-ADD      0          KEYTVV4
C          MOVE      TOKENPART1  TOKENPART2
C          MOVE      TOKENPART1  TOKENPART3
C          MOVE      TOKENPART1  TOKENPART4
C*
C* Set the form, version, and flag byte
C*
C          BITON      '7'        KEYFORM
C          BITON      '67'       KEYVERSION
C          BITON      '1'        KEYFLAG1
C*
C* The control vector for a PINGEN key that has the key part
C* flag set is (in hex):
C*
C*          00227E00  03480000  00227E00  03280000
C*
C* If each 4 byte hex part is converted to decimal you get:
C*
C*          2260480  55050240  2260480  52953088
C*
C* Build the control vector by placing the decimal number in
C* the appropriate half of the control vector field.
C*****
C          Z-ADD      2260480     LEFTHALFA
C          Z-ADD      55050240    LEFTHALFB
C          MOVE      LEFTHALF     KEYCV
C          Z-ADD      2260480     LEFTHALFA
C          Z-ADD      52953088    LEFTHALFB
C          MOVE      LEFTHALF     KEYCV
C*
C* Calculate the Token Validation value by adding every 4 bytes
C* and storing the result in the last 4 bytes.
C*
C          ADD      KEYTVV1       KEYTVV
C          ADD      KEYTVV2       KEYTVV
C          ADD      KEYTVV3       KEYTVV
C          ADD      KEYTVV4       KEYTVV
C          ADD      KEYTVV5       KEYTVV
C          ADD      KEYTVV6       KEYTVV
C          ADD      KEYTVV7       KEYTVV
C          ADD      KEYTVV8       KEYTVV
C          ADD      KEYTVV9       KEYTVV
C          ADD      KEYTVV10      KEYTVV
C          ADD      KEYTVV11      KEYTVV
C          ADD      KEYTVV12      KEYTVV
C          ADD      KEYTVV13      KEYTVV
C          ADD      KEYTVV14      KEYTVV
C          ADD      KEYTVV15      KEYTVV
C*
C* Copy token to PINGENKEY
C*
C          MOVE      KEYTOKEN     PINGENKEY
C*
C*****
C* Build a PINVER key token
C*
C* The control vector for a PINVER key that
C* has the key part flag set is (in hex):

```

```

C*
C*      00224200 03480000 00224200 03280000
C*
C* If each 4 byte hex part is converted to decimal you get:
C*
C*      2260480 55050240 2260480 52953088
C*
C* Build the control vector by placing the decimal number in
C* the appropriate half of the control vector field.
C      Z-ADD      2245120      LEFTHALFA
C      Z-ADD      55050240     LEFTHALFB
C      MOVE      LEFTHALF      KEYCV
C      Z-ADD      2245120      LEFTHALFA
C      Z-ADD      52953088     LEFTHALFB
C      MOVE      LEFTHALF      KEYCV
C*
C* Calculate the Token Validation value by adding every 4 bytes
C* and storing the result in the last 4 bytes.
C*
C      Z-ADD      0            KEYTVV
C      ADD      KEYTVV1      KEYTVV
C      ADD      KEYTVV2      KEYTVV
C      ADD      KEYTVV3      KEYTVV
C      ADD      KEYTVV4      KEYTVV
C      ADD      KEYTVV5      KEYTVV
C      ADD      KEYTVV6      KEYTVV
C      ADD      KEYTVV7      KEYTVV
C      ADD      KEYTVV8      KEYTVV
C      ADD      KEYTVV9      KEYTVV
C      ADD      KEYTVV10     KEYTVV
C      ADD      KEYTVV11     KEYTVV
C      ADD      KEYTVV12     KEYTVV
C      ADD      KEYTVV13     KEYTVV
C      ADD      KEYTVV14     KEYTVV
C      ADD      KEYTVV15     KEYTVV
C*
C* Copy token to PINVERKEY
C*
C      MOVE      KEYPIN      PINVERKEY
C*
C*
C*****
C* Build an IPINENC key token
C*
C* The control vector for an IPINENC key that
C* has the key part flag set is (in hex):
C*
C*      00215F00 03480000 00215F00 03280000
C*
C* If each 4 byte hex part is converted to decimal you get:
C*
C*      2187008 55050240 2187008 52953088
C*
C*****
C* Build the control vector by placing the decimal number in
C* the appropriate half of the control vector field.
C*****
C      Z-ADD      2187008      LEFTHALFA
C      Z-ADD      55050240     LEFTHALFB
C      MOVE      LEFTHALF      KEYCV
C      Z-ADD      2187008      LEFTHALFA
C      Z-ADD      52953088     LEFTHALFB
C      MOVE      LEFTHALF      KEYCV
C*
C* Calculate the Token Validation value by adding every 4 bytes
C* and storing the result in the last 4 bytes.
C*

```

```

C          Z-ADD      0          KEYTVV
C          ADD        KEYTVV1    KEYTVV
C          ADD        KEYTVV2    KEYTVV
C          ADD        KEYTVV3    KEYTVV
C          ADD        KEYTVV4    KEYTVV
C          ADD        KEYTVV5    KEYTVV
C          ADD        KEYTVV6    KEYTVV
C          ADD        KEYTVV7    KEYTVV
C          ADD        KEYTVV8    KEYTVV
C          ADD        KEYTVV9    KEYTVV
C          ADD        KEYTVV10   KEYTVV
C          ADD        KEYTVV11   KEYTVV
C          ADD        KEYTVV12   KEYTVV
C          ADD        KEYTVV13   KEYTVV
C          ADD        KEYTVV14   KEYTVV
C          ADD        KEYTVV15   KEYTVV
C*
C* Copy token to IPINENC
C*
C          MOVE      KEYTOKEN    IPINKEY
C*
C*
C*****
C* Build an OPINENC key token
C*
C* The control vector for an OPINENC key that
C* has the key part flag set is (in hex):
C*
C*      00247700  03480000  00247700  03280000
C*
C* If each 4 byte hex part is converted to decimal you get:
C*
C*      2389760  55050240  2389760  52953088
C*
C*****
C* Build the control vector by placing the decimal numbers in
C* the appropriate half of the control vector field.
C*****
C          Z-ADD      2389760    LEFTHALFA
C          Z-ADD      55050240   LEFTHALFB
C          MOVE      LEFTHALF    KEYCV
C          Z-ADD      2389760    LEFTHALFA
C          Z-ADD      52953088   LEFTHALFB
C          MOVE      LEFTHALF    KEYCV
C*
C* Calculate the Token Validation value by adding every 4 bytes
C* and storing the result in the last 4 bytes.
C*
C          Z-ADD      0          KEYTVV
C          ADD        KEYTVV1    KEYTVV
C          ADD        KEYTVV2    KEYTVV
C          ADD        KEYTVV3    KEYTVV
C          ADD        KEYTVV4    KEYTVV
C          ADD        KEYTVV5    KEYTVV
C          ADD        KEYTVV6    KEYTVV
C          ADD        KEYTVV7    KEYTVV
C          ADD        KEYTVV8    KEYTVV
C          ADD        KEYTVV9    KEYTVV
C          ADD        KEYTVV10   KEYTVV
C          ADD        KEYTVV11   KEYTVV
C          ADD        KEYTVV12   KEYTVV
C          ADD        KEYTVV13   KEYTVV
C          ADD        KEYTVV14   KEYTVV
C          ADD        KEYTVV15   KEYTVV
C*
C* Copy token to OPINENC
C*

```

```

C          MOVE      KEYTOKEN      OPINKEY
C*
C*
C*****
C*
C* Clear key value for PINGEN/PINVER form will be:
C*
C*   01234567 01765432  01234567 01765432
C*
C* The key will be imported into two parts that get excluded
C* OR'ed together. This program uses as key parts:
C*
C*   00224466 00775533  00224466 00775533  and
C*
C*   01010101 01010101  01010101 01010101
C*
C* Converting these to decimal results in
C*
C*   2245734 7820595  2245734 7820595  and
C*
C*   16843009 16843009  16843009 16843009
C*
C* In this example, the left half of the key is the same as
C* the right half. PIN keys in CCA are double length keys.
C* However, some implementation of DES (including Cryptographic
C* Support/400) use single length keys for PINs. If both
C* halves of a double are the same, then they produce the
C* same output as a single length key, thereby allowing you
C* to exchange data with non-CCA systems.
C*****
C* Import the PINGEN key
C*****
C          MOVE      'FIRST  '      RULEARRAY
C          Z-ADD      1              RULEARRAYCNT
C*****
C* Build the next clear key part by placing the decimal numbers
C* in the appropriate half of the clear key field.
C*****
C          Z-ADD      16843009      LEFTHALFA
C          Z-ADD      16843009      LEFTHALFB
C          MOVE      LEFTHALF      CLEARKEY
C          MOVE      LEFTHALF      CLEARKEY
C*****
C* Call Key Part Import the first time for the PINGEN key
C*****
C          CALLP      CSNBKPI      (RETURNCODE:
C                                REASONCODE:
C                                EXITDATALEN:
C                                EXITDATA:
C                                RULEARRAYCNT:
C                                RULEARRAY:
C                                CLEARKEY:
C                                PINGENKEY)
C          RETURNCODE  IFGT      4
C          MOVE      'CSNBKPI'    FAILMESSAGE
C          EXSR      SNDFAILMSG
C          SETON
C          ENDIF
C*****
C* Build the clear key part by placing the decimal number in
C* the appropriate half of the clear key field.
C*****
C          Z-ADD      2245734      LEFTHALFA
C          Z-ADD      7820595      LEFTHALFB
C          MOVE      LEFTHALF      CLEARKEY
C          MOVE      LEFTHALF      CLEARKEY
C*****

```

LR

```

C* Call Key Part Import the second time for the PINGEN key
C*****
C          MOVEL      'LAST      '      RULEARRAY
C          CALLP      CSNBKPI          (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     CLEARKEY:
C                                     PINGENKEY)
C          RETURNCODE  IFGT          4
C          MOVEL      'CSNBKPI'      FAILMESSAGE
C          EXSR       SNDFAILMSG
C          SETON
C          ENDIF
C*****
C* Import the PINVER key *
C*****
C          MOVEL      'FIRST      '      RULEARRAY
C          Z-ADD      1                RULEARRAYCNT
C          Z-ADD      16843009         LEFTHALFA
C          Z-ADD      16843009         LEFTHALFB
C          MOVEL      LEFTHALF         CLEARKEY
C          MOVE       LEFTHALF         CLEARKEY
C*****
C* Call Key Part Import the first time for the PINVER key
C*****
C          CALLP      CSNBKPI          (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     CLEARKEY:
C                                     PINVERKEY)
C          RETURNCODE  IFGT          4
C          MOVEL      'CSNBKPI'      FAILMESSAGE
C          EXSR       SNDFAILMSG
C          SETON
C          ENDIF
C*****
C* Build the clear key part by placing the decimal number in
C* the appropriate half of the clear key field.
C*****
C          Z-ADD      2245734          LEFTHALFA
C          Z-ADD      7820595          LEFTHALFB
C          MOVEL      LEFTHALF         CLEARKEY
C          MOVE       LEFTHALF         CLEARKEY
C*****
C* Call Key Part Import the second time for the PINVER key
C*****
C          MOVEL      'LAST      '      RULEARRAY
C          CALLP      CSNBKPI          (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     CLEARKEY:
C                                     PINVERKEY)
C          RETURNCODE  IFGT          4
C          MOVEL      'CSNBKPI'      FAILMESSAGE
C          EXSR       SNDFAILMSG
C          SETON
C          ENDIF
C*****

```

LR

LR

LR


```

C          SETON
C          ENDIF
C*****
C* Import the OPINENC key *
C*****
C          MOVEL 'FIRST '    RULEARRAY
C          Z-ADD 1          RULEARRAYCNT
C*****
C* Build the clear key part by placing the decimal number in
C* the appropriate half of the clear key field.
C*****
C          Z-ADD 16843009   LEFTHALFA
C          Z-ADD 16843009   LEFTHALFB
C          MOVEL LEFTHALF   CLEARKEY
C          MOVE  LEFTHALF   CLEARKEY
C*****
C* Call Key Part Import the first time for the OPINENC key
C*****
C          CALLP  CSNBKPI    (RETURNCODE:
C                          REASONCODE:
C                          EXITDATALEN:
C                          EXITDATA:
C                          RULEARRAYCNT:
C                          RULEARRAY:
C                          CLEARKEY:
C                          OPINKEY)
C          RETURNCODE  IFGT 4
C          MOVEL 'CSNBKPI' FAILMESSAGE
C          EXSR  SNDFAILMSG
C          SETON
C          ENDIF
C*****
C* Build the clear key part by placing the decimal number in
C* the appropriate half of the clear key field.
C*****
C          Z-ADD 2241518   LEFTHALFA
C          Z-ADD 197897    LEFTHALFB
C          MOVEL LEFTHALF   CLEARKEY
C          MOVE  LEFTHALF   CLEARKEY
C*****
C* Call Key Part Import the second time for the OPINENC key
C*****
C          MOVEL 'LAST '    RULEARRAY
C          CALLP  CSNBKPI    (RETURNCODE:
C                          REASONCODE:
C                          EXITDATALEN:
C                          EXITDATA:
C                          RULEARRAYCNT:
C                          RULEARRAY:
C                          CLEARKEY:
C                          OPINKEY)
C          RETURNCODE  IFGT 4
C          MOVEL 'CSNBKPI' FAILMESSAGE
C          EXSR  SNDFAILMSG
C          SETON
C          ENDIF
C*
C*****
C* Generate a Clear PIN with CSNBPGN (Clear_PIN_Generate)
C* Rule_array_count = 1
C* Rule_array = "IBM-PIN " (Same as Crypto Support/400)
C* PIN Length = 8
C* PIN Check length = 8 (But is ignored for IBM-PIN)
C* Data array:
C* Dec. table set to 0123456789123456
C* validation dta = 1111222233334444
C* clear PIN = ignored

```

```

C*****
C          Z-ADD      1          RULEARRAYCNT
C          MOVE      'IBM-PIN '  RULEARRAY
C          Z-ADD      8          PINLEN
C          Z-ADD      8          PINCKL
C          MOVE      '01234567'  DECTABLE
C          MOVE      '89123456'  DECTABLE
C          MOVE      '11112222'  VALDATA
C          MOVE      '33334444'  VALDATA
C*****
C* Call Clear PIN Generate
C*****
C          CALLP      CSNBPGN      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     PINGENKEY:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     PINLEN:
C                                     PINCKL:
C                                     DATAARRAY:
C                                     CPIN)
C          RETURNCODE  IFGT      4
C          MOVE      'CSNBPGN'  FAILMESSAGE
C          EXSR      SNDFAILMSG
C          SETON
C          ENDIF
C*
C*
C*****
C* Encrypt the clear PIN using CSNBCPE (Clear_PIN_Encrypt)
C* Rule_array_count = 1
C* Rule_array = "ENCRYPT "
C* PIN Profile = "3624  NONE          F"
C* PAN data is ignored
C* Sequence number is ignored but set to 99999 anyway
C*****
C          Z-ADD      1          RULEARRAYCNT
C          MOVE      'ENCRYPT '  RULEARRAY
C          MOVE      '3624  '  PINFORMAT
C          MOVE      'NONE  '  FORMATCONTROL
C          MOVE      '          F' PADDIGIT
C          Z-ADD      99999      SEQNUMBER
C*****
C* Call Clear PIN Encrypt
C*****
C          CALLP      CSNBCPE      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     OPINKEY:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     CPIN:
C                                     PROFILE:
C                                     PAN:
C                                     SEQNUMBER:
C                                     EPIN)
C          RETURNCODE  IFGT      4
C          MOVE      'CSNBCPE'  FAILMESSAGE
C          EXSR      SNDFAILMSG
C          SETON
C          ENDIF
C*
C*
C*****

```

LR

LR

```

C* Verify encrypted PIN using CSNBPVR (Encrypted_PIN_Verify)
C*****
C          MOVEL      'IBM-PIN '    RULEARRAY
C
C          CALLP      CSNBPVR      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     IPINKEY:
C                                     PINVERKEY:
C                                     PROFILE:
C                                     PAN:
C                                     EPIN:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     PINCKL:
C                                     DATAARRAY)
C
C          RETURNCODE  IFGT      4
C          MOVEL      'CSNBPVR'    FAILMESSAGE
C          EXSR      SNDFAILMSG
C          SETON
C          ENDIF
C
C*
C*****
C* Send successful completion message
C*****
C          CALL      'QMHSNDPM'
C          PARM
C          PARM      MESSAGEID
C          PARM      MESSAGEFILE
C          PARM      GOODMESSAGE
C          PARM      GOODMSGLENGTH
C          PARM      MSGTYPE
C          PARM      STACKENTRY
C          PARM      STACKCOUNTER
C          PARM      MSGKEY
C          PARM      ERRCODE
C*
C          SETON
C
C*
C*****
C* Subroutine to send a failure message
C*****
C          SNDFAILMSG  BEGSR
C          MOVE      FAILMESSAGE  FAILMSGTEXT
C          MOVE      RETURNCODE   FAILRETC
C          MOVE      REASONCODE   FAILRSNC
C          CALL      'QMHSNDPM'
C          PARM
C          PARM      MESSAGEID
C          PARM      MESSAGEFILE
C          PARM      FAILMSG
C          PARM      FAILMSGLENGTH
C          PARM      MSGTYPE
C          PARM      STACKENTRY
C          PARM      STACKCOUNTER
C          PARM      MSGKEY
C          PARM      ERRCODE
C          ENDSR

```

Generate and verify a digital signature

Generating a digital signature

You can protect data from undetected changes by including a proof of identity value called a digital signature. A digital signature relies on hashing and public key cryptography. When you sign data, you hash the data and encrypt the results with your private key. The encrypted hash value is called a digital signature.

If you change the original data, a different digital signature will be generated.

To use a PKA key to sign a file, write a program or change this program "Example: Signing a file with your Cryptographic Coprocessor."

Verifying a digital signature

Verifying a digital signature is the opposite of signing data. Verifying a signature will tell you if the signed data has changed or not. When a digital signature is verified, the signature is decrypted using the public key to produce the original hash value. The data that was signed is hashed. If the two hash values match, then the signature has been verified. To do this, write a program or change this "Example: Verifying a digital signature with your Cryptographic Coprocessor" on page 171.

Note: If you choose to use the program examples provided, change them to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

Example: Signing a file with your Cryptographic Coprocessor: Change this program example to suit your needs for signing a file with your Cryptographic Coprocessor.

```
/*-----*/
/* Description: Digitally signs a streams file.          */
/*                                                    */
/* COPYRIGHT      5769-SS1 (c) IBM Corp 1999          */
/*                                                    */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot    */
/* guarantee or imply reliability, serviceability, or function */
/* of these programs. All programs contained herein are    */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF    */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files.                             */
/*                                                    */
/* Parameters: File to be signed                        */
/*              File to contain signature              */
/*              Key label of key to use                */
/*                                                    */
/* Examples:                                           */
/* CALL PGM(SIGNFILE) PARM('file_to_sign' 'file_to_hold_sign' */
/*                          'key_label');                */
/*                                                    */
/* Note: The CCA verbs used in the this program are more fully */
/* described in the IBM CCA Basic Services Reference */
/* and Guide (SC31-8609) publication.                  */
/*                                                    */
/* Note: This program assumes the card you want to use is */
/* already identified either by defaulting to the CRP01 */
/* device or has been explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description.                    */
/*                                                    */
/* Use the following commands to compile this program:  */
/* ADDLIB LIB(QCCA)                                     */
/* CRTCMOD MODULE(SIGNFILE) SRCFILE(SAMPLE) SYSIFCOPT(*IFSIO) */
/* CRTPGM PGM(SIGNFILE) MODULE(SIGNFILE)              */
/*          BNDSRVPGM(QCCA/CSNDDSG QCCA/CSNBOWH)       */
/*                                                    */
/* Note: authority to the CSNDDSG and CSNBOWH service programs */
/* in the QCCA library is assumed.                    */
/*                                                    */
/* Common Cryptographic Architecture (CCA) verbs used: */
```

```

/*    Digital_Signature_Generate (CSNDDSG)                */
/*    One_Way_Hash (CSNBOWH)                            */
/*-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "csucincl.h"      /* header file for CCA Cryptographic
                          Service Provider for iSeries    */

/*-----*/
/* standard return codes                                */
/*-----*/
#define ERROR -1
#define OK    0

int hash_file(long h_len, char h_out[128], FILE *t_in);

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters                          */
    /*-----*/
    long return_code;
    long reason_code;
    long exit_data_length = 0L;
    char exit_data[2];
    long rule_array_count = 0L;
    char rule_array[1][8];

    /*-----*/
    /* parameters unique to this sample program        */
    /*-----*/
    long PKA_private_key_identifier_length = 64;
    char PKA_private_key_identifier[64];
    long hash_length = 16L;
    char hash[128];
    long signature_field_length = 128L;
    long signature_bit_length = 0L;
    char signature_field[256];
    char key_label[64];
    long key_token_length = 2500L;
    char key_token[2500];

    FILE *file2sign;
    FILE *signature;
    int hash_return;

    if (argc < 2)
    {
        printf("Name of file to be signed is missing.");
        return ERROR;
    }
    else if (argc < 3)
    {
        printf("Name of file where the signature should ");
        printf("be written is missing.");
        return ERROR;
    }
    else if (argc < 4)
    {
        printf("Key label for the key to be used for signing is missing.");
        return ERROR;
    }

    if ( ( strlen(argv[3])) > 64 )

```

```

    {
printf("Invalid Key Label. Key label longer than 64.");
return ERROR;
    }
    else
    {
memset(PKA_private_key_identifier, ' ', 64);
memcpy(PKA_private_key_identifier, argv[3],strlen(argv[3]));
    }

    /* Open the file that is being signed. */
    if ( (file2sign = fopen(argv[1],"rb")) == NULL)
    {
printf("Opening of file %s failed.",argv[1]);
return ERROR;
    }

    /* Obtain a hash value for the file. */
    hash_return = hash_file(hash_length, hash, file2sign);

    /* Close the file. */
    fclose(file2sign);

    if (hash_return != OK)
    {
printf("Signature generation failed due to hash error.\n");
    }

    else
    {
/* Use CSNDDSG to generate the signature. */
CSNDDSG(&return_code,
&reason_code,
&exit_data_length,
exit_data,
&rule_array_count,
(char *) rule_array,
&PKA_private_key_identifier_length,
PKA_private_key_identifier,
&hash_length,
hash,
&signature_field_length,
&signature_bit_length,
signature_field);
    }

    if (return_code != 0)
    {
printf("Signature generation failed with return/reason code %ld/%ld",
return_code, reason_code);
return ERROR;
    }
    else
    {
printf("Signature generation was successful.");
printf("Return/Reason codes = %ld/%ld\n", return_code, reason_code);
printf("Signature has length = %ld\n",signature_field_length);

signature = fopen(argv[2],"wb");
if (signature == NULL)
    {
printf("Open of file %s failed.",argv[2]);
printf("Signature was not saved.");
return ERROR;
    }

```

```

    fwrite(signature_field, 1, signature_field_length, signature);
    fclose(signature);
    printf("Signature was saved successfully in %s.", argv[2]);
    return OK;
}
}

```

```

int hash_file(long h_len, char h_out[128], FILE *t_in)
{
    /*-----*/
    /* standard CCA parameters */
    /*-----*/
    long return_code;
    long reason_code;
    long exit_data_length = 0;
    char exit_data[2];
    long rule_array_count = 2;
    char rule_array[2][8];

    /*-----*/
    /* parameters unique to this function */
    /*-----*/
    long text_length;
    char text[1024];
    long chaining_vector_length = 128;
    char chaining_vector[128];

    long file_length;

    fseek(t_in, 0, SEEK_END);
    file_length = ftell(t_in);
    rewind(t_in);

    text_length = fread(text, 1, 1024, t_in);

    memcpy(rule_array[0], "MD5      ", 8);

    if (file_length <= 1024) {
        memcpy(rule_array[1], "ONLY   ", 8);
    }
    else {
        memcpy(rule_array[1], "FIRST  ", 8);
    }

    while (file_length > 0)
    {
        CSNBOWH(&return_code,
                &reason_code,
                &exit_data_length,
                exit_data,
                &rule_array_count,
                (char *) rule_array,
                &text_length,
                text,
                &chaining_vector_length,
                chaining_vector,
                &h_len,
                h_out);

        if (return_code != 0)
            break;

        printf("Hash iteration worked.\n");

        file_length -= text_length;
    }
}

```



```

if (file_length > 0)
{
    text_length = fread(text, 1, 1024, t_in);

    if (file_length <= 1024) {
        memcpy(rule_array[1], "LAST    ", 8);
    }
    else {
        memcpy(rule_array[1], "MIDDLE ", 8);
    }
}

if (return_code != 0)
{
    printf("Hash function failed with return/reason code %ld/%ld\n",
        return_code, reason_code);
    return ERROR;
}
else
{
    printf("Hash completed successfully.\n");
    printf("hash length = %ld\n", h_len);
    printf("hash = %.32s\n", h_out);
    return OK;
}
}

```

Example: Verifying a digital signature with your Cryptographic Coprocessor: Change this program example to suit your needs for verifying a digital signature with your Cryptographic Coprocessor.

```

/*-----*/
/* Description: Verifies the digital signature of an IFS file */
/*              produced by the SIGNFILE sample program.    */
/*              */
/* COPYRIGHT    5769-SS1 (c) IBM Corp 1999                */
/*              */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot    */
/* guarantee or imply reliability, serviceability, or function */
/* of these programs. All programs contained herein are    */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF     */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files.                               */
/*              */
/* Parameters: Signed file                                 */
/*              File containing the signature              */
/*              Key label of the key to use               */
/*              */
/* Examples:                                              */
/* CALL PGM(VERFILESIG) PARM('name_of_signed_file' +     */
/*                            'name_of_file_w_signature' + */
/*                            'key_label');                */
/*              */
/* Note: The CCA verbs used in the this program are more fully */
/* described in the IBM CCA Basic Services Reference */
/* and Guide (SC31-8609) publication.                    */
/*              */
/* Note: This program assumes the card you want to use is */
/* already identified either by defaulting to the CRP01 */
/* device or has been explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description.                       */
/*              */

```

```

/* Use the following commands to compile this program:          */
/* ADDLIB LIB(QCCA)                                           */
/* CRTCMOD MODULE(VERFILESIG) SRCFILE(SAMPLE) SYSIFCOPT(*IFSIO)*/
/* CRTPGM PGM(SIGNFILE) MODULE(SIGNFILE) +                  */
/* BNSRVPGM(QCCA/CSNDDSV QCCA/CSNBOWH)                       */
/*                                                             */
/* Note: authority to the CSNDDSV and CSNBOWH service programs */
/*       in the QCCA library is assumed.                      */
/*                                                             */
/* Common Cryptographic Architecture (CCA) verbs used:       */
/*   Digital_Signature_Verify (CSNDDSV)                       */
/*   One_Way_Hash (CSNBOWH)                                   */
/*-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "csucincl.h" /* header file for CCA Cryptographic
                    Service Provider for iSeries */

/*-----*/
/* standard return codes                                     */
/*-----*/
#define ERROR -1
#define OK 0

int hash_file(long h_len, char h_out[128], FILE *t_in);

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters                               */
    /*-----*/

    long return_code;
    long reason_code;
    long exit_data_length = 0L;
    char exit_data[2];
    long rule_array_count = 0L;
    char rule_array[1][8];

    /*-----*/
    /* parameters unique to this sample program             */
    /*-----*/
    long PKA_public_key_identifier_length = 64;
    char PKA_public_key_identifier[64];
    long hash_length = 16L;
    char hash[128];
    long signature_field_length;
    char signature_field[256];
    char key_label[64];

    FILE *file2verify;
    FILE *signature;
    int hash_return;

    if (argc < 2)
    {
        printf("Name of file to be verified is missing.\n");
        return ERROR;
    }
    else if (argc < 3)
    {
        printf("Name of file containing the signature is missing.\n");
        return ERROR;
    }
    else if (argc < 4)

```

```

    {
printf("Key label for the key to be used for verification is missing.\n");
return ERROR;
    }

    if (strlen(argv[3]) > 64 )
    {
printf("Invalid Key Label. Key label longer than 64 bytes.");
return ERROR;
    }
    else
    {
memset(PKA_public_key_identifier, ' ', 64);
memcpy(PKA_public_key_identifier, argv[3], strlen(argv[3]));
    }

    /* Open the file that is being verified. */
    if ( (file2verify = fopen(argv[1],"rb")) == NULL)
    {
printf("Opening of file %s failed.",argv[1]);
return ERROR;
    }

    /* Obtain a hash value for the file. */
    hash_return = hash_file(hash_length, hash, file2verify);

    /* Close the file. */
    fclose(file2verify);

    if (hash_return != OK)
    {
printf("Signature verification failed due to hash error.\n");
return ERROR;
    }
    else
    {
signature = fopen(argv[2],"rb");
if (signature == NULL)
    {
printf("Open of signature file %s failed.",argv[2]);
printf("Signature was not verified.");
return ERROR;
    }
}

memset(signature_field, ' ', 256);

fseek(signature, 0, SEEK_END);
signature_field_length = ftell(signature);
rewind(signature);

fread(signature_field, 1, signature_field_length, signature);
fclose(signature);

/* Use CSNDDSV to verify the signature. */
CSNDDSV(&return_code,
&reason_code,
&exit_data_length,
exit_data,
&rule_array_count,
(char *) rule_array,
&PKA_public_key_identifier_length,
PKA_public_key_identifier,
&hash_length,
hash,
&signature_field_length,
signature_field);

```

```

    }

    if (return_code != 0)
    {
printf("Signature verification failed with return/reason code %ld/%ld",
return_code, reason_code);
return ERROR;
    }
    else
    {
printf("Signature verification was successful.");
printf("Return/Reason codes = %ld/%ld\n", return_code, reason_code);

    }
}

```

```

int hash_file(long h_len, char h_out[128], FILE *t_in)
{
/*-----*/
/* standard CCA parameters */
/*-----*/

long return_code;
long reason_code;
long exit_data_length = 0;
char exit_data[2];
long rule_array_count = 2;
char rule_array[2][8];

/*-----*/
/* parameters unique to this function */
/*-----*/
long text_length;
char text[1024];
long chaining_vector_length = 128;
char chaining_vector[128];

long file_length;

fseek(t_in, 0, SEEK_END);
file_length = ftell(t_in);
rewind(t_in);

text_length = fread(text, 1, 1024, t_in);

memcpy(rule_array[0], "MD5      ", 8);

if (file_length <= 1024) {
memcpy(rule_array[1], "ONLY   ", 8);
}
else {
memcpy(rule_array[1], "FIRST  ", 8);
}

while (file_length > 0)
{
CSNBOWH(&return_code,
&reason_code,
&exit_data_length,
exit_data,
&rule_array_count,
(char *) rule_array,
&text_length,
text,

```

```

&chaining_vector_length,
chaining_vector,
&h_len,
h_out);

if (return_code != 0)
    break;

printf("Hash iteration worked.\n");

file_length -= text_length;

if (file_length > 0)
{
    text_length = fread(text, 1, 1024, t_in);

    if (file_length <= 1024) {
        memcpy(rule_array[1], "LAST    ", 8);
    }
    else {
        memcpy(rule_array[1], "MIDDLE ", 8);
    }
}

if (return_code != 0)
{
    printf("Hash function failed with return/reason code %ld/%ld\n",
        return_code, reason_code);
    return ERROR;
}
else
{
    printf("Hash completed successfully.\n");
    printf("hash length = %ld\n", h_len);
    printf("hash = %.32s\n", h_out);
    return OK;
}
}

```

Manage multiple Cryptographic Coprocessors

You can have up to eight Cryptographic Coprocessors per partition. Note that the maximum number of Cryptographic Coprocessors supported per server is dependent the iSeries model. Spreading the work across multiple Cryptographic Coprocessors and multiple jobs gives you better performance provided that they are all configured the same. Only one Coprocessor (cryptographic device description) may be allocated to a job at one time. However, the job can switch between Coprocessors by deallocating the current Coprocessor and allocating a new one. For the i5/OS or OS/400 SSL user, the allocation and deallocation of the Coprocessors is managed by the system if the SSL configuration in DCM indicates that more than one Coprocessor is to be used for SSL session establishment.

If you configure all of the Coprocessors the same, then all operational keys will work identically on all of the Coprocessors. Any data encrypted on one Coprocessor can be decrypted on a different Coprocessor. All key store files will work interchangeably with any of the Coprocessors. The most important part of configuring the Coprocessors identically is the master keys. If you entered the master key in parts for one Coprocessor, you must enter the same master key parts for all of the other Coprocessors if you want them to work interchangeably. If a random master key was generated inside of the Coprocessor, then you must clone the master key to the other Coprocessors if you want all of the Coprocessors to work interchangeably.

There may be certain situations where you do not want all of the Coprocessors to be configured the same. They could all have different configurations or they could be set up in groups where the configuration within a group is the same but between groups is different. For these cases, all operational

keys may not work identically on all of the Coprocessors. Data encrypted on one Coprocessor may not be able to be recovered on a different Coprocessor. Also, the keystore files may not work interchangeably among Coprocessors. For these situations, you must keep track of which keystore files and operational keys will work for a given Coprocessor. While configuring the Coprocessors differently may limit the scalability of cryptographic applications, it can provide more granularity in terms of security. For example, you can grant different object authorities to different cryptographic device descriptions.

If you use retained PKA keys then the Coprocessors are also not interchangeable. Retained keys can not be exported in any manner outside of the Coprocessor. Therefore, any cryptographic request that uses that retained key must be sent to the Coprocessor that stores the retained key.

The following material is only applicable if you are using i5/OS or OS/400 applications:

Allocating a device

The `Cryptographic_Resource_Allocate` (CSUACRA) API verb is used to explicitly allocate a cryptographic device to your job so that the system can determine how to route all subsequent cryptographic requests. If you use any of the CCA API verbs without first explicitly using the `Cryptographic_Resource_Allocate` (CSUACRA) API verb, the system will attempt to allocate the default cryptographic device. The default device is the cryptographic device named CRP01. It must be created by either using the Basic Configuration wizard or the Create Device Crypto (CRTDEVCRP) CL command. You only need to use CSUACRA when you wish to use a device other than the default cryptographic device. A device allocated to a job, either explicitly or implicitly, remains allocated until either the job ends or the device is deallocated using the `Cryptographic_Resource_Deallocate` (CSUACRD) API verb. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both programs perform the same function.

- “Example: ILE C program for allocating a Coprocessor”
- “Example: ILE RPG program for allocating a Coprocessor” on page 178

Deallocating a device

When you have finished using a Cryptographic Coprocessor, you should deallocate the Cryptographic Coprocessor by using the `Cryptographic_Resource_Deallocate` (CSUACRD) API verb. A cryptographic device description can not be varied off until all jobs using the device have deallocated it. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both programs perform the same function.

- “Example: ILE C program for deallocating a Coprocessor” on page 181
- “Example: ILE RPG program for deallocating a Coprocessor” on page 182

Example: ILE C program for allocating a Coprocessor: Change this program example to suit your needs for allocating a Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```
/*-----*/
/* Allocate a crypto device to the job.          */
/*                                              */
/*                                              */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000 */
/*                                              */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
```

```

/* these programs and files. */
/* */
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: */
/* none. */
/* */
/* Example: */
/* CALL PGM(CRPALLOC) (CRP02) */
/* */
/* The Common Cryptographic Architecture (CCA) verb used is */
/* Cryptographic_Resource_Allocate (CSUACRA). */
/* */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(CRPALLOC) SRCFILE(SAMPLE) */
/* CRTPGM PGM(CRPALLOC) MODULE(CRPALLOC) */
/* BNDSRVPGM(QCCA/CSUACRA) */
/* */
/* Note: Authority to the CSUACRA service program in the */
/* QCCA library is assumed. */
/* */
/*-----*/
#include <string.h>
#include <stdio.h>
#include "csucincl.h"

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR -1
#define OK 0
#define WARNING 4

int main(int argc, char *argv[])
{
/*-----*/
/* standard CCA parameters */
/*-----*/
long return_code = 0;
long reason_code = 0;
long exit_data_length = 2;
char exit_data[4];
char rule_array[2][8];
long rule_array_count = 2;
long resource_name_length;

/*-----*/
/* Process the parameters */
/*-----*/
if (argc < 1)
{
printf("Device parameter must be specified.\n");
return(ERROR);
}

/*-----*/
/* Set the keyword in the rule array */
/*-----*/
memcpy(rule_array,"DEVICE ",8);
rule_array_count = 1;

```

```

/*-----*/
/* Set the resource name length */
/*-----*/
resource_name_length = strlen(argv[1]);

/*-----*/
/* Call Cryptographic Resource Allocate SAPI */
/*-----*/
CSUACRA( &return_code, &reason_code, &exit_data_length,
        (char *)exit_data,
        (long *) &rule_array_count,
        (char *) rule_array,
        (long *) &resource_name_length,
        (char *) argv[1]); /* resource name */

/*-----*/
/* Check the return code and display the results */
/*-----*/
if ( (return_code == OK) | (return_code == WARNING) )
{
    printf("Request was successful\n");
    return(OK);
}
else
{
    printf("Request failed with return/reason codes: %d/%d \n",
        return_code, reason_code);
    return(ERROR);
}
}

```

Example: ILE RPG program for allocating a Coprocessor: Change this program example to suit your needs for allocating a Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* CRPALLOC
D*
D* Sample program that allocates a crypto device to the job.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters:
D* Device Name
D*
D* Example:
D* CALL PGM(CRPALLOC) PARM(CRP02)
D*
D* Use these commands to compile this program on iSeries:

```



```

D* CRTRPGMOD MODULE(CRPALLOC) SRCFILE(SAMPLE)
D* CRTPGM PGM(CRPALLOC) MODULE(CRPALLOC)
D*      BNDSRVPGM(QCCA/CSUACRA)
D*
D* Note: Authority to the CSUACRA service program in the
D*      QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Resource_Allocate (CSUACRA)
D*
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*      ** Return code
DRETURNCODE      S          9B 0
D*      ** Reason code
DREASONCODE      S          9B 0
D*      ** Exit data length
DEXITDATALEN     S          9B 0
D*      ** Exit data
DEXITDATA        S          4
D*      ** Rule array count
DRULEARRAYCNT   S          9B 0
D*      ** Rule array
DRULEARRAY       S          16
D*      ** Resource name length
DRESOURCEAMLEN  S          9B 0
D*      ** Resource name
DRESOURCENAME   S          10
D*
D*****
D* Prototype for Cryptographic_Resource_Allocate (CSUACRA)
D*****
DCSUACRA      PR
DRETCODE      9B 0
DRSNCODE      9B 0
DEXTDTALEN    9B 0
DEXTDTA       4
DRARRAYCT     9B 0
DRARRAY       16
DRSCNAMLEN    9B 0
DRSCNAM       10
D*
D*-----
D*      ** Declares for sending messages to the
D*      ** job log using the QMHSDPM API
D*-----
DMSG          S          75  DIM(2) CTDATA PERRCD(1)
DMSGLENGTH    S          9B 0 INZ(75)
D             DS
DMSGTEXT      1          75
DFAILRET      41         44
DFAILRSNC     46         49
DMESSAGEID    S          7  INZ(' ')
DMESSAGEFILE  S          21 INZ(' ')
DMSGKEY       S          4  INZ(' ')
DMSGTYPE      S          10 INZ('*INFO ')
DSTACKENTRY   S          10 INZ('* ')
DSTACKCOUNTER S          9B 0 INZ(2)
DERRCODE      DS
DBYTESIN      1          4B 0 INZ(0)
DBYTESOUT     5          8B 0 INZ(0)
D*
C*****
C* START OF PROGRAM *
C* *
C*-----*

```

```

C      *ENTRY      PLIST
C      PARM              RESOURCENAME      10
C*-----*
C* Set the keyword in the rule array *
C*-----*
C              MOVEL      'DEVICE '      RULEARRAY
C              Z-ADD      1              RULEARRAYCNT
C*-----*
C* Set the resource name length *
C*-----*
C              Z-ADD      10              RESOURCENAMLEN
C*-----*
C* Call Cryptographic Resource Allocate SAPI *
C*-----*
C              CALLP      CSUACRA      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     RESOURCENAMLEN:
C                                     RESOURCENAME)
C*-----*
C* Check the return code *
C*-----*
C      RETURNCODE      IFGT      4
C*-----*
C*      * Send error message *
C*-----*
C              MOVE      MSG(1)      MSGTEXT
C              MOVE      RETURNCODE      FAILRETC
C              MOVE      REASONCODE      FAILRSNC
C              EXSR      SNDMSG
C*
C              ELSE
C*
C*      * Send success message *
C*-----*
C              MOVE      MSG(2)      MSGTEXT
C              EXSR      SNDMSG
C*
C              ENDIF
C*
C              SETON
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C              CALL      'QMHSNDPM'
C              PARM      MESSAGEID
C              PARM      MESSAGEFILE
C              PARM      MSGTEXT
C              PARM      MSGLENGTH
C              PARM      MSGTYPE
C              PARM      STACKENTRY
C              PARM      STACKCOUNTER
C              PARM      MSGKEY
C              PARM      ERRCODE
C              ENDSR

```

LR

```

C*
**
CSUACRA failed with return/reason codes 9999/9999'
The request completed successfully

```

Example: ILE C program for deallocating a Coprocessor: Change this program example to suit your needs for deallocating a Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Deallocate a crypto device from a job. */
/* */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: */
/* none. */
/* */
/* Example: */
/* CALL PGM(CRPDEALLOC) (CRP02) */
/* */
/* */
/* The Common Cryptographic Architecture (CCA) verb used is */
/* Cryptographic_Resource_Deallocate (CSUACRD). */
/* */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(CRPALLOC) SRCFILE(SAMPLE) */
/* CRTPGM PGM(CRPALLOC) MODULE(CRPALLOC) */
/* BNDSRVPGM(QCCA/CSUACRD) */
/* */
/* Note: Authority to the CSUACRD service program in the */
/* QCCA library is assumed. */
/* */
/*-----*/
#include <string.h>
#include <stdio.h>
#include "csucincl.h"

/*-----*/
/* standard return codes */
/*-----*/

#define ERROR -1
#define OK 0
#define WARNING 4

int main(int argc, char *argv[])
{
/*-----*/

```

```

/* standard CCA parameters */
/*-----*/
long return_code = 0;
long reason_code = 0;
long exit_data_length = 2;
char exit_data[4];
char rule_array[2][8];
long rule_array_count = 2;
long resource_name_length;

/*-----*/
/* Process the parameters */
/*-----*/
if (argc < 1)
{
    printf("Device parameter must be specified.\n");
    return(ERROR);
}

/*-----*/
/* Set the keyword in the rule array */
/*-----*/
memcpy(rule_array,"DEVICE ",8);
rule_array_count = 1;

/*-----*/
/* Set the resource name length */
/*-----*/
resource_name_length = strlen(argv[1]);

/*-----*/
/* Call Cryptographic Resource Deallocate SAPI */
/*-----*/
CSUACRD( &return_code, &reason_code, &exit_data_length,
        (char *)exit_data,
        (long *) &rule_array_count,
        (char *) rule_array,
        (long *) &resource_name_length,
        (char *) argv[1]); /* resource name */

/*-----*/
/* Check the return code and display the results */
/*-----*/
if ( (return_code == OK) | (return_code == WARNING) )
{
    printf("Request was successful\n");
    return(OK);
}
else
{
    printf("Request failed with return/reason codes: %d/%d \n",
        return_code, reason_code);
    return(ERROR);
}
}

```

Example: ILE RPG program for deallocating a Coprocessor: Change this program example to suit your needs for deallocating a Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* CRPDEALLOC
D*
D* Sample program that deallocates a crypto device to the job.
D*
D*

```

```

D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters:
D* Device name
D*
D* Example:
D* CALL PGM(CRPDEALLOC) PARM(CRP02)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(CRPDEALLOC) SRCFILE(SAMPLE)
D* CRTPGM PGM(CRPDEALLOC) MODULE(CRPDEALLOC)
D* BNDSRVPGM(QCCA/CSUACRD)
D*
D* Note: Authority to the CSUACRD service program in the
D* QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Resource_Deallocate (CSUACRD)
D*
D*
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S          9B 0
D*          ** Exit data
DEXITDATA S           4
D*          ** Rule array count
DRULEARRAYCNT S         9B 0
D*          ** Rule array
DRULEARRAY S           16
D*          ** Resource name length
DRESOURCENAMLEN S        9B 0
D*          ** Resource name
DRESOURCENAME S         10
D*
D*****
D* Prototype for Cryptographic_Resource_Deallocate (CSUACRD)
D*****
DCSUACRD PR
DRETCODE          9B 0
DRSNCODE          9B 0
DEXTDTALEN        9B 0
DEXTDTA           4
DRARRAYCT         9B 0
DRARRAY           16
DRSCNAMLEN        9B 0
DRSCNAM           10

```

```

D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG          S          75  DIM(2) CTDATA PERRCD(1)
DMSGLENGTH   S          9B 0 INZ(75)
D            DS
DMSGTEXT     1          75
DFAILRETC    41         44
DFAILRSNC    46         49
DMESSAGEID   S          7   INZ('      ')
DMESSAGEFILE S          21  INZ('      ')
DMSGKEY       S          4   INZ('      ')
DMSGTYPE     S          10  INZ('*INFO ')
DSTACKENTRY  S          10  INZ('*      ')
DSTACKCOUNTER S         9B 0 INZ(2)
DERRCODE     DS
DBYTESIN     1          4B 0 INZ(0)
DBYTESOUT    5          8B 0 INZ(0)
D*
C*****
C* START OF PROGRAM *
C* *
C*-----*
C  *ENTRY      PLIST
C              PARM          RESOURCENAME
C*-----*
C* Set the keyword in the rule array *
C*-----*
C              MOVE      'DEVICE '  RULEARRAY
C              Z-ADD     1          RULEARRAYCNT
C*
C*-----*
C* Set the resource name length *
C*-----*
C              Z-ADD     10          RESOURCENAMLEN
C*
C*-----*
C* Call Cryptographic Resource Deallocate SAPI *
C*-----*
C              CALLP     CSUACRD      (RETURNCODE:
C                                  REASONCODE:
C                                  EXITDATALEN:
C                                  EXITDATA:
C                                  RULEARRAYCNT:
C                                  RULEARRAY:
C                                  RESOURCENAMLEN:
C                                  RESOURCENAME)
C*-----*
C* Check the return code *
C*-----*
C  RETURNCODE  IFGT      4
C*
C*          *-----*
C*          * Send error message *
C*          *-----*
C              MOVE      MSG(1)      MSGTEXT
C              MOVE      RETURNCODE  FAILRETC
C              MOVE      REASONCODE  FAILRSNC
C              EXSR      SNDMSG
C*
C              ELSE
C*
C*          *-----*
C*          * Send success message *
C*          *-----*
C              MOVE      MSG(2)      MSGTEXT

```

```

C          EXSR      SNDMSG
C*
C          ENDIF
C*
C          SETON                      LR
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C                  CALL      'QMHSNDPM'
C                  PARM      MESSAGEID
C                  PARM      MESSAGEFILE
C                  PARM      MSGTEXT
C                  PARM      MSGLENGTH
C                  PARM      MSGTYPE
C                  PARM      STACKENTRY
C                  PARM      STACKCOUNTER
C                  PARM      MSGKEY
C                  PARM      ERRCODE
C                  ENDSR
C*

```

```

**
CSUACRD failed with return/reason codes 9999/9999'
The request completed successfully

```

Clone master keys

Master key cloning is a method for securely copying a master key from one Cryptographic Coprocessor to another without exposing the value of the master key. This is performed by a process of splitting the master key into n shares, where n is a number from 1 to 15. m shares are required to rebuild the master key in another Coprocessor, where m is a number from 1 to 15 and less than or equal to n .

The term "cloning" is used to differentiate the process from "copying" because no one share, or any combination of fewer than m shares, provide sufficient information needed to rebuild the master key.

The Coprocessor containing the master key to be cloned is referred to as either the master-key-share source node or the Sender. The Sender must generate a retained RSA key pair. This private key must also have been marked as suitable for use with cloning when it was generated. The key is known as either the Coprocessor Share Signing key or the Sender key. The Coprocessor that will receive the master key is referred to as either the master-key-share target node or the Receiver. The Receiver must also generate a retained RSA key pair and must also have been marked as suitable for use with cloning. This key is known as either the Coprocessor Share Receiving key or simply the Receiver key.

Both the Sender and Receiver public keys must be digitally signed or certified by a retained private key in a Coprocessor, referred to as the public key certifying node or the Certifier. This retained private key is the Certifier key. It is also referred to as the Share Administration key. The associated public key must be registered in both the Sender and the Receiver before shares can be generated and received. A Cryptographic Coprocessor can take on the role of Certifier only, or can it be both Certifier and Sender, or it can be both Certifier and Receiver.

As each share is generated it is signed by the Coprocessor using the Sender private key and encrypted by a newly generated triple DES key. The triple DES key is then wrapped or encrypted by the Receiver public key.

As each share is received, the signature on the share is verified using the Sender public key, the triple DES key is unwrapped or decrypted using the Receiver private key, and the share decrypted using the triple DES key. When m number of shares have been received, the cloned master key will be complete within the new master key register of the Receiver.

The easiest and fastest way to clone master keys is to use the Cryptographic Coprocessor configuration web-based utility. The utility includes the Master key cloning advisor. To start the master key cloning advisor, follow these steps:

1. Click on **Manage configuration** on the Cryptographic Coprocessor configuration page.
2. Click on **Master keys**.
3. Select a device.
4. Enter a valid Coprocessor profile and password.
5. Click on the **Clone** button.

If you would prefer to write your own application to clone master keys, you can do so by using the following API verbs:

- Cryptographic_Facility_Control (CSUACFC)
- PKA_Key_Token_Build (CSNDPKB) (may not be needed depending upon how you write your application)
- PKA_Key_Generate (CSNDPKG)
- PKA_Public_Key_Register (CSNDPKR)
- One_Way_Hash (CSNBOWH)
- Digital_Signature_Generate (CSNDDSG)
- Master_Key_Distribution (CSUAMKD)

Nine pairs of example programs are provided for your consideration. Each pair contains a program written in ILE C and a program written in ILE RPG. Both perform the same function.

- “Example: ILE C program for setting the min and max values for master key shares in your Cryptographic Coprocessor”
- “Example: ILE RPG program for setting the min and max values for master key shares in your Cryptographic Coprocessor” on page 189
- “Example: ILE C program for generating a retained key pair for cloning master keys” on page 191
- “Example: ILE RPG program for generating a retained key pair for cloning master keys” on page 196
- “Example: ILE C program for registering a public key hash” on page 203
- “Example: ILE RPG program for registering a public key hash” on page 206
- “Example: ILE C program for registering a public key certificate” on page 212
- “Example: ILE RPG program for registering a public key certificate” on page 215
- “Example: ILE C program for certifying a public key token” on page 219
- “Example: ILE RPG program for certifying a public key token” on page 224
- “Example: ILE C program for obtaining a master key share” on page 232
- “Example: ILE RPG program for obtaining a master key share” on page 235
- “Example: ILE C program for installing a master key share” on page 241
- “Example: ILE RPG program for installing a master key share” on page 245

The remaining two pairs of example programs are not necessary for master key cloning. They may be useful, however, for developing and testing the previous example programs.

- “Example: ILE C program for listing retained keys” on page 252
- “Example: ILE RPG program for listing retained keys” on page 254
- “Example: ILE C program for deleting retained keys” on page 257
- “Example: ILE RPG program for deleting retained keys” on page 259

Example: ILE C program for setting the min and max values for master key shares in your Cryptographic Coprocessor: Change this program example to suit your needs for setting the min and max values for master key shares in your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```
/*-----*/
/* Set the M-of-N values in the Coprocessor. These values are */
/* used in cloning of the master key. The master key is */
/* cryptographically split into N number of parts and M number of */
/* parts are needed to recover it. */
/*
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 2000
/*
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files.
/*
/*
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication.
/*
/* Parameters:
/* none.
/*
/* Example:
/* CALL PGM(SETMOFN) PARM(5 15)
/*
/* Note: This program assumes the device to use
/* already identified either by defaulting to the CRP01
/* device or by being explicitly named using the
/* Cryptographic_Resource_Allocate verb. Also this
/* device must be varied on and you must be authorized
/* to use this device description.
/*
/* Use these commands to compile this program on iSeries:
/* ADDLIB LIB(QCCA)
/* CRTCMOD MODULE(SETMOFN) SRCFILE(SAMPLE)
/* CRTPGM PGM(SETMOFN) MODULE(SETMOFN)
/* BNDSRVPGM(QCCA/CSUACFC)
/*
/* Note: Authority to the CSUACFC service program in the
/* QCCA library is assumed.
/*
/* The Common Cryptographic Architecture (CCA) verb used is
/* Cryptographic_Facilites_Control (CSUACFC).
/*
/*-----*/

#include "csucincl.h" /* header file for CCA Cryptographic */
/* Service Provider for iSeries */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "decimal.h"

/*-----*/
/* standard return codes
/*-----*/
#define ERROR -1
#define OK 0
#define WARNING 4
```

```

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters */
    /*-----*/
    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 2;
    char exit_data[4];
    char rule_array[2][8];
    long rule_array_count = 2;

    /*-----*/
    /* fields unique to this sample program */
    /*-----*/
    decimal(15,5) mparm, nparm;
    long verb_data[2];
    long verb_data_length = 8;

    /*-----*/
    /* Process parameters. Numeric parms from the command line are
    /* passed in decimal 15,5 format. The parms need to be converted
    /* to int format. */
    /*-----*/
    memcpy(&mparm,argv[1],sizeof(mparm));
    memcpy(&nparm,argv[2],sizeof(nparm));
    verb_data[0] = mparm;
    verb_data[1] = nparm;

    /*-----*/
    /* Set keywords in the rule array */
    /*-----*/
    memcpy(rule_array,"ADAPTER1SET-MOFN", 16);

    /*-----*/
    /* Invoke the verb to set the M of N values */
    /*-----*/
    CSUACFC( &return_code,
            &reason_code,
            &exit_data_length,
            exit_data,
            &rule_array_count,
            (char *)rule_array,
            &verb_data_length,
            (unsigned char *)verb_data);

    /*-----*/
    /* Check the results of the call */
    /*-----*/
    if ( (return_code == OK) | (return_code == WARNING) )
    {
        printf("M of N values were successfully set with ");
        printf("return/reason codes %ld/%ld\n\n",
              return_code, reason_code);
        return(OK);
    }
    else
    {
        printf("An error occurred while setting the M of N values.\n");
        printf("Return/reason codes %ld/%ld\n\n",
              return_code, reason_code);
        return(ERROR);
    }
}

```

Example: ILE RPG program for setting the min and max values for master key shares in your Cryptographic Coprocessor: Change this program example to suit your needs for setting the min and max values for master key shares in your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* SETMOFN
D*
D* Set the M-of-N values in the Cryptographic Coprocessor. These values
D* are used in cloning of the master key. The master key is
D* cryptographically split into N number of parts and M number of
D* parts are needed to recover it.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: M and N
D*
D* Example:
D* CALL PGM(SETMOFN) PARM(5 10)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(SETMOFN) SRCFILE(SAMPLE)
D* CRTPGM PGM(SETMOFN) MODULE(SETMOFN)
D* BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSUACFC service program in the
D* QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Facilty_Control (CSUACFC)
D*
D*****
D*-----
D* Declare variables used on CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S         9B 0
D*          ** Exit data
DEXITDATA S           4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY S           16
D*          ** Verb data length
DVERBDATALEN S         9B 0
D*          ** Verb data contain M (minimum) and N (maximum)

```

```

DVERBDATA      DS          8
DM              9B 0
DN              9B 0
D*
D*****
D* Prototype for Cryptographic_Facility_Control (CSUACFC)
D*****
DCSUACFC       PR
DRETCODE       9B 0
DRSNCODE       9B 0
DEXTDTALEN     9B 0
DEXTDTA        4
DRARRAYCT      9B 0
DRARRAY        16
DVRBDTALEN     9B 0
DVRBDTA        8
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG           S          75  DIM(2) CTDATA PERRCD(1)
DMSGLENGTH    S          9B 0 INZ(75)
D
DMSGTEXT       1          80
DFAILRET      41         44
DFAILRNC      46         49
DMESSAGEID     S          7  INZ(' ')
DMESSAGEFILE   S          21 INZ(' ')
DMSGKEY        S          4  INZ(' ')
DMSGTYPE       S          10 INZ('*INFO ')
DSTACKENTRY    S          10 INZ('* ')
DSTACKCOUNTER  S          9B 0 INZ(2)
DERRCODE       DS
DBYTESIN       1          4B 0 INZ(0)
DBYTESOUT      5          8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C*-----
C  *ENTRY      PLIST
C              PARM          MVALUE      15 5
C              PARM          NVALUE      15 5
C*-----
C* Set the keyword in the rule array *
C*-----
C              MOVE      'ADAPTER1'  RULEARRAY
C              MOVE      'SET-MOFN'  RULEARRAY
C              Z-ADD      2           RULEARRAYCNT
C*-----
C* Set the verb data length to 8 *
C*-----
C              Z-ADD      8           VERBDATALEN
C*-----
C* Set the M and N value (Convert from decimal 15 5 to binary)*
C*-----
C              EVAL      M = MVALUE
C              EVAL      N = NVALUE
C*****
C* Call Cryptographic Facility Control SAPI *
C*****
C              CALLP      CSUACFC      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDTALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:

```

```

C                                     VERBDATALEN:
C                                     VERBDATA)
C*-----*
C* Check the return code *
C*-----*
C      RETURNCODE      IFGT      0
C*          *-----*
C*          * Send error message *
C*          *-----*
C              MOVEL      MSG(1)      MSGTEXT
C              MOVE      RETURNCODE    FAILRETC
C              MOVE      REASONCODE    FAILRSNC
C              EXSR      SNDMSG
C*
C              ELSE
C*          *****
C*          * Send success message *
C*          *****
C              MOVEL      MSG(2)      MSGTEXT
C              EXSR      SNDMSG
C*
C              ENDIF
C*
C              SETON                                     LR
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C                  CALL      'QMHSNDPM'
C                  PARM      MESSAGEID
C                  PARM      MESSAGEFILE
C                  PARM      MSGTEXT
C                  PARM      MSGLENGTH
C                  PARM      MSGTYPE
C                  PARM      STACKENTRY
C                  PARM      STACKCOUNTER
C                  PARM      MSGKEY
C                  PARM      ERRCODE
C                  ENDSR

```

**
CSUACFC failed with return/reason codes 9999/9999.
The request completed successfully.

Example: ILE C program for generating a retained key pair for cloning master keys: Change this program example to suit your needs for generating a retained key pair for cloning master keys.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* GENRETAIN */
/*
/* Sample program to generate a retained key to be used for
/* master key cloning.
/*
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 1999
/*
/* This material contains programming source code for your
/* consideration. These examples have not been thoroughly
/* tested under all conditions. IBM, therefore, cannot
/* guarantee or imply reliability, serviceability, or function
/* of these program. All programs contained herein are
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
/* these programs and files.
/*
/*

```

```

/*                                                                    */
/* Note: Input format is more fully described in Chapter 2 of        */
/*       IBM CCA Basic Services Reference and Guide                  */
/*       (SC31-8609) publication.                                    */
/*                                                                    */
/* Parameters: RETAINED_KEY_NAME                                     */
/*                                                                    */
/* Example:                                                         */
/* CALL PGM(GENRETAIN) PARM(TESTKEY)                                 */
/*                                                                    */
/* Note: This program assumes the card with the profile is          */
/*       already identified either by defaulting to the CRP01        */
/*       device or by being explicitly named using the               */
/*       Cryptographic_Resource_Allocate verb. Also this            */
/*       device must be varied on and you must be authorized        */
/*       to use this device description.                             */
/*                                                                    */
/* The Common Cryptographic Architecture (CCA) verbs used are       */
/* PKA_Key_Token_Build (CSNDPKB) and PKA_Key_Generate (CSNDPKG).    */
/*                                                                    */
/* Use these commands to compile this program on iSeries:          */
/* ADDLIB LIB(QCCA)                                                 */
/* CRTCMOD MODULE(GENRETAIN) SRCFILE(SAMPLE)                        */
/* CRTPGM PGM(GENRETAIN) MODULE(GENRETAIN)                          */
/*       BNDDIR(QCCA/QC6BNDDIR)                                     */
/*                                                                    */
/* Note: Authority to the CSNDPKG and CSNDPKB service programs     */
/*       in the QCCA library is assumed.                             */
/*                                                                    */
/*-----*/
#include <stdio.h>
#include <string.h>
#include "csucincl.h"

int main(int argc, char *argv[])
{
/*-----*/
/* Declares for CCA parameters                                     */
/*-----*/
long return_code = 0;
long reason_code = 0;
long exit_data_length = 0;
char exit_data[4];
char rule_array[24];
long rule_array_count;
long token_len = 2500;
char token[2500];
char regen_data[4];
char transport_key_id[4];
struct {
    short modlen;
    short modlenfld;
    short pubexplen;
    short prvexplen;
    long pubexp;
} key_struct; /* Key structure for PKA Key Token Build */
long key_struct_length;
long zero = 0;
/*-----*/
/* Declares for working with a PKA token                         */
/*-----*/
long pub_sec_len; /* Public section length */
long prv_sec_len; /* Private section length */
long cert_sec_len; /* Certificate section length */
long info_subsec_len; /* Information subsection length */
long offset; /* Offset into token */

```

```

long tempOffset;          /* (Another) Offset into token          */
long tempLength;         /* Length variable                          */
long tempLen1, tempLen2; /* temporary length variables              */
char pub_token[2500];
long pub_token_len;
long name_len;
char name[64];

int i;                   /* Loop counter                              */
FILE *fp;                /* File pointer                              */

if (argc < 2)           /* Check the number of parameters passed */
{
    printf("Need to enter a private key name\n");
    return 1;
}

memset(token,0,2500);    /* Initialize token to 0                      */
memcpy((void*)rule_array,"RSA-PRIVKEY-MGMT",16); /* Set rule array */
rule_array_count = 2;

memset(name, ' ', 64);   /* Copy key name parameter                   */
memcpy(name, argv[1], strlen(argv[1]));
name_len = 64;

/*-----*/
/* Initialize key structure */
/*-----*/
memset((void*)&key_struct, 0, sizeof(key_struct));
key_struct.modlen = 1024; /* Modulus length is 1024 */
key_struct.pubexplen = 3;
key_struct.pubexp = 0x01000100; /* Public exponent is 65537 */
key_struct_length = sizeof(key_struct);
/*****/
/* Call PKA_Key-Token_Build SAPI */
/*****/
CSNDPKB( &return_code, &reason_code, &exit_data_length,
        exit_data,
        &rule_array_count,
        rule_array,
        &key_struct_length,
        (unsigned char *)&key_struct,
        &name_len,
        name,
        &zero, /* 1 */
        NULL,
        &zero, /* 2 */
        NULL,
        &zero, /* 3 */
        NULL,
        &zero, /* 4 */
        NULL,
        &zero, /* 5 */
        NULL,
        &token_len,
        token);

if (return_code != 0)
{
    printf("PKA Key Token Build Failed : return code %d : reason code %d\n",
        return_code, reason_code);
    return 1;
}

/*****/

```

```

/* Build certificate */
/*****
/* Determine length of token from length */
/* bytes at offset 2 and 3. */
token_len = ((256 * token[2]) + token[3]);
/* Determine length of private key */
/* section from length bytes at offset */
/* 10. */
prv_sec_len = ((256 * token[10]) + token[11]);
/* Determine length of public key section*/
/* section from length bytes at offset */
/* 10 + private section length */
pub_sec_len = ((256 * token[prv_sec_len + 10]) +
token[prv_sec_len + 11]);

/* Calculate the signature section length*/
cert_sec_len = 328 + /* from the signature subsection length, */
20 + /* EID subsection length, */
12 + /* Serial number subsection length, */
4 + /* Information subsection header length, */
pub_sec_len + /* Public key subsection length, */
4; /* and the certificate section hdr length*/

offset = token_len; /* Offset for additions to token */

/* Fill in certicate section header */
tempLen1 = cert_sec_len;
tempLen1 >>= 8;
token[offset++] = 0x40;
token[offset++] = 0x00;
token[offset++] = tempLen1;
token[offset++] = cert_sec_len;

/* Fill in public key subsection */
token[offset++] = 0x41;
for (i = 1 ; i < pub_sec_len ; i ++ )
{
/* Copy public key to certificate */
token[offset++] = token[prv_sec_len +(i+8)];
}

/* Fill Optional Information Subsection Header */
info_subsec_len = 20 + /* Length of EID section */
12 + /* Length of serial number section */
4; /* Length of Info subsection header */
tempLen1 = info_subsec_len;
tempLen1 >>= 8;
token[offset++] = 0x42;
token[offset++] = 0x00;
token[offset++] = tempLen1;
token[offset++] = info_subsec_len;

/* Fill in Public Key Certificate EID subsection */
token[offset++] = 0x51;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x14;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;

```



```

token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;

/* Public key Certificate Serial Number TLV */
token[offset++] = 0x52;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x0c;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;
token[offset++] = 0x00;

/* Fill in Signature Subsection */
token[offset++] = 0x45;
token[offset++] = 0x00;
token[offset++] = 0x01;
token[offset++] = 0x48;
token[offset++] = 0x01;
token[offset++] = 0x01;

for (i = 0 ; i < 64 ;i++)
{
    /* Copy private key name out of private key name section */
    /* into certificate */
    token[offset++] =
        token[prv_sec_len + pub_sec_len + 12 + i];
}

token_len = offset + 258; /* add 258 to allow for digital sig. */
token[3] = token_len; /* Set new token length */
token[2] = token_len >> 8;

/*****
/* Generate Retained key using PKA token with certificate */
*****/
memcpy((void*)rule_array,"RETAIN CLONE ",16);
rule_array_count = 2;
memset(pub_token,0,2500);
pub_token_len = 2500;
memset(transport_key_id,0,4);

/*****
/* Call PKA_Key_Generate SAPI */
*****/
CSNDPKG( &return_code, &reason_code, &exit_data_length,
        exit_data,
        &rule_array_count,
        rule_array,
        &zero, /* regenerated data length */
        regen_data,
        &token_len,
        token,
        transport_key_id,
        &pub_token_len,
        pub_token);

if (return_code != 0)

```

```

{
    printf("PKA Key Generate Failed : return code %d :reason code %d\n",
        return_code, reason_code);
    return 1;
}

/*****
/* Write public key token out to file */
/*****
/* Append ".PUB" to key name */
memcpy((void*)&name[strlen(argv[1]),".PUB",5);
fp = fopen(name,"wb"); /* Open the file */

if (!fp)
{
    printf("File open failed\n");
}
else
{
    fwrite(pub_token, pub_token_len, 1, fp); /* Write token to file */

    fclose(fp); /* Close the file */
    printf("Public token written to file %s.\n", name);
}

name[strlen(argv[1])] = 0; /* Convert name to string */
printf("Private key %s is retained in the hardware\n", name);
return 0;
}

```

Example: ILE RPG program for generating a retained key pair for cloning master keys: Change this program example to suit your needs for generating a retained key pair for cloning master keys.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* GENRETAIN
D*
D* Sample program to generate a retained key to be used for
D* master key cloning.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: RETAINED_KEY_NAME
D*
D* Example:
D* CALL PGM(GENRETAIN) PARM(TESTKEY)
D*
D* Use these commands to compile this program on iSeries:
D* CRTPRGMOD MODULE(GENRETAIN) SRCFILE(SAMPLE)

```

```

D* CRTPGM  PGM(GENRETAIN) MODULE(GENRETAIN)
D*          BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSNDPKG and CSNDPKB service programs
D*       in the QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* PKA_Key_Token_Build (CSNDPKB) and PKA_Key_Generate (CSNDPKG).
D*
D*****
D*-----
D* Declare variables used by CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE  S          9B 0
D*          ** Reason code
DREASONCODE  S          9B 0
D*          ** Exit data length
DEXITDATALEN S          9B 0
D*          ** Exit data
DEXITDATA    S          4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY   S          16
D*          ** Token length
DTOKENLEN    S          9B 0 INZ(2500)
D*          ** Token and array for subscripting
DTOKEN       DS         2500
DTOKENARRAY  S          1  DIM(2500)
D*          ** Regeneration data
DREGENDATA   S          4  INZ(X'00000000')
D*          ** Transport key encrypting key
DTRANSPORTKEK S        4  INZ(X'00000000')
D*          ** Generated keyid
DGENKEY      S          2500
D*          ** Generated keyid length
DGENKEYLEN   S          9B 0 INZ(2500)
D*          ** Key name and length
DKEYNAME     S          64
DKEYNAMELEN  S          9B 0 INZ(64)
D*          ** Key structure for PKA Key Token Build
DKEYSTRUCT   DS
DMODLEN      1          2B 0
DMODLENFLD   3          4B 0
DPUBEXPLEN   5          6B 0
DPRVEXPLEN   7          8B 0
DPUBEXP      9          12B 0
D*          ** Null parms needed for CSNDPKB and CSNDPKG
DZERO        S          9B 0 INZ(0)
DNULLPTR     S          *  INZ(*NULL)
D*          ** Key structure length
DKEYSTRUCTLEN S        9B 0 INZ(12)
D*          ** Data structure for aligning 2 bytes into
D*          ** a 2 bytes integer
DLENSTRUCT   DS          2
DMSB         1          1
DLSB         2          2
DLENGTH      1          2B 0
D*          ** Private key section length
DPRVSECLLEN S          9B 0
D*          ** Public key section length
DPUBSECLLEN  S          9B 0
D*          ** Index into Token array
DINDEX       S          9B 0
D*          ** Declares for copying private key name
DNAMEPTR1    S          *

```

```

DNAME1          S          64   BASED(NAMEPTR1)
DNAMEPTR2       S          *
DNAME2          S          64   BASED(NAMEPTR2)
D*             **   Loop counter
DI              S          9B 0
D*             **   File descriptor
DFILED         S          9B 0
D*             **   File path and length
DPATH          S          80   INZ(*ALLX'00')
DPATHLEN       S          9B 0
D*             **   Open flag - Create on open, open for writing,
D*             **   and clear if exists
DOFLAG        S          10I 0 INZ(X'4A')
D*
D*****
D* Prototype for PKA_Key_Token_Build (CSNDPKB)
D*****
DCSNDPKB       PR
DRETCODE       9B 0
DRSNCODE       9B 0
DEXTDTALEN     9B 0
DEXTDTA        4
DRARRAYCT      9B 0
DRARRAY        16
DKEYSTRLEN     9B 0
DKEYSTR        10
DKEYNML        9B 0
DKEYNM         64
DRSRVLN1       9B 0
DRSRV1         *   VALUE
DRSRVLN2       9B 0
DRSRV2         *   VALUE
DRSRVLN3       9B 0
DRSRV3         *   VALUE
DRSRVLN4       9B 0
DRSRV4         *   VALUE
DRSRVLN5       9B 0
DRSRV5         *   VALUE
DTKNLEN        9B 0
DTKN           2500  OPTIONS(*VARSIZE)
D*
D*****
D* Prototype for PKA_Key_Generate (CSNDPKG)
D*****
DCSNDPKG       PR
DRETCOD        9B 0
DRSNCOD        9B 0
DEXTDTALN     9B 0
DEXTDT         4
DRARRAYCT      9B 0
DRARRAY        16
DREGDTAL       9B 0
DREGDTA        20  OPTIONS(*VARSIZE)
DSKTKNL        9B 0
DSKTKN         2500  OPTIONS(*VARSIZE)
DTRNKEK        64   OPTIONS(*VARSIZE)
DGENKEYL       9B 0
DGENKEY        2500  OPTIONS(*VARSIZE)
D*
D*****
D* Prototype for open()
D*****
D*   value returned = file descriptor (OK), -1 (error)
Dopen          PR          9B 0 EXTPROC('open')
D*   path name of file to be opened.
D              128   OPTIONS(*VARSIZE)
D*   Open flags

```

```

D          9B 0 VALUE
D*   (OPTIONAL) mode - access rights
D          10U 0 VALUE OPTIONS(*NOPASS)
D*   (OPTIONAL) codepage
D          10U 0 VALUE OPTIONS(*NOPASS)
D*
D*****
D* Prototype for write()
D*****
D*   value returned = number of bytes actually written, or -1
Dwrite    PR          9B 0 EXTPROC('write')
D*   File descriptor returned from open()
D          9B 0 VALUE
D*   Data to be written
D          1200   OPTIONS(*VARSIZE)
D*   Length of data to write
D          9B 0 VALUE
D*
D*****
D* Prototype for close()
D*****
D*   value returned = 0 (OK), or -1
Dclose    PR          9B 0 EXTPROC('close')
D*   File descriptor returned from open()
D          9B 0 VALUE
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSDPM API
D*-----
DMSG      S          75   DIM(4) CTDATA PERRCD(1)
DMSGLENGT S          9B 0 INZ(75)
D
DMSGTEXT          1   75
DSAPI             1   7
DFAILRETC         41  44
DFAILRSNC         46  49
DMESSAGEID        S          7   INZ(' ')
DMESSAGEFILE      S          21  INZ(' ')
DMSGKEY           S          4   INZ(' ')
DMSGTYPE          S          10  INZ('*INFO ')
DSTACKENTRY       S          10  INZ('* ')
DSTACKCOUNTER     S          9B 0 INZ(2)
DERRCODE         DS
DBYTESIN          1   4B 0 INZ(0)
DBYTESOUT         5   8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* * * * *
C   *ENTRY      PLIST
C               PARM          KEYNAMEPARM      50
C* *-----*
C* * Initialize tokens to 0 *
C* *-----*
C               MOVEL   *ALLX'00'   TOKEN
C               MOVEL   *ALLX'00'   GENKEY
C* *-----*
C* * Initialize key struct *
C* *-----*
C               Z-ADD   1024        MODLEN
C               Z-ADD   0           MODLENFLD
C               Z-ADD   3           PUBEXPLEN
C               Z-ADD   0           PRVEXPLEN
C               EVAL    PUBEXP = 65537 * 256
C* *-----*
C* * Copy key name from parm*

```

```

C*  *-----*
C          MOVEL    KEYNAMEPARM  KEYNAME
C*  *-----*
C*  * Set the keywords in the rule array *
C*  *-----*
C          MOVEL    'RSA-PRIV'   RULEARRAY
C          MOVE     'KEY-MGMT'   RULEARRAY
C          Z-ADD    2             RULEARRAYCNT
C*****
C* Call PKA_Key_Token_Build SAPI
C*****
C          CALLP    CSNDPKB      (RETURNCODE:
C                                REASONCODE:
C                                EXITDATALEN:
C                                EXITDATA:
C                                RULEARRAYCNT:
C                                RULEARRAY:
C                                KEYSTRUCTLEN:
C                                KEYSTRUCT:
C                                KEYNAMEL:
C                                KEYNAME:
C                                ZERO:
C                                NULLPTR:
C                                ZERO:
C                                NULLPTR:
C                                ZERO:
C                                NULLPTR:
C                                ZERO:
C                                NULLPTR:
C                                ZERO:
C                                NULLPTR:
C                                ZERO:
C                                NULLPTR:
C                                TOKENLEN:
C                                TOKEN)
C*  *-----*
C*  * Check the return code *
C*  *-----*
C          RETURNCODE  IFGT      0
C*  *-----*
C*  * Send failure message *
C*  *-----*
C          MOVEL    MSG(1)       MSGTEXT
C          MOVE     RETURNCODE   FAILRETC
C          MOVE     REASONCODE   FAILRSNC
C          MOVEL    'CSNDPKB'    SAPI
C          EXSR     SNDMSG
C          RETURN
C          ENDIF
C*
C*-----*
C* Build the certificate *
C*-----*
C*  Get the private section length. The length is at position 11
C*  of the token
C          EVAL     MSB = TOKENARRAY(10+1)
C          EVAL     LSB = TOKENARRAY(11+1)
C          MOVE     LENGTH      PRVSECLN
C*  Get the public section length. The length is at position
C*  (11 + Private key section length).
C          EVAL     MSB = TOKENARRAY(10 + PRVSECLN + 1)
C          EVAL     LSB = TOKENARRAY(11 + PRVSECLN + 1)
C          MOVE     LENGTH      PUBSECLN
C*  Calculate the certificate section length
C*  Cert Section length = Signature length (328) +
C*  EID section length (20) +
C*  Serial number length (12) +
C*  Info subsection header length (4) +
C*  Public Key section length +

```

```

C*          Cert section header length (4)
C          EVAL      LENGTH = 328 + 20 + 12 + 4 + PUBSECLN + 4
C*  Fill Certificate section header
C          MOVE      TOKENLEN      INDEX
C          EVAL      TOKENARRAY(INDEX +1) = X'40'
C          EVAL      TOKENARRAY(INDEX +2) = X'00'
C          EVAL      TOKENARRAY(INDEX +3) = MSB
C          EVAL      TOKENARRAY(INDEX +4) = LSB
C*  Fill in public key subsection
C          EVAL      TOKENARRAY(INDEX +5) = X'41'
C          ADD       5          INDEX
C          Z-ADD    1          I
C*  Copy the public key section of the token into the public key
C*  subsection of the certificate section.
C  I          DOWLT    PUBSECLN
C          EVAL      TOKENARRAY(INDEX + I) =
C          TOKENARRAY(PRVSECLN + I + 8 + 1)
C  1          ADD      I          I
C          ENDDO
C          EVAL      INDEX = INDEX + PUBSECLN - 1
C*  Fill in Optional Information subsection header
C          Z-ADD    36          LENGTH
C          EVAL      TOKENARRAY(INDEX +1) = X'42'
C          EVAL      TOKENARRAY(INDEX +2) = X'00'
C          EVAL      TOKENARRAY(INDEX +3) = MSB
C          EVAL      TOKENARRAY(INDEX +4) = LSB
C*  Fill in Public Key Certificate EID
C          EVAL      INDEX = INDEX + 4
C          EVAL      TOKENARRAY(INDEX +1) = X'51'
C          EVAL      TOKENARRAY(INDEX +4) = X'14'
C*  Fill in Public Key Certificate Serial Number TLV
C          EVAL      INDEX = INDEX + 20
C          EVAL      TOKENARRAY(INDEX +1) = X'52'
C          EVAL      TOKENARRAY(INDEX +4) = X'0C'
C*  Fill in Signature Subsection
C          EVAL      INDEX = INDEX + 12
C          EVAL      TOKENARRAY(INDEX +1) = X'45'
C          EVAL      TOKENARRAY(INDEX +3) = X'01'
C          EVAL      TOKENARRAY(INDEX +4) = X'48'
C          EVAL      TOKENARRAY(INDEX +5) = X'01'
C          EVAL      TOKENARRAY(INDEX +6) = X'01'
C*  Fill in private key name
C          EVAL      INDEX = INDEX + 6
C          EVAL      NAMEPTR1 = %ADDR(TOKENARRAY(INDEX +1))
C          EVAL      NAMEPTR2 =
C          %ADDR(TOKENARRAY(PRVSECLN+PUBSECLN+12+1))
C          MOVE     NAME2      NAME1
C*  Adjust token length
C          EVAL      LENGTH = INDEX + 64 + 258
C          MOVE     MSB      TOKENARRAY(3)
C          MOVE     LSB      TOKENARRAY(4)
C          EVAL      TOKENLEN = LENGTH
C*  *-----*
C*  * Set the keywords in the rule array *
C*  *-----*
C          MOVE     'RETAIN '  RULEARRAY
C          MOVE     'CLONE  '  RULEARRAY
C          Z-ADD    2          RULEARRAYCNT
C
C*-----*
C* Call PKA_Key_Generate SAPI *
C*-----*
C          CALLP    CSNDPKG      (RETURNCODE:
C          REASONCODE:
C          EXITDATALEN:
C          EXITDATA:
C          RULEARRAYCNT:

```

```

C                                     RULEARRAY:
C                                     ZERO:
C                                     REGENDATA:
C                                     TOKENLEN:
C                                     TOKEN:
C                                     TRANSPORTKEK:
C                                     GENKEYLEN:
C                                     GENKEY)
C*-----*
C* Check the return code *
C*-----*
C      RETURNCODE    IFGT      0
C*      *-----*
C*      * Send failure message *
C*      *-----*
C          MOVE      MSG(1)      MSGTEXT
C          MOVE      RETURNCODE  FAILRETC
C          MOVE      REASONCODE  FAILRSNC
C          MOVE      'CSNDPKG'   SAPI
C          EXSR      SNDMSG
C          RETURN
C          ENDIF
C*
C*      *-----*
C*      * Send success message *
C*      *-----*
C          MOVE      MSG(2)      MSGTEXT
C          EXSR      SNDMSG
C*
C*-----*
C* Write certificate out to file *
C*-----*
C*  ** Build path name
C          EVAL      PATHLEN = %LEN(%TRIM(KEYNAMEPARM))
C      PATHLEN      SUBST    KEYNAMEPARM:1 PATH
C          EVAL      %SUBST(PATH:PATHLEN+1:4) = '.PUB'
C*
C*  ** Open the file
C*
C          EVAL      FILED = open(PATH: OFLAG)
C*
C*  ** Check if open worked
C*
C      FILED      IFEQ      -1
C*
C*  ** Open failed, send an error message
C*
C          MOVE      MSG(3)      MSGTEXT
C          EXSR      SNDMSG
C*
C          ELSE
C*
C*  ** Open worked, write certificate out to file and close file
C*
C          CALLP      write      (FILED:
C                                GENKEY:
C                                GENKEYLEN)
C          CALLP      close      (FILED)
C*
C*  ** Send completion message
C*
C          MOVE      MSG(4)      MSGTEXT
C          EVAL      %SUBST(MSGTEXT: 32: PATHLEN + 4) =
C                                %SUBST(PATH: 1: PATHLEN + 4)
C          EXSR      SNDMSG
C          ENDIF
C*

```



```

C          SETON                      LR
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C          CALL      'QMHSNDPM'
C          PARM      MESSAGEID
C          PARM      MESSAGEFILE
C          PARM      MSGTEXT
C          PARM      MSGLENGTH
C          PARM      MSGTYPE
C          PARM      STACKENTRY
C          PARM      STACKCOUNTER
C          PARM      MSGKEY
C          PARM      ERRCODE
C          ENDSR
C*

```

```

**
CSNDPKB failed with return/reason codes 9999/9999.
The retained key was successfully created.
The file could not be opened.
The certificate was written to

```

Example: ILE C program for registering a public key hash: Change this program example to suit your needs for registering a hash of a public key certificate.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* REGHASH */
/* */
/* Sample program to register the hash of a CCA public key */
/* certificate. */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 1999 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: Stream file containing public key certificate */
/* */
/* Example: */
/* CALL PGM(REGHASH) PARM(CERTFILE) */
/* */
/* Note: This program assumes the card with the profile is */
/* already identified either by defaulting to the CRP01 */
/* device or by being explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/* */
/* The Common Cryptographic Architecture (CCA) verbs used are */
/* PKA_Public_Key_Hash_Register (CSNDPKH) and One_Way_Hash (WH). */

```

```

/* (CSNBOWH). */
/* */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(REGHASH) SRCFILE(SAMPLE) */
/* CRTPGM PGM(REGHASH) MODULE(REGHASH) */
/* BNDDIR(QCCA/QC6BNDDIR) */
/* */
/* Note: Authority to the CSNDPKH and CSNBOWH service programs */
/* in the QCCA library is assumed. */
/* */
/*-----*/
#include <stdio.h>
#include <string.h>
#include "csucincl.h"

int main(int argc, char *argv[])
{
/*-----*/
/* Declares for CCA parameters */
/*-----*/
long return_code = 0;
long reason_code = 0;
long exit_data_length = 0;
char exit_data[4];
char rule_array[24];
long rule_array_count;
long token_len = 2500;
char token[2500];
long chaining_vector_length = 128;
long hash_length = 20;
long text_length;
unsigned char chaining_vector[128];
unsigned char hash[20];
/*-----*/
/* Declares for working with a PKA token */
/*-----*/
long pub_sec_len; /* Public section length */
long cert_sec_len; /* Certificate section length */
long offset; /* Offset into token */
long tempOffset; /* (Another) Offset into token */
char name[64]; /* Registered key name */

long count; /* Number of bytes read from file */
FILE *fp; /* File pointer */

if (argc < 2) /* Check the number of parameters passed */
{
printf("Need to enter a public key name\n");
return 1;
}

memset(name, ' ',64); /* Copy key name (and pad) to a 64 byte
/* field. */
memcpy(name,argv[1],strlen(argv[1]));

fp = fopen(argv[1],"rb"); /* Open the file for reading */
if (!fp)
{
printf("File %s not found.\n",argv[1]);
return 1;
}

memset(token,0,2500); /* Initialize the token to 0 */
count = fread(token,1,2500,fp); /* Read the token from the file */
fclose(fp); /* Close the file */

```

```

        /* Determine length of token from length */
        /* bytes at offset 2 and 3. */
token_len = ((256 * token[2]) + token[3]);
if (count < token_len) /* Check if whole token was read in */
{
    printf("Incomplete token in file\n");
    return 1;
}

/*****/
/* Find the certificate offset in the token */
/* */
/* The layout of the token is */
/* */
/* - Token header - 8 bytes - including 2 length bytes */
/* - Public key section - length bytes at offset 10 overall */
/* - Private key name - 68 bytes */
/* - Certificate section */
/* */
/*****/
pub_sec_len = ((256 * token[10]) + token[11]);

offset = pub_sec_len + 68 + 8; /* Set offset to certificate section */

        /* Determine certificate section */
        /* length from the length bytes at */
        /* offset 2 of the section. */
cert_sec_len = ((256 * token[offset + 2]) + token[offset + 3]);
tempOffset = offset + 4; /* Set offset to first subsection */

/*-----*/
/* Parse each subsection of the certificate until the */
/* signature subsection is found or the end is reached.*/
/* (Identifier for signature subsection is Hex 45.) */
/*-----*/
while(token[tempOffset] != 0x45 &&
    tempOffset < offset + cert_sec_len)
{
    tempOffset += 256 * token[tempOffset + 2] + token[tempOffset+3];
}

/*-----*/
/* Check if no signature was found before the end of */
/* the certificate section. */
/*-----*/
if (token[tempOffset] != 0x45)
{
    printf("Invalid certificate\n");
    return 1;
}

/*****/
/* Hash the certificate */
/*****/
text_length = tempOffset - offset + 70; /* Text length is length */
/* of certificate subsection. */

memcpy((void*)rule_array,"SHA-1 ",8); /* Set rule array */
rule_array_count = 1;
chaining_vector_length = 128;
hash_length = 20;

CSNBOWH( &return_code, &reason_code, &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char*)rule_array,
        &text_length,

```

```

        &token[offset],
        &chaining_vector_length,
        chaining_vector,
        &hash_length,
        hash);

if (return_code != 0)
{
    printf("One_Way_Hash Failed : return reason %d/%d\n",
           return_code, reason_code);
    return 1;
}

/*****
/* Register the Hash */
/*****
        /* Set the rule array */
memcpy((void*)rule_array,"SHA-1 CLONE ",16);
rule_array_count = 2;
        /* Build the name of the retained */
        /* key from the file and "RETAINED"*/
memcpy(&name[strlen(argv[1])],".RETAINED",9);

CSNDPKH( &return_code, &reason_code, &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char*)rule_array,
        name,
        &hash_length,
        hash);

if (return_code != 0)
{
    printf("Public Key Register_Hash Failed : return reason %d/%d\n",
           return_code, reason_code);
    return 1;
}

name[strlen(argv[1]) + 9] = 0; /* Convert name to a string */
printf("Hash registered for %s.\n",name);
}

```

Example: ILE RPG program for registering a public key hash: Change this program example to suit your needs for registering a hash of a public key certificate.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* REGHASH
D*
D* Sample program to register the hash of a CCA public key
D* certificate.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*

```

```

D*
D* Note: Input format is more fully described in Chapter 2 of
D*     IBM CCA Basic Services Reference and Guide
D*     (SC31-8609) publication.
D*
D* Parameters: Stream file containing public key certificate
D*
D* Example:
D*   CALL PGM(REGHASH) PARM(CERTFILE)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(REGHASH) SRCFILE(SAMPLE)
D* CRTPGM   PGM(REGHASH) MODULE(REGHASH)
D*         BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSNDPKH and CSNBOWH service programs
D*       in the QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* PKA_Public_Key_Hash_Register (CSNDPKH) and One_Way_Hash
C* (CSNBOWH).
D*
D*****
D*-----
D* Declare variables used by CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE  S          9B 0
D*          ** Reason code
DREASONCODE  S          9B 0
D*          ** Exit data length
DEXITDATALEN S          9B 0
D*          ** Exit data
DEXITDATA    S          4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY   S          16
D*          ** Token length
DTOKENLEN    S          9B 0 INZ(2500)
D*          ** Token and array for subscribing token
DTOKEN       DS         2500
DTOKENARRAY  S          1 DIM(2500)
D*          ** Chaining vector length
DCHAINVCTLEN S          9B 0 INZ(128)
D*          ** Chaining vector
DCHAINVCT    S          128
D*          ** Hash length
DHASHLEN     S          9B 0 INZ(20)
D*          ** Hash
DHASH        S          20
D*          ** Text length
DTXTLENGTH   S          9B 0
D*          ** Name of retained key
DNAME        S          64
D*          ** Structure used for aligning 2 bytes into a
D*          ** 2 byte integer.
DLENSTRUCT   DS          2
DMSB         S          1
DLSB         S          2
DLENGTH      S          1 2B 0
D*
D*          ** Certificate section length
DCRTSECLLEN  S          9B 0
D*          ** Public key section length
DPUBSECLLEN  S          9B 0
D*          ** Index into PKA key token

```

```

DTKNINDEX      S          9B 0
D*             ** Index into PKA key token
DTMPINDEX      S          9B 0
D*             ** File descriptor
DFILED         S          9B 0
D*             ** File path and path length
DPATH          S          80  INZ(*ALLX'00')
DPATHLEN       S          9B 0
D*             ** Open Flag - Open for Read only
DOFLAG        S          10I 0 INZ(1)
D*
D*****
D* Prototype for PKA_Public_Key_Hash_Register (CSNDPKH)
D*****
DCSNDPKH       PR
DRETCOD        9B 0
DRSNCOD        9B 0
DEXTDTALN     9B 0
DEXTDT         4
DRARRYCT       9B 0
DRARRY         16
DKYNAM         64
DHSHL          9B 0
DHS           20  OPTIONS(*VARSIZE)
D*
D*****
D* Prototype for One_Way_Hash (CSNBOWH)
D*****
DCSNBOWH       PR
DRETCOD        9B 0
DRSNCOD        9B 0
DEXTDTALN     9B 0
DEXTDT         4
DRARRYCT       9B 0
DRARRY         16
DTXTLEN        9B 0
DTXT           500  OPTIONS(*VARSIZE)
DCHNVCTLEN     9B 0
DCHNVCT        128
DHSLEN         9B 0
DHS            20
D*
D*
D*****
D* Prototype for open()
D*****
D* value returned = file descriptor (OK), -1 (error)
Dopen          PR          9B 0 EXTPROC('open')
D* path name of file to be opened.
D              128  OPTIONS(*VARSIZE)
D* Open flags
D              9B 0 VALUE
D* (OPTIONAL) mode - access rights
D              10U 0 VALUE OPTIONS(*NOPASS)
D* (OPTIONAL) codepage
D              10U 0 VALUE OPTIONS(*NOPASS)
D*
D*****
D* Prototype for read()
D*****
D* value returned = number of bytes actually read, or -1
Dread         PR          9B 0 EXTPROC('read')
D* File descriptor returned from open()
D              9B 0 VALUE
D* Input buffer
D              2500  OPTIONS(*VARSIZE)
D* Length of data to be read

```

```

D                                9B 0 VALUE
D*
D*****
D* Prototype for close()
D*****
D*   value returned = 0 (OK), or -1
Dclose          PR              9B 0 EXTPROC('close')
D*   File descriptor returned from open()
D                                9B 0 VALUE
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG            S              75  DIM(6) CTDATA PERRCD(1)
DMSGLENGTH     S              9B 0 INZ(75)
D
DMSGTEXT       1              80
DSAPI          1              7
DFAILRETC     41             44
DFAILRSNC     46             49
DMESSAGEID    S              7  INZ('      ')
DMESSAGEFILE  S              21 INZ('      ')
DMSGKEY       S              4  INZ('      ')
DMSGTYPE      S              10 INZ('*INFO ')
DSTACKENTRY   S              10 INZ('*      ')
DSTACKCOUNTER S              9B 0 INZ(2)
DERRCODE      DS
DBYTESIN      1              4B 0 INZ(0)
DBYTESOUT     5              8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* *
C   *ENTRY      PLIST
C               PARM              FILEPARM      50
C*****
C* Open certificate file
C*****
C* *-----*
C* ** Build path name *
C* *-----*
C               EVAL      PATHLEN = %LEN(%TRIM(FILEPARM))
C   PATHLEN     SUBST     FILEPARM:1  PATH
C* *-----*
C* * Open the file *
C* *-----*
C               EVAL      FILED = open(PATH: OFLAG)
C* *-----*
C* * Check if open worked *
C* *-----*
C   FILED       IFEQ      -1
C* *-----*
C* * Open failed, send an error message *
C* *-----*
C               MOVEL     MSG(1)      MSGTEXT
C               EXSR      SNDMSG
C               RETURN
C*
C               ENDIF
C* *-----*
C* * Open worked, read certificate and close the file *
C* *-----*
C               EVAL      TOKENLEN = read(FILED: TOKEN: TOKENLEN)
C               CALLP     close      (FILED)
C*
C* *-----*

```

```

C*      * Check if read operation was OK      *
C*      *-----*
C      TOKENLEN      IFEQ      -1
C      MOVEL      MSG(2)      MSGTEXT
C      EXSR      SNDMSG
C      RETURN
C      ENDIF
C*
C*      *-----*
C*      * Check if certificate length is valid *
C*      * The length bytes start at position 3 *
C*      *-----*
C      EVAL      MSB = TOKENARRAY(3)
C      EVAL      LSB = TOKENARRAY(4)
C      LENGTH      IFLT      TOKENLEN
C*      *-----*
C*      * Certificate length is not valid *
C*      *-----*
C      MOVEL      MSG(3)      MSGTEXT
C      EXSR      SNDMSG
C      RETURN
C      ENDIF
C*
C*****
C* Find the certificate in the token
C*
C* The layout of the token is
C*
C* - Token header - 8 bytes - including 2 length bytes
C* - Public key section - length bytes at position 3 (11 overall)
C* - Private key name - 68 bytes
C* - Certificate section
C*
C* Note: 1 is added because RPG arrays start at 1.
C*****
C      EVAL      MSB = TOKENARRAY(11)
C      EVAL      LSB = TOKENARRAY(12)
C      EVAL      PUBSECLN = LENGTH
C      EVAL      TKNINDEX = PUBSECLN + 68 + 8 + 1
C*
C*      *-----*
C*      * Determine length of certificate section *
C*      * Length bytes are at position 2 of the *
C*      * section.
C*      *-----*
C      EVAL      MSB = TOKENARRAY(TKNINDEX + 2)
C      EVAL      LSB = TOKENARRAY(TKNINDEX + 3)
C      EVAL      CRTSECLN = LENGTH
C      EVAL      TMPINDEX = TKNINDEX + 4
C*
C*      *-----*
C*      * Parse each subsection of the certificate until the *
C*      * signature subsection is found or the end is reached.*
C*      * (Identifier for signature subsection is Hex 45.) *
C*      *-----*
C      DOW      (TOKENARRAY(TMPINDEX) <> X'45') AND
C      (TMPINDEX < TKNINDEX + CRTSECLN)
C      EVAL      MSB = TOKENARRAY(TMPINDEX + 2)
C      EVAL      LSB = TOKENARRAY(TMPINDEX + 3)
C      TMPINDEX      ADD      LENGTH      TMPINDEX
C      ENDDO
C*
C*      *-----*
C*      * Check if no signature was found before the end of *
C*      * the certificate section.
C*      *-----*
C      IF      TOKENARRAY(TMPINDEX) <> X'45'

```



```

C          MOVEL      MSG(4)      MSGTEXT
C          EXSR       SNDMSG
C          RETURN
C          ENDIF
C*
C*****
C* Hash the certificate
C*****
C* -----*
C* * Calculate the length to hash *
C* -----*
C          EVAL      TXTLENGTH = TMPINDEX - TKNINDEX + 70
C* -----*
C* * Set the keywords in the rule array *
C* -----*
C          MOVEL      'SHA-1 '      RULEARRAY
C          Z-ADD      1              RULEARRAYCNT
C* -----*
C* * Call One Way Hash SAPI *
C* -----*
C          CALLP      CSNBOWH      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     TXTLENGTH:
C                                     TOKENARRAY(TKNINDEX):
C                                     CHAINVCTLEN:
C                                     CHAINVCT:
C                                     HASHLEN:
C                                     HASH)
C* -----*
C* * Check the return code *
C* -----*
C          RETURNCODE  IFGT      0
C* -----*
C* * Send failure message *
C* -----*
C          MOVEL      MSG(5)      MSGTEXT
C          MOVE       RETURNCODE  FAILRETC
C          MOVE       REASONCODE  FAILRSNC
C          MOVEL      'CSNBOWH'   SAPI
C          EXSR       SNDMSG
C          RETURN
C          ENDIF
C*
C*****
C* Register the certificate hash
C*****
C* -----*
C* * Set the keywords in the rule array *
C* -----*
C          MOVEL      'SHA-1 '      RULEARRAY
C          MOVE       'CLONE '      RULEARRAY
C          Z-ADD      2              RULEARRAYCNT
C* -----*
C* * Build the key name (FILENAME.RETAINED) *
C* -----*
C          EVAL      %SUBST(NAME: 1: PATHLEN) =
C                                     %SUBST(PATH: 1: PATHLEN)
C          EVAL      %SUBST(NAME:PATHLEN+1:9) = '.RETAINED'
C* -----*
C* * Call PKA Public Key Hash Register *
C* -----*
C          CALLP      CSNDPKH      (RETURNCODE:

```

```

C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     NAME:
C                                     HASHLEN:
C                                     HASH)
C* *-----*
C* * Check the return code *
C* *-----*
C   RETURNCODE   IFGT   0
C* *-----*
C* * Send failure message *
C* *-----*
C           MOVEL   MSG(5)   MSGTEXT
C           MOVE   RETURNCODE FAILRETC
C           MOVE   REASONCODE FAILRSNC
C           MOVEL   'CSNDPKH' SAPI
C           EXSR   SNDMSG
C           ELSE
C* *-----*
C* * Send success message *
C* *-----*
C           MOVEL   MSG(6)   MSGTEXT
C           EVAL   %SUBST(MSGTEXT: 41: PATHLEN + 9) =
C                   %SUBST(NAME: 1: PATHLEN + 9)
C           EXSR   SNDMSG
C           ENDIF
C*
C           SETON                                     LR
C*
C*****
C* Subroutine to send a message
C*****
C   SNDMSG   BEGSR
C           CALL   'QMHSNDPM'
C           PARM   MESSAGEID
C           PARM   MESSAGEFILE
C           PARM   MSGTEXT
C           PARM   MSGLENGTH
C           PARM   MSGTYPE
C           PARM   STACKENTRY
C           PARM   STACKCOUNTER
C           PARM   MSGKEY
C           PARM   ERRCODE
C           ENDSR

```

```

**
The file could not be opened.
There was an error reading from the file.
The length of the certificate is not valid.
The certificate is not valid.
CSNBOWH failed with return/reason codes 9999/9999.
The hash was successfully registered as

```

Example: ILE C program for registering a public key certificate: Change this program example to suit your needs for registering a public key certificate.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* REGPUBKEY */
/* */
/* Sample program to register a CCA public key certificate */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 1999 */
/* */

```

```

/* This material contains programming source code for your          */
/* consideration. These examples have not been thoroughly          */
/* tested under all conditions. IBM, therefore, cannot            */
/* guarantee or imply reliability, serviceability, or function    */
/* of these program. All programs contained herein are            */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF            */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE      */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for*/
/* these programs and files.                                       */
/*                                                                    */
/*                                                                    */
/* Note: Input format is more fully described in Chapter 2 of     */
/*       IBM CCA Basic Services Reference and Guide                */
/*       (SC31-8609) publication.                                  */
/*                                                                    */
/* Parameters: Stream file containing public key certificate      */
/*                                                                    */
/* Example:                                                        */
/* CALL PGM(REGPUBKEY) PARM(CERTFILE)                             */
/*                                                                    */
/* Note: This program assumes the card with the profile is        */
/* already identified either by defaulting to the CRP01           */
/* device or by being explicitly named using the                  */
/* Cryptographic_Resource_Allocate verb. Also this               */
/* device must be varied on and you must be authorized           */
/* to use this device description.                                 */
/*                                                                    */
/* The Common Cryptographic Architecture (CCA) verb used is      */
/* PKA_Public_Key_Register (CSNDPKR).                             */
/*                                                                    */
/* Use these commands to compile this program on iSeries:        */
/* ADDLIB LIB(QCCA)                                               */
/* CRTCMOD MODULE(REGPUBKEY) SRCFILE(SAMPLE)                      */
/* CRTPGM PGM(REGPUBKEY) MODULE(REGPUBKEY)                       */
/* BNDDIR(QCCA/QC6BNDDIR)                                         */
/*                                                                    */
/* Note: Authority to the CSNDPKR service program                 */
/* in the QCCA library is assumed.                                */
/*                                                                    */
/*-----*/
#include <stdio.h>
#include <string.h>
#include "csucincl.h"

int main(int argc, char *argv[])
{
/*-----*/
/* Declares for CCA parameters                                     */
/*-----*/
long return_code = 0;
long reason_code = 0;
long exit_data_length = 0;
char exit_data[4];
char rule_array[24];
long rule_array_count;
long token_len = 2500;
char token[2500];
/*-----*/
/* Declares for working with a PKA token                         */
/*-----*/
long pub_sec_len;          /* Public section length      */
long cert_sec_len;        /* Certificate section length */
long offset;              /* Offset into token         */
long tempOffset;          /* (Another) Offset into token */
char name[64];            /* Registered key name       */

```

```

long count;          /* Number of bytes read from file      */
FILE *fp;           /* File pointer                                             */

if (argc < 2)       /* Check the number of parameters passed */
{
    printf("Need to enter a public key name\n");
    return 1;
}

memset(name, ' ',64); /* Copy key name (and pad) to a 64 byte
/* field.
memcpy(name,argv[1],strlen(argv[1]));

fp = fopen(argv[1],"rb"); /* Open the file for reading
if (!fp)
{
    printf("File %s not found.\n",argv[1]);
    return 1;
}

memset(token,0,2500); /* Initialize the token to 0
count = fread(token,1,2500,fp); /* Read the token from the file
fclose(fp);          /* Close the file

/* Determine length of token from length
/* bytes at offset 2 and 3.
token_len = ((256 * token[2]) + token[3]);
if (count < token_len) /* Check if whole token was read in
{
    printf("Incomplete token in file\n");
    return 1;
}

/*****
/* Find the certificate length in the token
/*
/* The layout of the token is
/*
/* - Token header - 8 bytes - including 2 length bytes
/* - Public key section - length bytes at offset 2
/* - Private key name - 68 bytes
/* - Certificate section
*****/
pub_sec_len = ((256 * token[10]) + token[11]);

offset = pub_sec_len + 68 + 8; /* Set offset to certiiicate section */

/* Determine certificate section
/* length from the length bytes at
/* offset 2 of the section.
cert_sec_len = ((256 * token[offset + 2]) + token[offset + 3]);

/*****
/* Register the Public Key
*****/
memcpy((void*)rule_array,"CLONE ",8); /* Set rule array
rule_array_count = 1;

/* Build the name of the retained
/* key from the file and "RETAINED"*/
memcpy(&name[strlen(argv[1])],".RETAINED",9);

CSNDPKR( &return_code, &reason_code, &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char*)rule_array,
        name,
        &cert_sec_len,

```

```

        &token[offset]);

if (return_code != 0)
{
    printf("Public Key Register Failed : return reason %d/%d\n",
        return_code, reason_code);
    return 1;
}

name[strlen(argv[1]) + 9] = 0; /* Convert name to a string */
printf("Public key registered for %s.\n",name);
}

```

Example: ILE RPG program for registering a public key certificate: Change this program example to suit your needs for registering a public key certificate.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* REGPUBKEY
D*
D* Sample program to register a CCA public key
D* certificate.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: Stream file containing public key certificate
D*
D* Example:
D* CALL PGM(REGPUBKEY) PARM(CERTFILE)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(REGPUBKEY) SRCFILE(SAMPLE)
D* CRTPGM PGM(REGPUBKEY) MODULE(REGPUBKEY)
D* BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSNDPKR service program
D* in the QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* PKA_Public_Key_Register (CSNDPKR).
D*
D*****
D*-----
D* Declare variables used by CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0

```

```

D*          ** Exit data length
DEXITDATALEN S          9B 0
D*          ** Exit data
DEXITDATA    S          4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY   S         16
D*          ** Token length
DTOKENLEN    S         9B 0 INZ(2500)
D*          ** Token and array for subscribing token
DTOKEN       DS        2500
DTOKENARRAY  S         1 DIM(2500)
D*          ** Name of retained key
DNAME        S         64
D*          ** Structure used for aligning 2 bytes into a
D*          ** 2 byte integer.
DLENSTRUCT   DS         2
DMSB         S         1 1
DLSB         S         2 2
DLENGTH      S         1 2B 0
D*          ** Certificate section length
DCRTSECLLEN S         9B 0
D*          ** Public key section length
DPUBSECLLEN S         9B 0
D*          ** Index into PKA key token
DTKNINDEX    S         9B 0
D*          ** Index into PKA key token
DTMPINDEX    S         9B 0
D*          ** File descriptor
DFILED       S         9B 0
D*          ** File path and path length
DPATH        S         80 INZ(*ALLX'00')
DPATHLEN     S         9B 0
D*          ** Open Flag - Open for Read only
DOFLAG       S         10I 0 INZ(1)
D*
D*****
D* Prototype for PKA_Public_Key_Register (CSNDPKR)
D*****
DCSNDPKR     PR
DRETCOD      S         9B 0
DRSNCOD      S         9B 0
DEXTDTALN   S         9B 0
DEXTDT       S         4
DRARRAYCT    S         9B 0
DRARRAY      S         16
DKYNAM       S         64
DCRTLEN      S         9B 0
DCRT         S         500 OPTIONS(*VARSIZE)
D*
D*****
D* Prototype for open()
D*****
D* value returned = file descriptor (OK), -1 (error)
Dopen        PR         9B 0 EXTPROC('open')
D* path name of file to be opened.
D           S         128 OPTIONS(*VARSIZE)
D* Open flags
D           S         9B 0 VALUE
D* (OPTIONAL) mode - access rights
D           S         10U 0 VALUE OPTIONS(*NOPASS)
D* (OPTIONAL) codepage
D           S         10U 0 VALUE OPTIONS(*NOPASS)
D*
D*****
D* Prototype for read()

```

```

D*****
D*   value returned = number of bytes actually read, or -1
Dread      PR          9B 0 EXTPROC('read')
D*   File descriptor returned from open()
D          9B 0 VALUE
D*   Input buffer
D          2500  OPTIONS(*VARSIZE)
D*   Length of data to be read
D          9B 0 VALUE
D*
D*****
D* Prototype for close()
D*****
D*   value returned = 0 (OK), or -1
Dclose     PR          9B 0 EXTPROC('close')
D*   File descriptor returned from open()
D          9B 0 VALUE
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG       S          75  DIM(5) CTDATA PERRCD(1)
DMSGLENGT S          9B 0 INZ(75)
D          DS
DMSGTEXT   1          80
DFAILRETC 41          44
DFAILRSNC 46          49
DMESSAGEID S          7  INZ(' ')
DMESSAGEFILE S        21 INZ(' ')
DMSGKEY    S          4  INZ(' ')
DMSGTYPE   S          10 INZ('*INFO ')
DSTACKENTRY S         10 INZ('* ')
DSTACKCOUNTER S       9B 0 INZ(2)
DERRCODE   DS
DBYTESIN   1          4B 0 INZ(0)
DBYTESOUT  5          8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* *
C   *ENTRY      PLIST
C               PARM          FILEPARM      50
C*****
C* Open certificate file
C*****
C* *-----*
C* ** Build path name *
C* *-----*
C          EVAL      PATHLEN = %LEN(%TRIM(FILEPARM))
C   PATHLEN      SUBST  FILEPARM:1  PATH
C* *-----*
C* * Open the file *
C* *-----*
C          EVAL      FILED = open(PATH: OFLAG)
C* *-----*
C* * Check if open worked *
C* *-----*
C   FILED          IFEQ      -1
C* *-----*
C* * Open failed, send an error message *
C* *-----*
C          MOVEL      MSG(1)      MSGTEXT
C          EXSR      SNDMSG
C          RETURN
C*
C          ENDIF

```

```

C*      *-----*
C*      * Open worked, read certificate and close the file *
C*      *-----*
C          EVAL      TOKENLEN = read(FILED: TOKEN: TOKENLEN)
C          CALLP      close          (FILED)
C*
C*      *-----*
C*      * Check if read operation was OK *
C*      *-----*
C  TOKENLEN      IFEQ      -1
C                MOVEL      MSG(2)      MSGTEXT
C                EXSR      SNDMSG
C                RETURN
C                ENDIF
C*
C*      *-----*
C*      * Check if certificate length is valid *
C*      * The length bytes start at position 3 *
C*      *-----*
C                EVAL      MSB = TOKENARRAY(3)
C                EVAL      LSB = TOKENARRAY(4)
C  LENGTH        IFLT      TOKENLEN
C*
C*      *-----*
C*      * Certificate length is not valid *
C*      *-----*
C                MOVEL      MSG(3)      MSGTEXT
C                EXSR      SNDMSG
C                RETURN
C                ENDIF
C*
C*****
C* Find the certificate in the token
C*
C* The layout of the token is
C*
C* - Token header - 8 bytes - including 2 length bytes
C* - Public key section - length bytes at position 3 (11 overall)
C* - Private key name - 68 bytes
C* - Certificate section
C*
C* Note: 1 is added because RPG arrays start at 1.
C*****
C                EVAL      MSB = TOKENARRAY(11)
C                EVAL      LSB = TOKENARRAY(12)
C                EVAL      PUBSECLN = LENGTH
C                EVAL      TKNINDEX = PUBSECLN + 68 + 8 + 1
C*
C*      *-----*
C*      * Determine length of certificate section *
C*      * Length bytes are at position 2 of the *
C*      * section.
C*      *-----*
C                EVAL      MSB = TOKENARRAY(TKNINDEX + 2)
C                EVAL      LSB = TOKENARRAY(TKNINDEX + 3)
C                EVAL      CRTSECLN = LENGTH
C*
C*****
C* Register the public key
C*****
C*      *-----*
C*      * Set the keywords in the rule array *
C*      *-----*
C                MOVEL      'CLONE '      RULEARRAY
C                Z-ADD      1            RULEARRAYCNT
C*
C*      * Build the key name (FILENAME.RETAINED) *
C*      *-----*

```



```

C          EVAL      %SUBST(NAME: 1: PATHLEN) =
C                      %SUBST(PATH: 1: PATHLEN)
C          EVAL      %SUBST(NAME:PATHLEN+1:9) = '.RETAINED'

C* *-----*
C* * Call PKA Public Key Register *
C* *-----*
C          CALLP      CSNDPKR          (RETURNCODE:
C                                         REASONCODE:
C                                         EXITDATALEN:
C                                         EXITDATA:
C                                         RULEARRAYCNT:
C                                         RULEARRAY:
C                                         NAME:
C                                         CRTSECLN:
C                                         TOKENARRAY(TKNINDEX))
C* *-----*
C* * Check the return code *
C* *-----*
C          RETURNCODE  IFGT      0
C* *-----*
C* * Send failure message *
C* *-----*
C          MOVE      MSG(4)      MSGTEXT
C          MOVE      RETURNCODE  FAILRETC
C          MOVE      REASONCODE  FAILRSNC
C          EXSR      SNDMSG
C          ELSE
C* *-----*
C* * Send success message *
C* *-----*
C          MOVE      MSG(5)      MSGTEXT
C          EVAL      %SUBST(MSGTEXT: 41: PATHLEN + 9) =
C                      %SUBST(NAME: 1: PATHLEN + 9)
C          EXSR      SNDMSG
C          ENDIF
C*
C          SETON                      LR
C*
C*****
C* Subroutine to send a message
C*****
C          SNDMSG      BEGSR
C          CALL        'QMHSNDPM'
C          PARM        MESSAGEID
C          PARM        MESSAGEFILE
C          PARM        MSGTEXT
C          PARM        MSGLENGTH
C          PARM        MSGTYPE
C          PARM        STACKENTRY
C          PARM        STACKCOUNTER
C          PARM        MSGKEY
C          PARM        ERRCODE
C          ENDSR

```

**

The file could not be opened.
 There was an error reading from the file.
 The length of the certificate is not valid.
 CSNDPKR failed with return/reason codes 9999/9999.
 The hash was successfully registered as

Example: ILE C program for certifying a public key token: Change this program example to suit your needs for certifying a public key token.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* CERTKEY */
/*
/* Sample program to certify a CCA public key certificate to be
/* used for master key cloning.
/*
/*
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 1999
/*
/*
/* This material contains programming source code for your
/* consideration. These examples have not been thoroughly
/* tested under all conditions. IBM, therefore, cannot
/* guarantee or imply reliability, serviceability, or function
/* of these programs. All programs contained herein are
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
/* these programs and files.
/*
/*
/*
/* Note: Input format is more fully described in Chapter 2 of
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication.
/*
/*
/* Parameters: FILENAME - File containing public key token
/* RETAINED_KEY_NAME - Name of key to certify token
/*
/* Example:
/* CALL PGM(CERTKEY) PARM(MYKEY.PUB CERTKEY)
/*
/*
/* Note: This program assumes the card with the profile is
/* already identified either by defaulting to the CRP01
/* device or by being explicitly named using the
/* Cryptographic_Resource_Allocate verb. Also this
/* device must be varied on and you must be authorized
/* to use this device description.
/*
/*
/* The Common Cryptographic Architecture (CCA) verbs used are
/* Digital_Signature_Generate (CSNDDSG) and One_Way_Hash (CSNBOWH).
/*
/*
/* Use these commands to compile this program on iSeries:
/* ADDLIB LIB(QCCA)
/* CRTCMOD MODULE(CERTKEY) SRCFILE(SAMPLE)
/* CRTPGM PGM(CERTKEY) MODULE(CERTKEY)
/* BNDDIR(QCCA/QC6BNDDIR)
/*
/* Note: Authority to the CSNDDSG and CSNBOWH service programs
/* in the QCCA library is assumed.
/*
/*-----*/
#include <stdio.h>
#include <string.h>
#include "csucincl.h"
#include "decimal.h"

extern void QDCXLATE(decimal(5,0), char *, char*, char *);
#pragma linkage (QDCXLATE, OS, nowiden)

int main(int argc, char *argv[])
{
/*-----*/
/* Declares for CCA parameters */
/*-----*/
long return_code = 0;
long reason_code = 0;
long exit_data_length = 0;
char exit_data[4];

```

```

char rule_array[24];
long rule_array_count;
long token_len = 2500;
char token[2500];
long chaining_vector_length = 128;
long hash_length = 20;
long text_length;
unsigned char chaining_vector[128];
unsigned char hash[20];
long signature_length = 256;
long signature_bit_length;
/*-----*/
/* Declares for working with a PKA token */
/*-----*/
long pub_sec_len; /* Public section length */
long cert_sec_len; /* Certificate section length */
long offset; /* Offset into token */
long tempOffset; /* (Another) Offset into token */
long tempLength; /* Length variable */
char name[64]; /* Private key name */
char SAname[64]; /* Share administration or certifying
/* key name.
char SAnameASCII[64]; /* Share admin key name in ASCII
long SAname_length = 64; /* Length of Share admin key name
long count; /* Number of bytes read from file
decimal(5,0) xlate_length = 64; /* Packed decimal variable
/* needed for call to QDCXLATE.
FILE *fp; /* File pointer

if (argc < 3) /* Check the number of parameters passed */
{
printf("Need to enter a public key name and SA key\n");
return 1;
}

name[0] = 0; /* Make copy of name parameters
strcpy(name,argv[1]);
memset(SAname, ' ', 64); /* Make copy of Share Admin key name
memcpy(SAname,argv[2],strlen(argv[2]));

fp = fopen(name,"rb"); /* Open the file containing the token
if (!fp)
{
printf("File %s not found.\n",argv[1]);
return 1;
}

memset(token,0,2500); /* Read the token from the file
count = fread(token,1,2500,fp);
fclose(fp);
/* Determine length of token from length
/* bytes at offset 2 and 3.
token_len = ((256 * token[2]) + token[3]);
if (count < token_len) /* Check if whole token was read in
{
printf("Incomplete token in file\n");
return 1;
}

/*****/
/* Find the certificate offset in the token */
/*
/* The layout of the token is */
/*
/* - Token header - 8 bytes - including 2 length bytes */
/* - Public key section - length bytes at offset 10 overall */
/* - Private key name - 68 bytes */

```

```

/* - Certificate section */
/* */
/*****/
pub_sec_len = ((256 * token[10]) + token[11]);

offset = pub_sec_len + 68 + 8; /* Set offset to certificate section */

/* Determine certificate section */
/* length from the length bytes at */
/* offset 2 of the section. */
cert_sec_len = ((256 * token[offset + 2]) + token[offset + 3]);
tempOffset = offset + 4; /* Set offset to first subsection */

/*-----*/
/* Parse each subsection of the certificate until the */
/* signature subsection is found or the end is reached.*/
/* (Identifier for signature subsection is Hex 45.) */
/*-----*/
while(token[tempOffset] != 0x45 &&
tempOffset < offset + cert_sec_len)
{
tempOffset += 256 * token[tempOffset + 2] + token[tempOffset+3];
}

/*-----*/
/* Check if no signature was found before the end of */
/* the certificate section. */
/*-----*/
if (token[tempOffset] != 0x45)
{
printf("Invalid certificate\n");
return 1;
}

/*****/
/* Replace Private key name in certificate with the */
/* Share admin key name (expressed in ASCII). */
/*****/
text_length = tempOffset - offset + 70;
memcpy(SAnameASCII,SAname,64);

/*-----*/
/* Convert the Share Admin key name to ASCII */
/*-----*/
QDCXLATE(xlate_length, SAnameASCII, "QASCII ", "QSYS ");
memcpy(&token[tempOffset + 6], SAnameASCII, 64);

/*****/
/* Hash the certificate */
/*****/
memcpy((void*)rule_array,"SHA-1 ",8);
rule_array_count = 1;
chaining_vector_length = 128;
hash_length = 20;

CSNBOWH( &return_code, &reason_code, &exit_data_length,
exit_data,
&rule_array_count,
(unsigned char*)rule_array,
&text_length,
&token[offset],
&chaining_vector_length,
chaining_vector,
&hash_length,
hash);

if (return_code != 0)
{

```

```

printf("One_Way_Hash Failed : return reason %d/%d\n",
      return_code, reason_code);
return 1;
}

/*****
/* Create a signature */
*****/
memcpy((void*)rule_array,"ISO-9796",8);
rule_array_count = 1;

CSNDDSG( &return_code, &reason_code, &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char*)rule_array,
        &SName_length,
        SName,
        &hash_length,
        hash,
        &signature_length,
        &signature_bit_length,
        &token[tempOffset+70]);

if (return_code != 0)
{
printf("Digital Signature Generate Failed : return reason %d/%d\n",
      return_code, reason_code);
return 1;
}

/*-----*/
/* Check if the new signature is longer than the */
/* original signature */
/*-----*/
if((token[tempOffset + 2] * 256 + token[tempOffset + 3]) - 70 !=
signature_length)
{
printf("Signature Length change from %d to %d.\n",
      token[tempOffset + 2] * 256 + token[tempOffset + 3] - 70,
      signature_length);

/* Adjust length in signature subsection */
token[tempOffset + 2] = signature_length >> 8;
token[tempOffset + 3] = signature_length;

/* Adjust length in certificate section */
token[offset + 2] = (text_length + signature_length) >> 8;
token[offset + 3] = text_length + signature_length;

/* Adjust length in token header section */
tempLength = 8 + pub_sec_len + 68 + text_length +
signature_length;
token[2] = tempLength >> 8;
token[3] = tempLength;
}
else tempLength = token[2] * 256 + token[3];

/*****
/* Write certified public key out to a file */
*****/
strcat(name,".CRT"); /* Append .CRP to filename */
fp = fopen(name,"wb"); /* Open the certificate file */
if (!fp)
{
printf("File open failed for output\n");
}
else

```

```

    {
        fwrite(token, 1, tempLength, fp);
        fclose(fp);
        printf("Public token written to file %s.\n",name);
    }
}

```

Example: ILE RPG program for certifying a public key token: Change this program example to suit your needs for certifying a public key token.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* CERTKEY
D*
D* Sample program to certify a CCA public key certificate to be
D* used for master key cloning.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: FILENAME - File containing public key token
D* RETAINED_KEY_NAME - Name of key to certify token
D*
D* Example:
D* CALL PGM(CERTKEY) PARM(MYKEY.PUB CERTKEY)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(CERTKEY) SRCFILE(SAMPLE)
D* CRTPGM PGM(CERTKEY) MODULE(CERTKEY)
D* BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSNDDSG and CSNBOWH service programs
D* in the QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Digital_Signature_Generate (CSNDDSG) and One_Way_Hash (CSNBOWH).
D*
D*****
D*-----
D* Declare variables used by CCA SAPI calls
D*-----
D* ** Return code
DRETURNCODE S 9B 0
D* ** Reason code
DREASONCODE S 9B 0
D* ** Exit data length
DEXITDATALEN S 9B 0
D* ** Exit data
DEXITDATA S 4
D* ** Rule array count

```

```

DRULEARRAYCNT    S          9B 0
D*               ** Rule array
DRULEARRAY       S          16
D*               ** Token length
DTOKENLEN        S          9B 0 INZ(2500)
D*               ** Token and array for subscripting token
DTOKEN           DS         2500
DTOKENARRAY      S          1   DIM(2500)
D*               ** Chaining vector length
DCHAINVCTLEN     S          9B 0 INZ(128)
D*               ** Chaining vector
DCHAINVCT        S          128
D*               ** Hash length
DHASHLEN         S          9B 0 INZ(20)
D*               ** Hash
DHASH            S          20
D*               ** Text length
DTXTLENGTH       S          9B 0
D*               ** Signature length
DSIGLENGTH       S          9B 0 INZ(256)
D*               ** Signature length in bits
DSIGBITLEN       S          9B 0
D*-----
D* Declare variables for working with tokens
D*-----
D*               ** NAMEPTR and NAME are used for copying
D*               ** private key name
DNAMEPTR         S          *
DNAME            S          64   BASED(NAMEPTR)
D*               ** Share administrator (certifying key) name length
DSNAMEELEN       S          9B 0
D*               ** Share administrator (certifying key) name
DSANAME          S          64
D*               ** Share administrator name expressed in ASCII
DSANAMEASC       S          64
D*               ** Certificate section length
DCRTSECTLEN     S          9B 0
D*               ** Public key section length
DPUBSECTLEN     S          9B 0
D*               ** Index into PKA key token
DTKNINDEX        S          9B 0
D*               ** Index into PKA key token
DTMPINDEX        S          9B 0
D*               ** Structure used for aligning 2 bytes into a
D*               ** 2 byte integer.
DLENSTRUCT       DS         2
DMSB             S          1   1
DLSB             S          2   2
DLENGTH          S          1   2B 0
D*               ** File descriptor
DFILED           S          9B 0
D*               ** File path and path length
DPATH            S          80   INZ(*ALLX'00')
DPATHLEN         S          9B 0
D*               ** Open flag - Create on open, open for writing,
D*               ** and clear if exists
DOFLAGW         S          10I 0 INZ(X'4A')
D*               ** Open Flag - Open for Read only
DOFLAGR         S          10I 0 INZ(1)
D*               ** Declares for calling QDCXLATE API
DXTABLE         S          10   INZ('QASCII ')
DLIB             S          10   INZ('QSYS ')
DXLATLEN        S          5   0 INZ(64)
D
D*
D*****
D* Prototype for Digital_Signature_Generate (CSNDDSG)

```

```

D*****
DCSNDDSG          PR
DRETCOD           9B 0
DRSNCOD           9B 0
DEXTDTALN        9B 0
DEXTDT            4
DRARRYCT         9B 0
DRARRY           16
DKEYIDLEN        9B 0
DKEYID           2500  OPTIONS(*VARSIZE)
DSHL             9B 0
DHS             20  OPTIONS(*VARSIZE)
DSIGFLDL        9B 0
DSIGBTL         9B 0
DSIGFLD         256  OPTIONS(*VARSIZE)
D*
D*****
D* Prototype for One_Way_Hash (CSNBOWH)
D*****
DCSNBOWH          PR
DRETCOD           9B 0
DRSNCOD           9B 0
DEXTDTALN        9B 0
DEXTDT            4
DRARRYCT         9B 0
DRARRY           16
DTXTLEN          9B 0
DTXT             500  OPTIONS(*VARSIZE)
DCHNVCTLEN       9B 0
DCHNVCT          128
DHSLEN           9B 0
DHS              20
D*
D*
D*****
D* Prototype for open()
D*****
D*   value returned = file descriptor (OK), -1 (error)
Dopen            PR          9B 0  EXTPROC('open')
D*   path name of file to be opened.
D               128  OPTIONS(*VARSIZE)
D*   Open flags
D               9B 0  VALUE
D*   (OPTIONAL) mode - access rights
D               10U 0  VALUE  OPTIONS(*NOPASS)
D*   (OPTIONAL) codepage
D               10U 0  VALUE  OPTIONS(*NOPASS)
D*
D*****
D* Prototype for read()
D*****
D*   value returned = number of bytes actually read, or -1
Dread           PR          9B 0  EXTPROC('read')
D*   File descriptor returned from open()
D               9B 0  VALUE
D*   Input buffer
D               2500  OPTIONS(*VARSIZE)
D*   Length of data to be read
D               9B 0  VALUE
D*
D*****
D* Prototype for write()
D*****
D*   value returned = number of bytes written, or -1
Dwrite          PR          9B 0  EXTPROC('write')
D*   File descriptor returned from open()
D               9B 0  VALUE

```



```

D*   Output buffer
D           2500   OPTIONS(*VARSIZE)
D*   Length of data to be written
D           9B 0 VALUE
D*
D*****
D* Prototype for close()
D*****
D*   value returned = 0 (OK), or -1
Dclose      PR           9B 0 EXTPROC('close')
D*   File descriptor returned from open()
D           9B 0 VALUE
D*
D*-----
D*           ** Declares for sending messages to the
D*           ** job log using the QMHSNDPM API
D*-----
DMSG        S           75   DIM(7) CTDATA PERRCD(1)
DMSGLENGTH  S           9B 0 INZ(75)
D           DS
DMSGTEXT    1           75
DSAPI       1           7
DFAILRETC   41         44
DFAILRSNC   46         49
DMESSAGEID  S           7   INZ(' ')
DMESSAGEFILE S         21   INZ(' ')
DMSGKEY     S           4   INZ(' ')
DMSGTYPE    S           10  INZ('*INFO ')
DSTACKENTRY S           10  INZ('* ')
DSTACKCOUNTER S        9B 0 INZ(2)
DERRCODE    DS
DBYTESIN    1           4B 0 INZ(0)
DBYTESOUT   5           8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C*****
C   *ENTRY      PLIST
C               PARM           FILEPARM      32
C               PARM           CKEY          32
C*****
C* Open certificate file
C*****
C* *-----*
C* ** Build path name *
C* *-----*
C   PATHLEN     EVAL      PATHLEN = %LEN(%TRIM(FILEPARM))
C   PATHLEN     SUBST     FILEPARM:1  PATH
C* *-----*
C* * Open the file *
C* *-----*
C               EVAL      FILED = open(PATH: OFLAGR)
C* *-----*
C* * Check if open worked *
C* *-----*
C   FILED       IFEQ      -1
C* *-----*
C* * Open failed, send an error message *
C* *-----*
C               MOVEL     MSG(1)      MSGTEXT
C               EXSR      SNDMSG
C               RETURN
C*
C               ENDIF
C* *-----*
C* * Open worked, read certificate and close the file *
C* *-----*

```

```

C          EVAL      TOKENLEN = read(FILED: TOKEN: TOKENLEN)
C          CALLP     close          (FILED)
C*
C* -----*
C* * Check if read operation was OK *
C* -----*
C  TOKENLEN      IFEQ      -1
C                MOVEL     MSG(2)      MSGTEXT
C                EXSR      SNDMSG
C                ENDIF
C*
C* -----*
C* * Check if certificate length is valid *
C* -----*
C          EVAL      MSB = TOKENARRAY(3)
C          EVAL      LSB = TOKENARRAY(4)
C  LENGTH        IFLT      TOKENLEN
C*
C* -----*
C* * Certificate length is not valid *
C* -----*
C                MOVEL     MSG(3)      MSGTEXT
C                EXSR      SNDMSG
C                RETURN
C                ENDIF
C*
C*****
C* Find the certificate in the token
C*
C* The layout of the token is
C*
C* - Token header - 8 bytes - including 2 length bytes
C* - Public key section - length bytes at offset 2
C* - Private key name - 68 bytes
C* - Certificate section
C*
C*****
C* -----*
C* * Certificate starts after the public key header section *
C* -----*
C          EVAL      MSB = TOKENARRAY(11)
C          EVAL      LSB = TOKENARRAY(12)
C          EVAL      PUBSECLN = LENGTH
C          EVAL      TKNINDEX = PUBSECLN + 68 + 8 + 1
C*
C* -----*
C* * Determine length of certificate section *
C* -----*
C          EVAL      MSB = TOKENARRAY(TKNINDEX + 2)
C          EVAL      LSB = TOKENARRAY(TKNINDEX + 3)
C          EVAL      CRTSECLN = LENGTH
C          EVAL      TMPINDEX = TKNINDEX + 4
C*
C* -----*
C* * Parse each subsection of the certificate until the *
C* * signature subsection is found or the end is reached.*
C* * (Identifier for signature subsection is Hex 45.) *
C* -----*
C          DOW      (TOKENARRAY(TMPINDEX) <> X'45') AND
C                (TMPINDEX < TKNINDEX + CRTSECLN)
C          EVAL      MSB = TOKENARRAY(TMPINDEX + 2)
C          EVAL      LSB = TOKENARRAY(TMPINDEX + 3)
C  TMPINDEX      ADD      LENGTH      TMPINDEX
C                ENDDO
C*
C* -----*
C* * Check if no signature was found before the end of *
C* * the certificate section. *

```

```

C*  *-----*
C          IF      TOKENARRAY(TMPINDEX) <> X'45'
C          MOVE    MSG(4)      MSGTEXT
C          EXSR   SNDMSG
C          RETURN
C          ENDIF
C*
C*****
C* Sign the Certificate
C*****
C*  *-----*
C*  * Convert the Certifying Keyname to ASCII  *
C*  *-----*
C          EVAL    SANAMELEN = %LEN(%TRIM(CKEY))
C  SANAMELEN     SUBST  CKEY:1      SANAME
C          MOVE    SANAME      SANAMEASC
C          CALL    'QDCXLATE'
C          PARM
C          PARM      XLATLEN
C          PARM      SANAMEASC
C          PARM      XTABLE
C          PARM      LIB
C*  *-----*
C*  * Replace the private key name in the certificate  *
C*  *-----*
C          EVAL    NAMEPTR = %ADDR(TOKENARRAY(TMPINDEX + 6))
C          MOVE    SANAMEASC      NAME
C*  *-----*
C*  * Calculate length of data to hash  *
C*  * TKNINDEX is the start of the certificate,  *
C*  * TMPINDEX is start of signature subsection,  *
C*  * signature subsection header is 70 bytes long  *
C*  *-----*
C          EVAL    TXTLENGTH = TMPINDEX - TKNINDEX + 70
C*  *-----*
C*  * Set the keywords in the rule array  *
C*  *-----*
C          MOVE    'SHA-1 '      RULEARRAY
C          Z-ADD   1              RULEARRAYCNT
C*  *-----*
C*  * Call One Way Hash SAPI  *
C*  *-----*
C          CALLP   CSNBOWH      (RETURNCODE:
C                                REASONCODE:
C                                EXITDATALEN:
C                                EXITDATA:
C                                RULEARRAYCNT:
C                                RULEARRAY:
C                                TXTLENGTH:
C                                TOKENARRAY(TKNINDEX):
C                                CHAINVCTLEN:
C                                CHAINVCT:
C                                HASHLEN:
C                                HASH)
C*  *-----*
C*  * Check the return code  *
C*  *-----*
C          RETURNCODE  IFGT      0
C*  *-----*
C*  * Send failure message *
C*  *-----*
C          MOVE    MSG(5)      MSGTEXT
C          MOVE    RETURNCODE  FAILRET
C          MOVE    REASONCODE  FAILRNC
C          MOVE    'CSNBOWH'  SAPI
C          EXSR   SNDMSG
C          RETURN
C          ENDIF

```

```

C* *-----*
C* * Set the keywords in the rule array *
C* *-----*
C          MOVEL      'ISO-9796'      RULEARRAY
C          Z-ADD      1                RULEARRAYCNT
C* *-----*
C* * Adjust TMPINDEX to where signature starts*
C* * in the certificate *
C* *-----*
C  TMPINDEX      ADD      70          TMPINDEX
C* *-----*
C* * Set the Key name length *
C* *-----*
C          Z-ADD      64              SANAMELEN
C* *-----*
C* * Call Digital Signature Generate SAPI *
C* *-----*
C          CALLP      CSNDDSG          (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     SANAMELEN:
C                                     SANAME:
C                                     HASHLEN:
C                                     HASH:
C                                     SIGLENGTH:
C                                     SIGBITLEN:
C                                     TOKENARRAY(TMPINDEX))
C* *-----*
C* * Check the return code *
C* *-----*
C  RETURNCODE     IFGT      0
C* *-----*
C* * Send failure message *
C* *-----*
C          MOVEL      MSG(5)          MSGTEXT
C          MOVE      RETURNCODE      FAILRETC
C          MOVE      REASONCODE      FAILRSNC
C          MOVEL      'CSNDDSG'      SAPI
C          EXSR      SNDMSG
C          RETURN
C          ENDIF
C* *-----*
C* * Check if the new signature is longer than the *
C* * original signature *
C* *-----*
C* ** Adjust TMPINDEX back the start of the subsection
C  TMPINDEX      SUB      70          TMPINDEX
C* ** Get two byte length of subsection
C          EVAL      MSB = TOKENARRAY(TMPINDEX + 2)
C          EVAL      LSB = TOKENARRAY(TMPINDEX + 3)
C* ** Subtract length of subsection header
C  LENGTH        SUB      70          LENGTH
C* ** Compare old length with new length
C  LENGTH        IFNE      SIGLENGTH
C* *-----*
C* * Adjust certificate lengths *
C* *-----*
C* ** Adjust signature length
C          EVAL      LENGTH = SIGLENGTH
C          EVAL      TOKENARRAY(TMPINDEX + 2) = MSB
C          EVAL      TOKENARRAY(TMPINDEX + 3) = LSB
C* ** Adjust certificate section length
C          EVAL      LENGTH = LENGTH + TXTLENGTH

```

```

C          EVAL      TOKENARRAY(TKNINDEX + 2) = MSB
C          EVAL      TOKENARRAY(TKNINDEX + 3) = LSB
C*      ** Adjust length in token header section
C          EVAL      LENGTH = LENGTH + 8 + PUBSECLN + 68
C          EVAL      TOKENARRAY(3) = MSB
C          EVAL      TOKENARRAY(4) = LSB
C          Z-ADD     LENGTH      TOKENLEN
C          ENDIF
C*
C*****
C* Write certified public key out to a file
C*****
C*      ** Build path name
C          EVAL      %SUBST(PATH:PATHLEN+1:4) = '.CRT'
C*
C*      ** Open the file
C*
C          EVAL      FILED = open(PATH: OFLAGW)
C*
C*      ** Check if open worked
C*
C      FILED      IFEQ      -1
C*
C*      ** Open failed, send an error message
C*
C          MOVEL     MSG(6)      MSGTEXT
C          EXSR      SNDMSG
C*
C          ELSE
C*
C*      ** Open worked, write certificate out to file and close file
C*
C          CALLP     write      (FILED:
C                               TOKEN:
C                               TOKENLEN)
C          CALLP     close      (FILED)
C*
C*      ** Send completion message
C*
C          MOVEL     MSG(7)      MSGTEXT
C          EVAL      %SUBST(MSGTEXT: 41: PATHLEN + 4) =
C                               %SUBST(PATH: 1: PATHLEN + 4)
C          EXSR      SNDMSG
C          ENDIF
C*
C          SETON                                          LR
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C          CALL      'QMHSNDPM'
C          PARM      MESSAGEID
C          PARM      MESSAGEFILE
C          PARM      MSGTEXT
C          PARM      MSGLENGTH
C          PARM      MSGTYPE
C          PARM      STACKENTRY
C          PARM      STACKCOUNTER
C          PARM      MSGKEY
C          PARM      ERRCODE
C          ENDSR
C*

```

```

**
The input file could not be opened.
There was an error reading from the file.
The length of the certificate is not valid.

```

The certificate is not valid.
 CSNBOWH failed with return/reason codes 9999/9999.
 The output file could not be opened.
 The certified token was written to file

Example: ILE C program for obtaining a master key share: Change this program example to suit your needs for obtaining a master key share.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* GETSHARE */
/* */
/* Sample program to obtain a master key share as part of the */
/* master key cloning process. */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 1999 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */
/* Parameters: Share number */
/* Name of share sender private key */
/* Name of certifying key */
/* Stream file containing receiver certificate */
/* */
/* Example: */
/* CALL PGM(GETSHARE) PARM(2 SENDR SAKEY RECVR.PUB) */
/* */
/* Note: This program assumes the card with the profile is */
/* already identified either by defaulting to the CRP01 */
/* device or by being explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/* */
/* The Common Cryptographic Architecture (CCA) verbs used is */
/* Master_Key_Distribution (CSUAMKD). */
/* */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(GETSHARE) SRCFILE(SAMPLE) */
/* CRTPGM PGM(GETSHARE) MODULE(GETSHARE) */
/* BNDDIR(QCCA/QC6BNDDIR) */
/* */
/* Note: Authority to the CSUAMKD service program */
/* in the QCCA library is assumed. */
/* */
/*-----*/
#include <stdio.h>
#include <string.h>
#include "csucincl.h"
#include "decimal.h"

```

```

extern void QDCXLATE(decimal(5,0), char *, char*, char *);
#pragma linkage (QDCXLATE, OS, nowiden)

int main(int argc, char *argv[])
{
/*-----*/
/* Declares for CCA parameters */
/*-----*/
long return_code = 0;
long reason_code = 0;
long exit_data_length = 0;
char exit_data[4];
char rule_array[24];
long rule_array_count;
long token_len = 2500;
char token[2500];
long cloneInfoKeyLength = 500;
unsigned char cloneInfoKey[500];
long cloneInfoLength = 400;
unsigned char cloneInfo[400];
long shareIdx;
char name[64];
char SAname[64];
/*-----*/
/* Declares for working with a PKA token */
/*-----*/
long pub_sec_len; /* Public section length */
long prv_sec_len; /* Private section length */
long cert_sec_len; /* Certificate section length */
long info_subsec_len; /* Information subsection length */
long offset; /* Offset into token */
long tempOffset; /* (Another) Offset into token */
long tempLength; /* Length variable */
long tempLen1, tempLen2; /* temporary length variables */

char cloneShare[] = "cloneShare00"; /* Base cloning share filename */
long count; /* Number of bytes read in from file */
decimal(15,5) shareParm; /* Packed 15 5 var used for converting
/* from packed 15 5 to binary. Numeric
/* parms on iSeries are passed as dec 15 5*/
FILE *fp; /* File pointer

if (argc < 5) /* Check the number of parameters passed */
{
printf("Need to Share index, Sender name, SA name, and cert\n");
return 1;
}

/* Convert the packed decimal 15 5 parm
/* to binary.
memcpy(&shareParm,argv[1],sizeof(shareParm));
shareIdx = shareParm;
memset(name,' ',64); /* Copy the Private key name parm to a
memcpy(name,argv[2],strlen(argv[2])); /* 64 byte space padded var.
memset(SAname,' ',64); /* Copy the Share Admin name parm to a
memcpy(SAname,argv[3],strlen(argv[3]));/* 64 byte space padded var.

fp = fopen(argv[4],"rb"); /* Open the file containing the token
if (!fp)
{
printf("File %s not found.\n",argv[4]);
return 1;
}

memset(token,0,2500); /* Read the token from the file
count = fread(token,1,2500,fp);

```

```

fclose(fp);          /* Close the file          */

                        /* Determine length of token from length */
                        /* bytes at offset 2 and 3.          */
token_len = ((256 * token[2]) + token[3]);
if (count < token_len) /* Check if whole token was read in */
{
    printf("Incomplete token in file\n");
    return 1;
}

/*****/
/* Find the certificate offset in the token          */
/*          */
/* The layout of the token is          */
/*          */
/* - Token header - 8 bytes - including 2 length bytes */
/* - Public key section - length bytes at offset 10 overall */
/* - Private key name - 68 bytes          */
/* - Certificate section          */
/*          */
/*****/
pub_sec_len = ((256 * token[10]) + token[11]);

offset = pub_sec_len + 68 + 8; /* Set offset to certicate section */

                        /* Determine certificate section */
                        /* length from the length bytes at */
                        /* offset 2 of the section.          */
cert_sec_len = ((256 * token[offset + 2]) + token[offset + 3]);

/*****/
/* Obtain a share          */
/*****/
memcpy((void*)rule_array,"OBTAIN ",8); /* Set rule array */
rule_array_count = 1;

CSUAMKD( &return_code, &reason_code, &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char*)rule_array,
        &shareIdx,
        name,
        SName,
        &cert_sec_len,
        &token[offset],
        &cloneInfoKeyLength,
        cloneInfoKey,
        &cloneInfoLength,
        cloneInfo);

if (return_code != 0)
{
    printf("Master Key Distribution Failed : return reason %d/%d\n",
        return_code, reason_code);
    return 1;
}
else
{
    /*****/
    /* Write signed token out to a file          */
    /*****/
    printf("Master Key Distribution worked\n");

                                /* Build file path name          */

```



```

if (shareIdx < 9) cloneShare[11] = '0' + shareIdx;
else
{
    cloneShare[10] = '1';
    cloneShare[11] = '0' + shareIdx - 10;
}

fp = fopen(cloneShare,"wb"); /* Open the file */
if (!fp)
{
    printf("File %s not be opened for output.\n",cloneShare);
    return 1;
}

/* Write out the length of KEK */
fwrite((char*)&cloneInfoKeyLength,1,4,fp);
/* Write out the KEK */
fwrite((char*)cloneInfoKey,1,cloneInfoKeyLength,fp);
/* Write out the length of info */
fwrite((char*)&cloneInfoLength,1,4,fp);
/* Write out the clone info */
fwrite((char*)cloneInfo,1,cloneInfoLength,fp);
printf("Clone share %d written to %s.\n",shareIdx,cloneShare);

fclose(fp); /* Close the file */
return 0;
}
}

```

Example: ILE RPG program for obtaining a master key share: Change this program example to suit your needs for obtaining a master key share.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* GETSHARE
D*
D* Sample program to obtain a master key share as part of the
D* master key cloning process.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: Share number
D*             Name of share sender private key
D*             Name of certifying key
D*             Path name of stream file containing receiver certificate
D*
D* Example:
D* CALL PGM(GETSHARE) PARM(2 SENDR SAKEY RECVR.PUB)
D*

```

```

D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(GETSHARE) SRCFILE(SAMPLE)
D* CRTPGM PGM(GETSHARE) MODULE(GETSHARE)
D* BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSUAMKD service program
D* in the QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used is
D* Master_Key_Distribution (CSUAMKD).
D*
D*****
D*-----
D* Declare variables used by CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S         9B 0
D*          ** Exit data
DEXITDATA S           4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY S           16
D*          ** Token length
DTOKENLEN S           9B 0 INZ(2500)
D*          ** Token and array for subscribing
DTOKEN DS            2500
DTOKENARRAY S           1 DIM(2500)
D*          ** Private key name
DPRVNAME S            64
D*          ** Certifying key name
DCERTKEY S            64
D*
DLSTRUCT DS
D*          ** Clone KEK length - one is binary form and the
D*          ** other is used for reading the value from a file
DCLONEKEKL S          9B 0 INZ(500)
DCLONEKEKLC S          1 4
D*          ** Clone info length - one is binary form and the
D*          ** other is used for reading the value from a file
DCLONEINFOLEN S        9B 0 INZ(400)
DCLONEINFOLENC S        5 8
D*          ** Cloning key-encrypting-key
DCLONEKEK S           500
D*          ** Cloning info
DCLONEINFO S           400
D*          ** Share index
DSHAREIDX S           9B 0
D*          ** Data structure for aligning 2 bytes into
D*          ** a 2 bytes integer
DLENSTRUCT DS          2
DMSB S           1 1
DLSB S           2 2
DLENGTH S           1 2B 0
D*          ** Certificate section length
DCRTSECLN S           9B 0
D*          ** Public key section length
DPUBSECLN S           9B 0
D*          ** Index into Token array
DTKNINDEX S           9B 0
D*          ** Number of bytes to write out to a file
DOUTLEN S            9B 0
D*          ** File descriptor

```

```

D* FILED          S          9B 0
D* ** File path and length
DPSTRUCT         DS
DPATH            80      INZ(*ALLX'00')
DSIDX            11      12B 0
DPATHLEN        S          9B 0
D* ** Open Flag - Open for Read only
DOFLAGR        S          10I 0 INZ(1)
D* ** Open flag - Create on open, open for writing,
D* ** and clear if exists
DOFLAGW        S          10I 0 INZ(X'4A')
D* ** Base name of file to store cloning share
DSHAREFILE      S          12      INZ('cloneShare00')
D*
D*****
D* Prototype for Master_Key_Distribution (CSUAMKD)
D*****
DCSUAMKD        PR
DRETCOD         9B 0
DRSNCOD         9B 0
DEXTDTALN      9B 0
DEXTDT          4
DRARRYCT       9B 0
DRARRY         16
DSHRINDX       9B 0
DKYNAM         64
DCRTKYNAM      64
DCRTL          9B 0
DCRT           2500  OPTIONS(*VARSIZE)
DCLNKEKL       9B 0
DCLNKEK        1200  OPTIONS(*VARSIZE)
DCLNL          9B 0
DCLN           400   OPTIONS(*VARSIZE)
D*
D*****
D* Prototype for open()
D*****
D* value returned = file descriptor (OK), -1 (error)
Dopen          PR          9B 0 EXTPROC('open')
D* path name of file to be opened.
D             128      OPTIONS(*VARSIZE)
D* Open flags
D             9B 0 VALUE
D* (OPTIONAL) mode - access rights
D             10U 0 VALUE OPTIONS(*NOPASS)
D* (OPTIONAL) codepage
D             10U 0 VALUE OPTIONS(*NOPASS)
D*
D*****
D* Prototype for write()
D*****
D* value returned = number of bytes written, or -1
Dwrite        PR          9B 0 EXTPROC('write')
D* File descriptor returned from open()
D             9B 0 VALUE
D* Output buffer
D             2500  OPTIONS(*VARSIZE)
D* Length of data to be written
D             9B 0 VALUE
D*
D*****
D* Prototype for read()
D*****
D* value returned = number of bytes actually read, or -1
Dread         PR          9B 0 EXTPROC('read')
D* File descriptor returned from open()
D             9B 0 VALUE

```

```

D*   Input buffer
D           2500   OPTIONS(*VARSIZE)
D*   Length of data to be read
D           9B 0 VALUE
D*
D*****
D* Prototype for close()
D*****
D*   value returned = 0 (OK), or -1
Dclose      PR           9B 0 EXTPROC('close')
D*   File descriptor returned from open()
D           9B 0 VALUE
D*
D*-----
D*           ** Declares for sending messages to the
D*           ** job log using the QMHSNDPM API
D*-----
DMSG        S           75   DIM(6) CTDATA PERRCD(1)
DMSGLENGTH  S           9B 0 INZ(80)
D           DS
DMSGTEXT    1           80
DSAPI       1           7
DFAILRETC   41          44
DFAILRSNC   46          49
DMESSAGEID  S           7   INZ(' ')
DMESSAGEFILE S          21  INZ(' ')
DMSGKEY     S           4   INZ(' ')
DMSGTYPE    S           10  INZ('*INFO ')
DSTACKENTRY S           10  INZ('* ')
DSTACKCOUNTER S          9B 0 INZ(2)
DERRCODE    DS
DBYTESIN    1           4B 0 INZ(0)
DBYTESOUT   5           8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* *
C   *ENTRY      PLIST
C               PARM                SINDEXT 15 5
C               PARM                PRVKEY  32
C               PARM                SAKEY   32
C               PARM                FILEPARM 32
C*****
C* Open certificate file
C*****
C* *-----*
C* ** Build path name *
C* *-----*
C           EVAL      PATHLEN = %LEN(%TRIM(FILEPARM))
C   PATHLEN  SUBST    FILEPARM:1  PATH
C* *-----*
C* * Open the file *
C* *-----*
C           EVAL      FILED = open(PATH: OFLAGR)
C* *-----*
C* * Check if open worked *
C* *-----*
C   FILED      IFEQ    -1
C* *-----*
C* * Open failed, send an error message *
C* *-----*
C           MOVEL    MSG(1)      MSGTEXT
C           EXSR     SNDMSG
C           RETURN
C*
C           ENDIF
C* *-----*

```

```

C*   * Open worked, read certificate and close file *
C*   *-----*
C           EVAL      TOKENLEN = read(FILED: TOKEN: TOKENLEN)
C           CALLP     close          (FILED)
C*
C*   *-----*
C*   * Check if read operation was OK *
C*   *-----*
C  TOKENLEN      IFEQ      -1
C                MOVEL     MSG(2)      MSGTEXT
C                EXSR      SNDMSG
C                ENDIF
C*
C*   *-----*
C*   * Check if certificate length is valid *
C*   * The length bytes start at position 3 *
C*   *-----*
C                EVAL      MSB = TOKENARRAY(3)
C                EVAL      LSB = TOKENARRAY(4)
C  LENGTH        IFLT      TOKENLEN
C*
C*   *-----*
C*   * Certificate length is not valid *
C*   *-----*
C                MOVEL     MSG(3)      MSGTEXT
C                EXSR      SNDMSG
C                RETURN
C                ENDIF
C*
C*****
C* Find the certificate in the token
C*
C* The layout of the token is
C*
C* - Token header - 8 bytes - including 2 length bytes
C* - Public key section - length bytes at position 3 (11 overall)
C* - Private key name - 68 bytes
C* - Certificate section
C*
C* Note: 1 is added because RPG arrays start at 1.
C*****
C                EVAL      MSB = TOKENARRAY(11)
C                EVAL      LSB = TOKENARRAY(12)
C                EVAL      PUBSECLN = LENGTH
C                EVAL      TKNINDEX = PUBSECLN + 68 + 8 + 1
C*
C*   *-----*
C*   * Determine length of certificate section *
C*   * Length bytes are at position 2 of the *
C*   * section. *
C*   *-----*
C                EVAL      MSB = TOKENARRAY(TKNINDEX + 2)
C                EVAL      LSB = TOKENARRAY(TKNINDEX + 3)
C                EVAL      CRTSECLN = LENGTH
C*
C*****
C* Obtain a certificate
C*****
C*   *-----*
C*   * Set share index number *
C*   * (Convert from packed 15 5 to binary) *
C*   *-----*
C                Z-ADD     SINDEX      SHAREIDX
C*
C*   * Set private key name *
C*   *-----*
C                EVAL      LENGTH = %LEN(%TRIM(PRVKEY))
C  LENGTH        SUBST     PRVKEY:1    PRVNAME

```

```

C* *-----*
C* * Set certifying key name *
C* *-----*
C          EVAL          LENGTH = %LEN(%TRIM(SAKEY))
C          LENGTH      SUBST      SAKEY:1      CERTKEY
C* *-----*
C* * Set the keywords in the rule array *
C* *-----*
C          MOVE          'OBTAIN '      RULEARRAY
C          Z-ADD          1              RULEARRAYCNT
C* *-----*
C* * Call Master Key Distribution SAPI *
C* *-----*
C          CALLP          CSUAMKD      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     SHAREIDX:
C                                     PRVNAME:
C                                     CERTKEY:
C                                     CRTSECLN:
C                                     TOKENARRAY(TKNINDEX):
C                                     CLONEKEKL:
C                                     CLONEKEK:
C                                     CLONEINFOLEN:
C                                     CLONEINFO)
C* *-----*
C* * Check the return code *
C* *-----*
C          RETURNCODE    IFGT          0
C* *-----*
C* * Send failure message *
C* *-----*
C          MOVE          MSG(4)        MSGTEXT
C          MOVE          RETURNCODE    FAILRETC
C          MOVE          REASONCODE    FAILRSNC
C          MOVE          'CSUAMKD'     SAPI
C          EXSR          SNDMSG
C          RETURN
C          ENDIF
C*
C*****
C* Write share out to a file
C*****
C* ** Build path name
C          MOVE          *ALLX'00'     PATH
C          MOVE          SHAREFILE     PATH
C          SIDX          ADD          SHAREIDX     SIDX
C          SHAREIDX     IFGE          10
C          SIDX          ADD          246         SIDX
C          ENDIF
C*
C* ** Open the file
C*
C          EVAL          FILED = open(PATH: OFLAGW)
C*
C* ** Check if open worked
C*
C          FILED          IFEQ          -1
C*
C* ** Open failed, send an error message
C*
C          MOVE          MSG(5)        MSGTEXT
C          EXSR          SNDMSG
C*

```

```

C          ELSE
C*
C*      ** Open worked, write certificate out to file and close file
C*
C          Z-ADD      4          OUTLEN
C          CALLP     write      (FILED:
C                               CLONEKEKLC:
C                               OUTLEN)
C          CALLP     write      (FILED:
C                               CLONEKEK:
C                               CLONEKEKL)
C          CALLP     write      (FILED:
C                               CLONEINFOLENC:
C                               OUTLEN)
C          CALLP     write      (FILED:
C                               CLONEINFO:
C                               CLONEINFOLEN)
C          CALLP     close      (FILED)
C*
C*      ** Send completion message
C*
C          MOVEL    MSG(6)      MSGTEXT
C          EVAL     %SUBST(MSGTEXT: 32: 12) =
C                  %SUBST(PATH: 1: 12)
C          EXSR     SNDMSG
C          ENDIF
C*
C          SETON                                           LR
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C                  CALL      'QMHSNDPM'
C                  PARM      MESSAGEID
C                  PARM      MESSAGEFILE
C                  PARM      MSGTEXT
C                  PARM      MSGLENGTH
C                  PARM      MSGTYPE
C                  PARM      STACKENTRY
C                  PARM      STACKCOUNTER
C                  PARM      MSGKEY
C                  PARM      ERRCODE
C                  ENDSR
C*

```

 The input file could not be opened.
 There was an error reading from the file.
 The length of the certificate is not valid.
 CSUAMKD failed with return/reason codes 9999/9999.
 The output file could not be opened.
 The share was written to file

Example: ILE C program for installing a master key share: Change this program example to suit your needs for installing a master key share.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* PUTSHARE */
/*
/* Sample program to install a master key share as part of the
/* master key cloning process.
/*
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999, 1999
/*
/* This material contains programming source code for your

```

```

/* consideration. These examples have not been thoroughly      */
/* tested under all conditions. IBM, therefore, cannot         */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are         */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF         */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE   */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services */
/* these programs and files.                                   */
/*                                                            */
/* Note: Input format is more fully described in Chapter 2 of  */
/*       IBM CCA Basic Services Reference and Guide             */
/*       (SC31-8609) publication.                               */
/*                                                            */
/* Parameters: Share number                                   */
/*              Name of share receiver private key             */
/*              Name of certifying key                         */
/*              Stream file containing sender certificate       */
/*                                                            */
/* Example:                                                  */
/* CALL PGM(PUTSHARE) PARM(2 RECVR SAKEY SNDR.PUB)            */
/*                                                            */
/* Note: This program assumes the card with the profile is     */
/*       already identified either by defaulting to the CRP01  */
/*       device or by being explicitly named using the         */
/*       Cryptographic_Resource_Allocate verb. Also this      */
/*       device must be varied on and you must be authorized  */
/*       to use this device description.                       */
/*                                                            */
/* The Common Cryptographic Architecture (CCA) verbs used is  */
/* Master_Key_Distribution (CSUAMKD).                         */
/*                                                            */
/* Use these commands to compile this program on iSeries:     */
/* ADDLIB LIB(QCCA)                                           */
/* CRTCMOD MODULE(PUTSHARE) SRCFILE(SAMPLE)                   */
/* CRTPGM PGM(PUTSHARE) MODULE(PUTSHARE)                     */
/*       BNDDIR(QCCA/QC6BNDDIR)                               */
/*                                                            */
/* Note: Authority to the CSUAMKD service program             */
/*       in the QCCA library is assumed.                      */
/*                                                            */
/*-----*/
#include <stdio.h>
#include <string.h>
#include "csucincl.h"
#include "decimal.h"

extern void QDCXLATE(decimal(5,0), char *, char*, char *);
#pragma linkage (QDCXLATE, OS, nowiden)

int main(int argc, char *argv[])
{
/*-----*/
/* Declares for CCA parameters                               */
/*-----*/
long return_code = 0;
long reason_code = 0;
long exit_data_length = 0;
char exit_data[4];
char rule_array[24];
long rule_array_count;
long token_len = 2500;
char token[2500];
long cloneInfoKeyLength = 500;
unsigned char cloneInfoKey[500];

```



```

long cloneInfoLength = 400;
unsigned char cloneInfo[400];
long shareIdx;
char name[64];
char SAname[64];
/*-----*/
/* Declares for working with a PKA token */
/*-----*/
long pub_sec_len; /* Public section length */
long prv_sec_len; /* Private section length */
long cert_sec_len; /* Certificate section length */
long info_subsec_len; /* Information subsection length */
long offset; /* Offset into token */
long tempOffset; /* (Another) Offset into token */
long tempLength; /* Length variable */
long tempLen1, tempLen2; /* temporary length variables */

char cloneShare[] = "cloneShare00"; /* Base cloning share filename */
long count; /* Number of bytes read in from file */
decimal(15,5) shareParm; /* Packed 15 5 var used for converting */
/* from packed 15 5 to binary. Numeric */
/* parms on iSeries are passed as dec 15 5*/
FILE *fp; /* File pointer */

if (argc < 5) /* Check number of parameters passed in */
{
printf("Need Share index, Receiver name, SA name, and cert\n");
return 1;
}

/* Convert the packed decimal 15 5 parm */
/* to binary. */
memcpy(&shareParm,argv[1],sizeof(shareParm));
shareIdx = shareParm;
memset(name,' ',64); /* Copy the Private key name parm to a */
memcpy(name,argv[2],strlen(argv[2])); /* 64 byte space padded var. */
memset(SAname,' ',64); /* Copy the Share Admin name parm to a */
memcpy(SAname,argv[3],strlen(argv[3]));/* 64 byte space padded var. */

fp = fopen(argv[4],"rb"); /* Open the file containing the token */
if (!fp)
{
printf("File %s not found.\n",argv[4]);
return 1;
}

memset(token,0,2500); /* Read the token from the file */
count = fread(token,1,2500,fp);

fclose(fp); /* Close the file */

/* Determine length of token from length */
/* bytes at offset 2 and 3. */
token_len = ((256 * token[2]) + token[3]);
if (count < token_len) /* Check if whole token was read in */
{
printf("Incomplete token in file\n");
return 1;
}

/*****/
/* Find the certificate offset in the token */
/* */
/* The layout of the token is */
/* */
/* - Token header - 8 bytes - including 2 length bytes */
/* - Public key section - length bytes at offset 10 overall */

```

```

/* - Private key name - 68 bytes          */
/* - Certificate section                  */
/*                                       */
/*****/
pub_sec_len = ((256 * token[10]) + token[11]);

offset = pub_sec_len + 68 + 8; /* Set offset to certificate section */

/* Determine certificate section          */
/* length from the length bytes at       */
/* offset 2 of the section.              */
cert_sec_len = ((256 * token[offset + 2]) + token[offset + 3]);

/*****/
/* Open and read the clone file          */
/*****/
/* Build path name from the base        */
/* file name and the index              */
if (shareIdx < 9) cloneShare[11] = '0' + shareIdx;
else
{
    cloneShare[10] = '1';
    cloneShare[11] = '0' + shareIdx - 10;
}

fp = fopen(cloneShare,"rb"); /* Open the file with the share */
if (!fp)
{
    printf("Clone share file %s not found.\n",cloneShare);
    return 1;
}

/* Read in the length of the KEK        */
count = fread((char*)&cloneInfoKeyLength,1,4,fp);

if (count < 4) /* Check if there was an error */
{
    printf("Clone share file %s contains invalid data.\n",
        cloneShare);
    fclose(fp);
    return 1;
}

/* Read in the Key encrypting key        */
count = fread((char*)cloneInfoKey,1,cloneInfoKeyLength,fp);

if (count < cloneInfoKeyLength) /* Check for an error reading */
{
    printf("Clone share file %s contains invalid data.\n",
        cloneShare);
    fclose(fp);
    return 1;
}

/* Read in the length of the clone info */
count = fread((char*)&cloneInfoLength,1,4,fp);

if (count < 4) /* Check for an error */
{
    printf("Clone share file %s contains invalid data.\n",
        cloneShare);
    fclose(fp);
    return 1;
}

/* Read in the clone info                */
count = fread((char*)cloneInfo,1,cloneInfoLength,fp);

```

```

if (count < cloneInfoLength) /* Check for an error */
{
    printf("Clone share file %s contains invalid data.\n",
           cloneShare);
    fclose(fp);
    return 1;
}

fclose(fp); /* Close the file */

/*****
/* Install the share */
*****/
memcpy((void*)rule_array,"INSTALL ",8); /* Set rule array */
rule_array_count = 1;

CSUAMKD( &return_code, &reason_code, &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char*)rule_array,
        &shareIdx,
        name,
        SAname,
        &cert_sec_len,
        &token[offset],
        &cloneInfoKeyLength,
        cloneInfoKey,
        &cloneInfoLength,
        cloneInfo);

if (return_code > 4 )
{
    printf("Master Key Distribution Failed : return reason %d/%d\n",
           return_code, reason_code);
    return 1;
}
else
{
    printf("Master Key share %d successfully installed.\n",shareIdx);
    printf("Return reason codes %d/%d\n",return_code, reason_code);
    return 0;
}
}

```

Example: ILE RPG program for installing a master key share: Change this program example to suit your needs for installing a master key share.

Note: Read the "Code disclaimer information" on page 276 for important legal information.

```

D*****
D* PUTSHARE
D*
D* Sample program to install a master key share as part of
D* the master key cloning process.
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for

```

```

D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D*   IBM CCA Basic Services Reference and Guide
D*   (SC31-8609) publication.
D*
D* Parameters: Share number
D*             Name of share receiver private key
D*             Name of certifying key
D*             Path name of stream file containing sender certificate
D*
D* Example:
D*   CALL PGM(PUTSHARE) PARM(2 RECVR SAKEY SENDER.PUB)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(PUTSHARE) SRCFILE(SAMPLE)
D* CRTPGM   PGM(PUTSHARE) MODULE(PUTSHARE)
D*         BNDDIR(QCCA/QC6BNDDIR)
D*
D* Note: Authority to the CSUAMKD service program
D*       in the QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used is
D* Master_Key_Distribution (CSUAMKD).
D*
D*****
D*-----
D* Declare variables used by CCA SAPI calls
D*-----
D*           ** Return code
DRETURNCODE S           9B 0
D*           ** Reason code
DREASONCODE S           9B 0
D*           ** Exit data length
DEXITDATALEN S          9B 0
D*           ** Exit data
DEXITDATA   S            4
D*           ** Rule array count
DRULEARRAYCNT S         9B 0
D*           ** Rule array
DRULEARRAY  S            16
D*           ** Token length
DTOKENLEN  S           9B 0 INZ(2500)
D*           ** Token and array for subscripting
DTOKEN     DS            2500
DTOKENARRAY          1   DIM(2500)
D*           ** Private key name
DPRVNAME   S             64
D*           ** Certifying key name
DCERTKEY   S             64
D*
DLSTRUCT   DS
D*           ** Clone KEK length - one is binary form and the
D*           ** other is used for reading the value from a file
DCLONEKEKL          9B 0 INZ(500)
DCLONEKEKLC         1     4
D*           ** Clone info length - one is binary form and the
D*           ** other is used for reading the value from a file
DCLONEINFOLEN      9B 0 INZ(400)
DCLONEINFOLENC     5     8
D*           ** Cloning key-encrypting-key
DCLONEKEK          S            500
D*           ** Cloning info
DCLONEINFO         S            400
D*           ** Share index
DSHAREIDX         S            9B 0

```

```

D*          ** Data structure for aligning 2 bytes into
D*          ** a 2 bytes integer
DLENSTRUCT DS          2
DMSB          1      1
DLSB          2      2
DLENGTH       1      2B 0
D*          ** Certificate section length
DCRTSECLLEN  S          9B 0
D*          ** Public key section length
DPUBSECLLEN  S          9B 0
D*          ** Index into Token array
DTKNINDEX    S          9B 0
D*          ** Number of bytes to read from a file
DINLEN       S          9B 0
D*          ** File descriptor
DFILED       S          9B 0
D*          ** File path and length
DPSTRUCT     DS
DPATH          80      INZ(*ALLX'00')
DSIDX          11      12B 0
DPATHLEN      S          9B 0
D*          ** Open Flag - Open for Read only
DOFLAGR      S          10I 0 INZ(1)
D*          ** Base name of file to store cloning share
DSHAREFILE   S          12      INZ('cloneShare00')
D*
D*****
D* Prototype for Master_Key_Distribution (CSUAMKD)
D*****
DCSUAMKD      PR
DRETCOD          9B 0
DRSNCOD          9B 0
DEXTDTALN       9B 0
DEXTDT           4
DRARRYCT        9B 0
DRARRY           16
DSHRINDX         9B 0
DKYNAM           64
DCRTKYNAM        64
DCRTL            9B 0
DCRT             2500  OPTIONS(*VARSIZE)
DCLNKEKL         9B 0
DCLNKEK         1200  OPTIONS(*VARSIZE)
DCLNL            9B 0
DCLN             400   OPTIONS(*VARSIZE)
D*
D*****
D* Prototype for open()
D*****
D*   value returned = file descriptor (OK), -1 (error)
Dopen          PR          9B 0 EXTPROC('open')
D*   path name of file to be opened.
D              128      OPTIONS(*VARSIZE)
D*   Open flags
D              9B 0 VALUE
D*   (OPTIONAL) mode - access rights
D              10U 0 VALUE OPTIONS(*NOPASS)
D*   (OPTIONAL) codepage
D              10U 0 VALUE OPTIONS(*NOPASS)
D*
D*****
D* Prototype for read()
D*****
D*   value returned = number of bytes actually read, or -1
Dread          PR          9B 0 EXTPROC('read')
D*   File descriptor returned from open()
D              9B 0 VALUE

```

```

D*   Input buffer
D           2500   OPTIONS(*VARSIZE)
D*   Length of data to be read
D           9B 0 VALUE
D*
D*****
D* Prototype for close()
D*****
D*   value returned = 0 (OK), or -1
Dclose      PR           9B 0 EXTPROC('close')
D*   File descriptor returned from open()
D           9B 0 VALUE
D*
D*-----
D*           ** Declares for sending messages to the
D*           ** job log using the QMHSNDPM API
D*-----
DMSG        S           75   DIM(7) CTDATA PERRCD(1)
D           DS
DMSGTEXT    1           80
DSAPI       1           7
DFAILRETC   41         44
DFAILRSNC   46         49
DMSGLENGTH  S           9B 0 INZ(80)
DMESSAGEID  S           7   INZ('      ')
DMESSAGEFILE S          21  INZ('          ')
DMSGKEY      S           4   INZ('      ')
DMSGTYPE     S           10  INZ('*INFO  ')
DSTACKENTRY S           10  INZ('*      ')
DSTACKCOUNTER S          9B 0 INZ(2)
DERRCODE     DS
DBYTESIN     1           4B 0 INZ(0)
DBYTESOUT    5           8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* *
C   *ENTRY      PLIST
C               PARM                SINDEXT 15 5
C               PARM                PRVKEY  32
C               PARM                SAKEY   32
C               PARM                FILEPARM 32
C*****
C* Open certificate file
C*****
C* *-----*
C* ** Build path name *
C* *-----*
C           EVAL      PATHLEN = %LEN(%TRIM(FILEPARM))
C   PATHLEN  SUBST    FILEPARM:1  PATH
C* *-----*
C* * Open the file *
C* *-----*
C           EVAL      FILED = open(PATH: OFLAGR)
C* *-----*
C* * Check if open worked *
C* *-----*
C   FILED      IFEQ    -1
C* *-----*
C* * Open failed, send an error message *
C* *-----*
C           MOVEL     MSG(1)      MSGTEXT
C           EXSR      SNDMSG
C           RETURN
C*
C           ENDIF
C* *-----*

```

```

C*      * Open worked, read certificate from file and close file *
C*      *-----*
C              EVAL      TOKENLEN = read(FILED: TOKEN: TOKENLEN)
C              CALLP      close          (FILED)
C*
C*      *-----*
C*      * Check if read operation was OK *
C*      *-----*
C  TOKENLEN      IFEQ      -1
C              MOVEL      MSG(2)      MSGTEXT
C              EXSR      SNDMSG
C              ENDIF
C*
C*      *-----*
C*      * Check if certificate length is valid *
C*      * The length bytes start at position 3 *
C*      *-----*
C              EVAL      MSB = TOKENARRAY(3)
C              EVAL      LSB = TOKENARRAY(4)
C  LENGTH      IFLT      TOKENLEN
C*
C*      *-----*
C*      * Certificate length is not valid *
C*      *-----*
C              MOVEL      MSG(3)      MSGTEXT
C              EXSR      SNDMSG
C              RETURN
C              ENDIF
C*
C*****
C* Find the certificate in the token
C*
C* The layout of the token is
C*
C* - Token header - 8 bytes - including 2 length bytes
C* - Public key section - length bytes at position 2 (11 overall)
C* - Private key name - 68 bytes
C* - Certificate section
C*
C* Note: 1 is added because RPG arrays start at 1.
C*****
C              EVAL      MSB = TOKENARRAY(11)
C              EVAL      LSB = TOKENARRAY(12)
C              EVAL      PUBSECLN = LENGTH
C              EVAL      TKNINDEX = PUBSECLN + 68 + 8 + 1
C*
C*      *-----*
C*      * Determine length of certificate section *
C*      * Length bytes are at position 2 of the *
C*      * section. *
C*      *-----*
C              EVAL      MSB = TOKENARRAY(TKNINDEX + 2)
C              EVAL      LSB = TOKENARRAY(TKNINDEX + 3)
C              EVAL      CRTSECLN = LENGTH
C*
C*****
C* Open and read the clone file
C*****
C*      *-----*
C*      * Set share index number *
C*      * (Convert from packed 15 5 to binary) *
C*      *-----*
C              Z-ADD      SINDEK      SHAREIDX
C*      ** Build path name
C              MOVEL      *ALLX'00'      PATH
C              MOVEL      SHAREFILE      PATH
C*      ** Adjust two digits on file name by adding to their
C*      ** character value

```

```

C   SIDX      ADD      SHAREIDX   SIDX
C*   ** If the index is greater than or equal to 10
C*   ** then add 246 to force the first character to change
C   SHAREIDX  IFGE     10
C   SIDX      ADD      246        SIDX
C
C*   ENDIF
C*
C*   ** Open the file
C*
C           EVAL      FILED = open(PATH: OFLAGR)
C*
C*   ** Check if open worked
C*
C   FILED     IFEQ     -1
C*
C*   ** Open failed, send an error message
C*
C           MOVEL     MSG(4)      MSGTEXT
C           EXSR      SNDMSG
C*
C           ELSE
C*
C*   ** Open worked, read in the clone information and close file
C*
C           SETON                                01
C           Z-ADD     4          INLEN
C           EVAL      INLEN = read(FILED: CLONEKEKLC: INLEN)
C*
C*   *-----*
C*   * Check if read operation was OK *
C*   *-----*
C   INLEN     IFNE     4
C           MOVEL     MSG(5)      MSGTEXT
C           EXSR      SNDMSG
C           SETOFF                                01
C           ENDIF
C*
C   01        EVAL      INLEN = read(FILED: CLONEKEK: CLONEKEKL)
C*
C   01INLEN   IFNE     CLONEKEKL
C           MOVEL     MSG(5)      MSGTEXT
C           EXSR      SNDMSG
C           SETOFF                                01
C           ENDIF
C*
C   01        Z-ADD     4          INLEN
C   01        EVAL      INLEN = read(FILED: CLONEINFOLENC: INLEN)
C*
C*   *-----*
C*   * Check if read operation was OK *
C*   *-----*
C   01INLEN   IFNE     4
C           MOVEL     MSG(5)      MSGTEXT
C           EXSR      SNDMSG
C           SETOFF                                01
C           ENDIF
C*
C   01        EVAL      INLEN = read(FILED: CLONEINFO: CLONEINFOLEN)
C*
C*   *-----*
C*   * Check if read operation was OK *
C*   *-----*
C   01INLEN   IFNE     CLONEINFOLEN
C           MOVEL     MSG(5)      MSGTEXT
C           EXSR      SNDMSG
C           SETOFF                                01
C           ENDIF

```



```

C*
C          CALLP      close      (FILED)
C N01          SETON
C*
C*****
C* Obtain a certificate
C*****
C* -----*
C* * Set share index number *
C* -----*
C          Z-ADD      SINDEXT      SHAREIDX
C* -----*
C* * Set private key name *
C* -----*
C          EVAL      LENGTH = %LEN(%TRIM(PRVKEY))
C          LENGTH    SUBST      PRVKEY:1      PRVNAME
C* -----*
C* * Set certifying key name *
C* -----*
C          EVAL      LENGTH = %LEN(%TRIM(SAKEY))
C          LENGTH    SUBST      SAKEY:1      CERTKEY
C* -----*
C* * Set the keywords in the rule array *
C* -----*
C          MOVE      'INSTALL '      RULEARRAY
C          Z-ADD      1                RULEARRAYCNT
C* -----*
C* * Call Master Key Distribution SAPI *
C* -----*
C          CALLP      CSUAMKD      (RETURNCODE:
C                                     REASONCODE:
C                                     EXITDATALEN:
C                                     EXITDATA:
C                                     RULEARRAYCNT:
C                                     RULEARRAY:
C                                     SHAREIDX:
C                                     PRVNAME:
C                                     CERTKEY:
C                                     CRTSECLN:
C                                     TOKENARRAY(TKNINDEX):
C                                     CLONEKEKL:
C                                     CLONEKEK:
C                                     CLONEINFOLEN:
C                                     CLONEINFO)
C* -----*
C* * Check the return code *
C* -----*
C          RETURNCODE  IFGT      4
C* -----*
C* * Send failure message *
C* -----*
C          MOVE      MSG(6)      MSGTEXT
C          MOVE      RETURNCODE  FAILRETC
C          MOVE      REASONCODE  FAILRSNC
C          MOVE      'CSUAMKD'    SAPI
C          EXSR      SNDMSG
C          RETURN
C          ENDIF
C* -----*
C* * Send success message *
C* -----*
C          MOVE      MSG(7)      MSGTEXT
C          EVAL      %SUBST(MSGTEXT: 32: 12) =
C                                     %SUBST(PATH: 1: 12)
C          EXSR      SNDMSG
C          ENDIF
C*

```

```

C                SETON                LR
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C                CALL      'QMHSNDPM'
C                PARM      MESSAGEID
C                PARM      MESSAGEFILE
C                PARM      MSGTEXT
C                PARM      MSGLENGTH
C                PARM      MSGTYPE
C                PARM      STACKENTRY
C                PARM      STACKCOUNTER
C                PARM      MSGKEY
C                PARM      ERRCODE
C                ENDSR
C*

```

```

**
The certificate file could not be opened.
There was an error reading from the certificate file.
The length of the certificate is not valid.
The clone share file could not be opened.
The clone share file either could not be read or has invalid data.
CSUAMKD failed with return/reason codes 9999/9999.
The share was successfully installed.

```

Example: ILE C program for listing retained keys: Change this program example to suit your needs for listing retained keys.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* List the names of the RSA private keys retained.      */
/*                                                       */
/*                                                       */
/*                                                       */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000         */
/*                                                       */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot   */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are   */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF   */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files.                             */
/*                                                       */
/*                                                       */
/* Note: Input format is more fully described in Chapter 2 of */
/*       IBM CCA Basic Services Reference and Guide          */
/*       (SC31-8609) publication.                          */
/*                                                       */
/* Parameters:                                           */
/* none.                                                */
/*                                                       */
/* Example:                                             */
/* CALL PGM(LISTRETAIN)                                */
/*                                                       */
/* Note: This program assumes the card with the profile is */
/* already identified either by defaulting to the CRP01   */
/* device or by being explicitly named using the         */
/* Cryptographic_Resource_Allocate verb. Also this     */
/* device must be varied on and you must be authorized  */
/* to use this device description.                      */

```

```

/*                                                                    */
/* The Common Cryptographic Architecture (CCA) verb used is          */
/* Access_Control_Initialization (CSUAACI).                          */
/*                                                                    */
/* Use these commands to compile this program on iSeries:          */
/* ADDLIB LIB(QCCA)                                                 */
/* CRTCMOD MODULE(LISTRETAIN) SRCFILE(SAMPLE)                       */
/* CRTPGM PGM(LISTRETAIN) MODULE(LISTRETAIN)                       */
/*     BNDSRVPGM(QCCA/CSNDRKL)                                     */
/*                                                                    */
/* Note: Authority to the CSNDRKL service program in the           */
/*     QCCA library is assumed.                                     */
/*                                                                    */
/* The Common Cryptographic Architecture (CCA) verb used is          */
/* Retained_Key_List (CSNDRKL).                                     */
/*                                                                    */
/*-----*/
#include <string.h>
#include <stdio.h>
#include "csucincl.h"

void main(void)
{
/*-----*/
/* standard CCA parameters                                         */
/*-----*/
long      return_code;
long      reason_code;
long      exit_data_length;
unsigned char exit_data[2];
long      rule_array_count;
unsigned char rule_array[2][8];
/*-----*/
/* CCA parameters unique to CSNDRKL                                */
/*-----*/
unsigned char key_label_mask[64];
unsigned char key_label[500][64];
long      retain_key_count;
long      key_label_count = 500;
int       k;

/*-----*/
/* Set up label mask, ie. which key name to retrieve.             */
/* *.*.*.*.*.* is a wildcard for all keys.                       */
/*-----*/
memset(key_label, 0x00, sizeof(key_label) );
memset(key_label_mask, ' ', sizeof(key_label_mask));
memcpy(key_label_mask, "*.*.*.*.*.*", 13);
rule_array_count = 0;

/*-----*/
/* Invoke the verb to get the list of the retained keys.          */
/*-----*/
CSNDRKL(&return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char*)rule_array,
        key_label_mask,
        &retain_key_count,
        &key_label_count,
        (unsigned char*)key_label);

/*-----*/
/* Check the results                                               */
/*-----*/
}

```

```

if (return_code != 0)
{
    printf("Retained Key List failed with return/reason %d/%d \n",
        return_code, reason_code);
    return;
}
else
{
    /*-----*/
    /* Display number of keys retained/returned.          */
    /*-----*/
    printf("Retained key count [%d]\n",retain_key_count);
    printf( "No. of key labels returned [%d]\n",key_label_count);
    if (key_label_count > 0)
    {
        /*-----*/
        /* Display the names of each key returned.          */
        /*-----*/
        printf("Retain list = \n" );
        for (k = 0 ;k < key_label_count; k++)
        {
            printf( "[%.64s]\n",key_label[k]);
        }
    }
}
}
}

```

Example: ILE RPG program for listing retained keys: Change this program example to suit your needs for listing retained keys.

Note: Read the "Code disclaimer information" on page 276 for important legal information.

```

D*****
D*
D* List the names of the RSA private keys retained within the
D* .
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters: None
D*
D* Example:
D* CALL PGM(LISTRETAIN)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(LISTRETAIN) SRCFILE(SAMPLE)
D* CRTPGM PGM(LISTRETAIN) MODULE(LISTRETAIN)
D* BNDSRVPGM(QCCA/CSNDRKL)
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Retained_key_List (CSNDRKL)

```

```

D*
D* Note: Authority to the CSNDRKL service program in the
D*       QCCA library is assumed.
D*
D*
D* Note: This program assumes the card with the profile is
D*       already identified either by defaulting to the CRP01
D*       device or by being explicitly named using the
D*       Cryptographic_Resource_Allocate verb. Also this
D*       device must be varied on and you must be authorized
D*       to use this device description.
D*
D*****
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*
D*       ** Return code
DRETURNCODE      S           9B 0
D*
D*       ** Reason code
DREASONCODE      S           9B 0
D*
D*       ** Exit data length
DEXITDATALEN     S           9B 0
D*
D*       ** Exit data
DEXITDATA        S            4
D*
D*       ** Rule array count
DRULEARRAYCNT    S           9B 0
D*
D*       ** Rule array
DRULEARRAY       S            16
D*
D*       ** Key label mask
DKEYLBLMASK      S            64
D*
D*       ** Key count
DKEYCOUNT       S           9B 0
D*
D*       ** Label count
DLABELCOUNT    S           9B 0
D*
D*       ** Label list and label array
DLABELLIST       DS          3200
DLABELS          S            64   DIM(50)
D*
D*       ** Loop counter
DI                S           9B 0
D*
D*****
D* Prototype for Retained_Key_List
D*****
DCSNDRKL         PR
DRETCODE         S           9B 0
DRSNCODE         S           9B 0
DEXTDTALEN      S           9B 0
DEXTDTA          S            4
DRARRAYCT       S           9B 0
DRARRAY          S            16
DKYLBLMSK       S            64
DKYCOUNT        S           9B 0
DLBLCOUNT        S           9B 0
DLBLS           S            64
D*
D*-----
D*       ** Declares for sending messages to the
D*       ** job log using the QMHSDPM API
D*-----
DMSG             S           75   DIM(4) CTDATA PERRCD(1)
DMSGLENGTH       S           9B 0 INZ(75)
D                DS
DMSGTEXT         S            1    75
DNUMKEYS         S            1    3
DNUMLABELS       S           25   26
DDSPLBL         S            2    65
DFAILRET        S           41   44

```

```

DFAILRSNC          46      49
DMESSAGEID        S          7      INZ('      ')
DMESSAGEFILE      S          21     INZ('      ')
DMSGKEY           S          4      INZ('      ')
DMSGTYPE          S          10     INZ('*INFO ')
DSTACKENTRY       S          10     INZ('*      ')
DSTACKCOUNTER     S          9B 0   INZ(2)
DERRCODE          DS
DBYTESIN          1          4B 0   INZ(0)
DBYTESOUT         5          8B 0   INZ(0)
D*
C*****
C* START OF PROGRAM *
C* *
C*-----*
C* No rule array keywords *
C*-----*
C          Z-ADD      0          RULEARRAYCNT
C*-----*
C* Get up to 50 labels *
C*-----*
C          Z-ADD      50         LABELCOUNT
C*-----*
C* Set the mask to everything *
C*-----*
C          MOVE      '*'         KEYLBLMASK
C*-----*
C* Call Retained Key List SAPI *
C*-----*
C          CALLP     CSNDRKL      (RETURNCODE:
C                                REASONCODE:
C                                EXITDATALEN:
C                                EXITDATA:
C                                RULEARRAYCNT:
C                                RULEARRAY:
C                                KEYLBLMASK:
C                                KEYCOUNT:
C                                LABELCOUNT:
C                                LABELLIST)
C*-----*
C* Check the return code *
C*-----*
C          RETURNCODE  IFGT      4
C*          *-----*
C*          * Send error message *
C*          *-----*
C          MOVE      MSG(1)      MSGTEXT
C          MOVE      RETURNCODE  FAILRETC
C          MOVE      REASONCODE  FAILRSNC
C          EXSR      SNDMSG
C*
C          ELSE
C*
C* *-----*
C* * Check number of keys *
C* *-----*
C          LABELCOUNT  IFEQ      0
C*          *-----*
C*          * Send message saying there are no keys *
C*          *-----*
C          MOVE      MSG(2)      MSGTEXT
C          EXSR      SNDMSG
C*
C          ELSE
C*
C*          *-----*
C*          * Send message with number of keys *

```

```

C*          *-----*
C          MOVE      MSG(3)      MSGTEXT
C          MOVE      KEYCOUNT    NUMKEYS
C          MOVE      LABELCOUNT  NUMLABELS
C          EXSR      SNDMSG
C*
C*          *-----*
C*          * Display each key label up to 50 *
C*          *-----*
C          MOVE      MSG(4)      MSGTEXT
C          FOR        I=1 BY 1 TO LABELCOUNT
C          MOVE      LABELS(I)    DSPLBL
C          EXSR      SNDMSG
C          ENDFOR
C*
C          ENDIF
C          ENDIF
C*
C          SETON                                     LR
C*
C*****
C* Subroutine to send a message
C*****
C  SNDMSG      BEGSR
C              CALL      'QMHSNDPM'
C              PARM      MESSAGEID
C              PARM      MESSAGEFILE
C              PARM      MSGTEXT
C              PARM      MSGLENGTH
C              PARM      MSGTYPE
C              PARM      STACKENTRY
C              PARM      STACKCOUNTER
C              PARM      MSGKEY
C              PARM      ERRCODE
C              ENDSR

```

```

**
CSNDRKL failed with return/reason codes 9999/9999
There are no retained keys
000 keys were found and 00 labels returned
[

```

Example: ILE C program for deleting retained keys: Change this program example to suit your needs for deleting retained keys.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

/*-----*/
/* Delete a retained key */
/* */
/* */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000 */
/* */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files. */
/* */
/* */
/* Note: Input format is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide */
/* (SC31-8609) publication. */
/* */

```

```

/* Parameters: */
/* none. */
/* */
/* Example: */
/* CALL PGM(DLTRTNKEY) (SSLPRIV.KEY.ONE) */
/* */
/* */
/* Note: This program assumes the card with the profile is */
/* already identified either by defaulting to the CRP01 */
/* device or by being explicitly named using the */
/* Cryptographic_Resource_Allocate verb. Also this */
/* device must be varied on and you must be authorized */
/* to use this device description. */
/* */
/* The Common Cryptographic Architecture (CCA) verb used is */
/* Retained_Key_Delete (CSNDRKD). */
/* */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA) */
/* CRTCMOD MODULE(DLTRTNKEY) SRCFILE(SAMPLE) */
/* CRTPGM PGM(DLTRTNKEY) MODULE(DLTRTNKEY) */
/* BNDSRVPGM(QCCA/CSNDRKD) */
/* */
/* Note: Authority to the CSNDRKD service program in the */
/* QCCA library is assumed. */
/* */
/*-----*/
#include <string.h>
#include <stdio.h>
#include "csucincl.h"

/*-----*/
/* standard return codes */
/*-----*/

#define OK 0
#define WARNING 4

void main(int argc, char * argv[1])
{
/*-----*/
/* standard CCA parameters */
/*-----*/
long return_code;
long reason_code;
long exit_data_length;
unsigned char exit_data[2];
long rule_array_count = 0;
unsigned char rule_array[1][8];
unsigned char key_label[64];

/*-----*/
/* Process the parameters */
/*-----*/
if (argc < 1)
{
printf("Key label parameter must be specified.\n");
return;
}

/*-----*/
/* Set up the key label */
/*-----*/
memset(key_label, ' ', 64 );

```



```

memcpy(key_label, argv[1], strlen(argv[1]) );

/*-----*/
/* Call the Retained Key List SAPI */
/*-----*/
CSNDRKD(&return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (unsigned char*)rule_array,
        key_label);

/*-----*/
/* Check the return code and display the results */
/*-----*/
if ( (return_code == OK) || (return_code == WARNING) )
{
    printf("Request was successful\n");
    return;
}
else
{
    printf("Request failed with return/reason codes: %d/%d \n",
        return_code, reason_code);
    return;
}

}

```

Example: ILE RPG program for deleting retained keys: Change this program example to suit your needs for deleting retained keys.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* DLTRTNKEY
D*
D* Sample program to delete a retained key
D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters:
D* Retained key label name
D* (64 characters - pad with blanks on the right)
D*
D* Example:
D*
D* CALL DLTRTNKEY +

```

```

D* 'PKA.RETAINED.KEY.123
D*
D* Use these commands to compile this program on iSeries:
D* CRTTRPGMOD MODULE(DLTRTNKEY) SRCFILE(SAMPLE)
D* CRTPGM PGM(DLTRTNKEY) MODULE(DLTRTNKEY)
D*         BNDSRVPGM(QCCA/CSNDRKD)
D*
D* Note: Authority to the CSNDRKD service program in the
D*       QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Retained_Key_Delete (CSNDRKD)
D*
D*****
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*         ** Return code
DRETURNCODE      S              9B 0
D*         ** Reason code
DREASONCODE      S              9B 0
D*         ** Exit data length
DEXITDATALEN     S              9B 0
D*         ** Exit data
DEXITDATA        S              4
D*         ** Rule array count
DRULEARRAYCNT    S              9B 0
D*         ** Rule array
DRULEARRAY       S              16
D*         ** Retained key label
DKEYNAME         S              64
D*
D*****
D* Prototype for Retained_Key_Delete (CSNDRKD)
D*****
DCSNDRKD         PR
DRETCODE         S              9B 0
DRSNCODE         S              9B 0
DEXTDTALEN       S              9B 0
DEXTDTA          S              4
DRARRAYCT        S              9B 0
DRARRAY          S              16
DKEYNAM          S              64
D*
D*-----
D*         ** Declares for sending messages to the
D*         ** job log using the QMHSNDPM API
D*-----
DMSG             S              75   DIM(2) CTDATA PERRCD(1)
DMSGLENGTH      S              9B 0 INZ(75)
D               DS
DMSGTEXT         S              1    75
DFAILMSGTEXT     S              1    50
DFAILRETC        S              41   44
DFAILRSNC        S              46   49
DMESSAGEID       S              7    INZ(' ')
DMESSAGEFILE     S              21   INZ(' ')
DMSGKEY          S              4    INZ(' ')
DMSGTYPE         S              10   INZ('*INFO ')
DSTACKENTRY      S              10   INZ('* ')
DSTACKCOUNTER    S              9B 0 INZ(2)
DERRCODE         DS
DBYTESIN         S              1    4B 0 INZ(0)
DBYTESOUT        S              5    8B 0 INZ(0)
D*
C*****
C* START OF PROGRAM
*

```

```

C*                                     *
C   *ENTRY          PLIST
C                   PARM                KEYNAME
C*                                     *
C*-----*
C* Set the keywords in the rule array *
C*-----*
C                   Z-ADD      0          RULEARRAYCNT
C*-----*
C* Call Retained Key Delete SAPI *
C*-----*
C                   CALLP      CSNDRKD    (RETURNCODE:
C                                           REASONCODE:
C                                           EXITDATALEN:
C                                           EXITDATA:
C                                           RULEARRAYCNT:
C                                           RULEARRAY:
C                                           KEYNAME)
C*-----*
C* Check the return code *
C*-----*
C   RETURNCODE     IFGT      4
C*-----*
C*       * Send error message *
C*-----*
C                   MOVE      MSG(1)      MSGTEXT
C                   MOVE      RETURNCODE  FAILRETC
C                   MOVE      REASONCODE  FAILRSNC
C                   EXSR      SNDMSG
C*
C                   ELSE
C*-----*
C*       * Send success message *
C*-----*
C                   MOVE      MSG(2)      MSGTEXT
C                   EXSR      SNDMSG
C*
C                   ENDIF
C*
C                   SETON                                     LR
C*
C*****
C* Subroutine to send a message
C*****
C   SNDMSG          BEGSR
C                   CALL      'QMHSNDPM'
C                   PARM      MESSAGEID
C                   PARM      MESSAGEFILE
C                   PARM      MSGTEXT
C                   PARM      MSGLENGTH
C                   PARM      MSGTYPE
C                   PARM      STACKENTRY
C                   PARM      STACKCOUNTER
C                   PARM      MSGKEY
C                   PARM      ERRCODE
C                   ENDSR
C*

```

**
CSNDRKD failed with return/reason codes 9999/9999'
The request completed successfully

Troubleshoot the Cryptographic Coprocessor

Use the methods below to tackle some of the basic problems that may occur with your Cryptographic Coprocessor. If the troubleshooting information does not address your problem, contact your service representative.

Always assure that you have applied all current PTFs for the relevant products and programs.

Using return codes

The primary method for detecting and troubleshooting problems is by monitoring return codes and reason codes.

- **A return code of 0** indicates successful completion. To provide some additional information, the Cryptographic Coprocessor associates some non-zero reason codes with this return code.
- **A return code of 4** indicates that the application programming interface (API) has completed processing, but an unusual event occurred. It could be related to a problem created by the application program, or it could be a normal occurrence based on data that is supplied to the API.
- **A return code of 8** indicates that the API did not complete successfully. An application programming error most likely caused this.
- **A return code of 12** normally indicates some type of problem in the setup or configuration of your Coprocessor. This code means that the processing of the API did not complete successfully.
- **A return code of 16** normally indicates a severe error in Common Cryptographic Architecture Cryptographic Service Provider (CCA CSP), iSeries licensed internal code, or the Cryptographic Coprocessor licensed internal code. For these types of errors, you should contact your service representative.

You can also troubleshoot problems by analyzing the messages that appear in the job log or in the system operator (QSYSOPR) queue. Generally, any event that sends a message to the job log also returns an associated return code and a reason code to the calling programming. Messages sent to the system operator message, if reporting a severe problem, will normally point to a source of additional information about the problem. Such information is intended for IBM service, and therefore you may not necessarily find them useful for problem determination.

Common errors

You should watch out for these common errors:

- **Did you vary on the device?** You cannot send any requests to your Cryptographic Coprocessor until you vary on the device.
- **Is the Cryptographic Coprocessor finding a device?** If you do not explicitly use the `Cryptographic_Resource_Allocate` API, you must name the cryptographic device CRP01. If you do not name it that, the CCA cannot select any device. Either name the device CRP01 or change your program to use the `Cryptographic_Resource_Allocate` CCA API to select the device.
- **Are you selecting the correct device?** If you have a default device (for example, a device named CRP01) and an additional device, the Cryptographic Coprocessor will select the default device, unless you use `Cryptographic_Resource_Allocate`.
- **Is the Cryptographic Coprocessor finding a key store file?** If you do not explicitly use the `Key_Store_Designate` SAPI, the CCA CSP support will attempt to use the files named on the device description. If you have named no files on the device description, the Cryptographic Coprocessor will not find any files.
- **Have you loaded and set a master key?** The Cryptographic Coprocessor will not complete any cryptographic requests other than those for configuring your Cryptographic Coprocessor, unless you load a master key.
- **Does the Old master key register contain a key?** The Cryptographic Coprocessor cannot re-encrypt keys under the Current master key unless the Old master key register contains a value.
- **Does your default role have authority to use a given hardware command?** If not, you will need to log on by using a profile that uses a role that has the correct authority.
- **Does any role have authority to use a given hardware command?** If your Cryptographic Coprocessor requires the hardware command and you have not authorized a role to use that command, you must reinitialize your Cryptographic Coprocessor. Do this by using either the `Cryptographic_Facility_Control`

API or the Hardware Service Manager that is found in System Service Tools. Using the Cryptographic_Facility_Control API requires that you authorize a role to the hardware command that reinitializes the Cryptographic Coprocessor. If no such role exists, you must use the Hardware Service Manager.

- **Is a function control vector loaded?** Your Cryptographic Coprocessor cannot run any cryptographic operations other than configuration until you load a function control vector.
- **Is one of the Cryptographic Access Provider products installed?** IBM ships the function control vectors with these products.
- **If you are loading a master key, did you begin by clearing out the new master key register?** If your Cryptographic Coprocessor has a partially loaded new master key register, you cannot load the first part of a master key.
- **Did you remember to set the clock in your Coprocessor before removing the authority to do so from the DEFAULT role?** If not, you must reinitialize your Cryptographic Coprocessor by using either the Cryptographic_Facility_Control API or the Hardware Service Manager found in System Service Tools. Using the Cryptographic_Facility_Control API requires that you authorize a role to the hardware command that reinitializes the Cryptographic Coprocessor. If no such role exists, you must use the Hardware Service Manager.
- **Did you set the EID before trying to generate public-private key pairs?** You must set the EID before you can generate RSA keys.
- **Did you correctly initialize the first byte of a null key token to binary 0?** If not, the CCA support may try to use it as a key label. CCA Support will either report it as a bad label format or report that it could find the key record.
- **Do you use the same name for a label in a PKA key store file and a retained PKA key?** If so, your Cryptographic Coprocessor will never find the retained key because the Cryptographic Coprocessor always searches the key store file first.
- **Do you have EBCDIC data in any fields in a skeleton PKA key token?** The Cryptographic Coprocessor specifically checks for ASCII data in a number of the fields and will return an error if it finds EBCDIC data.

For further troubleshooting information, see “Reinitialize the Cryptographic Coprocessor” and “Use the Hardware Service Manager” on page 270.

Reinitialize the Cryptographic Coprocessor

If you set up your Cryptographic Coprocessor incorrectly, you can end up with an unusable configuration with which you cannot perform any cryptographic functions and cannot use any of the APIs to recover. For example, you can configure it such that you have no role authorized to set the master key and no role authorized to change or create new roles or profiles.

You can call the hardware command for reinitializing the card by using the Cryptographic_Facility_Control (CSUACFC) SAPI. Two example programs are provided for your consideration. One of them is written in ILE C, while the other is written in ILE RPG. Both perform the same function.

- “Example: ILE C program for reinitializing the Cryptographic Coprocessor” on page 264
- “Example: ILE RPG program for reinitializing your Cryptographic Coprocessor” on page 266

Note: If you choose to use the program example that is provided, change it to suit your specific needs. For security reasons, IBM recommends that you individualize these program examples rather than using the default values provided.

However, in some cases, there may not be a role that is authorized to any hardware command. In this case, you must reload the Licensed Internal Code by using the function that is provided in Hardware Service Manager in System Service Tools as described in “Use the Hardware Service Manager” on page 270.

Updating the Licensed Internal Code in the Cryptographic Coprocessor

Loading the Licensed Internal Code in your Cryptographic Coprocessor erases the master key, all private keys, and all roles and profiles that are stored in your Cryptographic Coprocessor. Because of this, the server does not automatically load PTFs for the Licensed Internal Code in the Cryptographic Coprocessor, and the PTFs always require action on your part to enable them. Before you load the Licensed Internal Code, take appropriate actions to ensure that you can recover, such as ensuring that you have a hard copy of your master key.

Note: If you randomly generated your master key, you will need to clone that key into a second Cryptographic Coprocessor. If you do not, you will lose all your encrypted keys when you reinitialize your Cryptographic Coprocessor.

Example: ILE C program for reinitializing the Cryptographic Coprocessor: Change this program example to suit your needs for reinitializing your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```
/*-----*/
/* Clear the card (reset to manufactured state).          */
/*                                                         */
/*                                                         */
/* COPYRIGHT 5769-SS1 (C) IBM CORP. 1999                */
/*                                                         */
/* This material contains programming source code for your */
/* consideration. These examples have not been thoroughly */
/* tested under all conditions. IBM, therefore, cannot    */
/* guarantee or imply reliability, serviceability, or function */
/* of these program. All programs contained herein are    */
/* provided to you "AS IS". THE IMPLIED WARRANTIES OF    */
/* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE */
/* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for */
/* these programs and files.                              */
/*                                                         */
/*                                                         */
/* Note: This verb is more fully described in Chapter 2 of */
/* IBM CCA Basic Services Reference and Guide             */
/* (SC31-8609) publication.                               */
/*                                                         */
/* Parameters:                                           */
/* none.                                                 */
/*                                                         */
/* Example:                                             */
/* CALL PGM(REINIT)                                     */
/*                                                         */
/* Note: This program assumes the device to use is      */
/* already identified either by defaulting to the CRP01  */
/* device or by being explicitly named using the        */
/* Cryptographic_Resource_Allocate verb. Also this     */
/* device must be varied on and you must be authorized  */
/* to use this device description.                      */
/*                                                         */
/* Use these commands to compile this program on iSeries: */
/* ADDLIB LIB(QCCA)                                     */
/* CRTCMOD MODULE(REINIT) SRCFILE(SAMPLE)               */
/* CRTPGM PGM(REINIT) MODULE(REINIT) BNDSRVPGM(QCCA/CSUACFC) */
/*                                                         */
/* Note: Authority to the CSUACFC service program in the */
/* QCCA library is assumed.                             */
/*                                                         */
/* The Common Cryptographic Architecture (CCA) verb used is */
/* Cryptographic_Facilities_Control (CSUACFC).          */
/*                                                         */
/*-----*/
```

```

#include "csucincl.h"    /* header file for CCA Cryptographic    */
                        /* Service Provider for iSeries    */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*-----*/
/* standard return codes    */
/*-----*/

#define ERROR    -1
#define OK      0
#define WARNING  4

#define TOKENSIZE 8    /* number of bytes in random token    */

int main(int argc, char *argv[])
{
    /*-----*/
    /* standard CCA parameters    */
    /*-----*/

    long return_code = 0;
    long reason_code = 0;
    long exit_data_length = 2;
    char exit_data[4];
    char rule_array[2][8];
    long rule_array_count = 2;

    /*-----*/
    /* fields unique to this sample program    */
    /*-----*/

    long verb_data_length = TOKENSIZE;
    char verb_data[TOKENSIZE];
    char verb_data2[TOKENSIZE];
    int i;

    /* set keywords in the rule array    */

    memcpy(rule_array,"ADAPTER1RQ-TOKEN",16);

    /* get a random token from the card - returned in verb_data    */

    CSUACFC( &return_code,
            &reason_code,
            &exit_data_length,
            exit_data,
            &rule_array_count,
            (char *)rule_array,
            &verb_data_length,
            (char *)verb_data);

    if ( (return_code == OK) | (return_code == WARNING) )
    {
    printf("Random token was successfully returned.\n");

    printf("Return/reason codes ");

    printf("%ld/%ld\n\n", return_code, reason_code);
    }
}

```

```

/* get the one's complement of token and store in verb_data2. */
/* operate on one byte at a time */

for(i = 0; i < TOKENSIZE; i++)
{
    verb_data2[i] = ~verb_data[i];
}

/* change keyword in rule array */

memcpy(&rule_array[1],"RQ-REINT",8);

/* invoke the verb to reset the card */

CSUACFC( &return_code,
        &reason_code,
        &exit_data_length,
        exit_data,
        &rule_array_count,
        (char *)rule_array,
        &verb_data_length,
        verb_data2);

if ( (return_code == OK) | (return_code == WARNING) )
{
    printf("card successfully cleared/reset.\n");

    printf("Return/reason codes ");

    printf("%ld/%ld\n\n", return_code, reason_code);

    return(OK);
}
else
{
    printf("An error occurred while clearing the card");

    printf("card.\n Return/");

    printf("reason codes %ld/%ld\n\n", return_code, reason_code);

    return(ERROR);
}

else
{
    printf("An error occurred while getting the random token.\n");

    printf("Return/reason codes ");

    printf("%ld/%ld\n\n", return_code, reason_code);

    return(ERROR);
}
}

```

Example: ILE RPG program for reinitializing your Cryptographic Coprocessor: Change this program example to suit your needs for reinitializing your Cryptographic Coprocessor.

Note: Read the “Code disclaimer information” on page 276 for important legal information.

```

D*****
D* REINIT
D*
D* Clear the card (reset to manufactured state).

```



```

D*
D*
D* COPYRIGHT 5769-SS1 (C) IBM CORP. 2000, 2000
D*
D* This material contains programming source code for your
D* consideration. These example has not been thoroughly
D* tested under all conditions. IBM, therefore, cannot
D* guarantee or imply reliability, serviceability, or function
D* of these programs. All programs contained herein are
D* provided to you "AS IS". THE IMPLIED WARRANTIES OF
D* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
D* ARE EXPRESSLY DISCLAIMED. IBM provides no program services for
D* these programs and files.
D*
D*
D* Note: Input format is more fully described in Chapter 2 of
D* IBM CCA Basic Services Reference and Guide
D* (SC31-8609) publication.
D*
D* Parameters:
D* char * new time 16 characters
D*
D* Example:
D* CALL PGM(REINIT)
D*
D* Use these commands to compile this program on iSeries:
D* CRTRPGMOD MODULE(REINIT) SRCFILE(SAMPLE)
D* CRTPGM PGM(REINIT) MODULE(REINIT)
D* BNDSRVPGM(QCCA/CSUACFC)
D*
D* Note: Authority to the CSUACFC service program in the
D* QCCA library is assumed.
D*
D* The Common Cryptographic Architecture (CCA) verbs used are
D* Cryptographic_Facility_Control (CSUACFC)
D*
D*****
D*-----
D* Declare variables for CCA SAPI calls
D*-----
D*          ** Return code
DRETURNCODE S          9B 0
D*          ** Reason code
DREASONCODE S          9B 0
D*          ** Exit data length
DEXITDATALEN S         9B 0
D*          ** Exit data
DEXITDATA S           4
D*          ** Rule array count
DRULEARRAYCNT S        9B 0
D*          ** Rule array
DRULEARRAY S           16
D*          ** Verb data length
DVERBDATALEN S         9B 0
D*          ** Verb data
DVERBDATA S            8
D*
D*-----
D* Declares for calculating one's complement
D*-----
DBUFFER DS
DA1      1      2
DA2      3      4
DA3      5      6
DA4      7      8
D*
DWORKBUFF DS

```

```

DINT4          1      4B 0
DINT2          3      4
D*
D*
D*****
D* Prototype for Cryptographic_Facility_Control (CSUACFC)
D*****
DCSUACFC      PR
DRETCODE      9B 0
DRSNCODE      9B 0
DEXTDTALEN    9B 0
DEXTDTA       4
DRARRAYCT     9B 0
DRARRAY       16
DVRBDTALEN    9B 0
DVRBDTA       8
D*
D*-----
D*          ** Declares for sending messages to the
D*          ** job log using the QMHSNDPM API
D*-----
DMSG          S          75  DIM(3) CTDATA PERRCD(1)
DMSGLENGTH    S          9B 0 INZ(64)
D
DMSGTEXT      1      80
DFAILRETC     41     44
DFAILRSNC     46     49
DMESSAGEID    S          7  INZ(' ')
DMESSAGEFILE  S          21 INZ(' ')
DMSGKEY       S          4  INZ(' ')
DMSGTYPE      S          10 INZ('*INFO ')
DSTACKENTRY   S          10 INZ('* ')
DSTACKCOUNTER S          9B 0 INZ(2)
DERRCODE      DS
DBYTESIN      1      4B 0 INZ(0)
DBYTESOUT     5      8B 0 INZ(0)
C*
C*****
C* START OF PROGRAM *
C* *
C* *
C*-----*
C* Set the keyword in the rule array *
C*-----*
C          MOVEL   'ADAPTER1'  RULEARRAY
C          MOVE    'RQ-TOKEN'  RULEARRAY
C          Z-ADD   2           RULEARRAYCNT
C*-----*
C* Set the verb data length to 8 *
C*-----*
C          Z-ADD   8           VERBDATALEN
C*****
C* Call Cryptographic Facility Control SAPI */
C*****
C          CALLP   CSUACFC     (RETURNCODE:
C                               REASONCODE:
C                               EXITDATALEN:
C                               EXITDATA:
C                               RULEARRAYCNT:
C                               RULEARRAY:
C                               VERBDATALEN:
C                               VERBDATA)
C*-----*
C* Check the return code *
C*-----*
C          RETURNCODE  IFGT      4
C*          *-----*

```

```

C*          * Send error message *
C*          *-----*
C              MOVE      MSG(1)      MSGTEXT
C              MOVE      RETURNCODE   FAILRETC
C              MOVE      REASONCODE   FAILRSNC
C              EXSR      SNDMSG
C              RETURN
C              ENDIF
C*
C*          *-----*
C*          * Send success message for the 1st step *
C*          *-----*
C              MOVE      MSG(2)      MSGTEXT
C              EXSR      SNDMSG
C*
C*-----*
C* Set the keyword in the rule array for 2nd step *
C*-----*
C              MOVE      'RQ-REINT'   RULEARRAY
C*
C*-----*
C* Convert the token into the one's complement of it *
C*-----*
C              MOVE      VERBDATA     BUFFER
C              Z-ADD     0             INT4
C              MOVE      A1           INT2
C              EVAL     INT4 = 65535 - INT4
C              MOVE      INT2         A1
C              MOVE      A2           INT2
C              EVAL     INT4 = 65535 - INT4
C              MOVE      INT2         A2
C              MOVE      A3           INT2
C              EVAL     INT4 = 65535 - INT4
C              MOVE      INT2         A3
C              MOVE      A4           INT2
C              EVAL     INT4 = 65535 - INT4
C              MOVE      INT2         A4
C              MOVE      BUFFER       VERBDATA
C*
C*****
C* Call Cryptographic Facility Control SAPI *
C***** */
C              CALLP     CSUACFC      (RETURNCODE:
C                                  REASONCODE:
C                                  EXITDATALEN:
C                                  EXITDATA:
C                                  RULEARRAYCNT:
C                                  RULEARRAY:
C                                  VERBDATALEN:
C                                  VERBDATA)
C*-----*
C* Check the return code *
C*-----*
C      RETURNCODE   IFGT      4
C*
C*          *-----*
C*          * Send error message *
C*          *-----*
C              MOVE      MSG(1)      MSGTEXT
C              MOVE      RETURNCODE   FAILRETC
C              MOVE      REASONCODE   FAILRSNC
C              EXSR      SNDMSG
C*
C              ELSE
C*
C*          *-----*
C*          * Send success message *
C*          *-----*
C              MOVE      MSG(3)      MSGTEXT

```

```

C          EXSR      SNDMSG
C*
C          ENDIF
C          SETON
C*
C*****
C* Subroutine to send a message
C*****
C      SNDMSG      BEGSR
C                  CALL      'QMHSNDPM'
C                  PARM      MESSAGEID
C                  PARM      MESSAGEFILE
C                  PARM      MSGTEXT
C                  PARM      MSGLENGTH
C                  PARM      MSGTYPE
C                  PARM      STACKENTRY
C                  PARM      STACKCOUNTER
C                  PARM      MSGKEY
C                  PARM      ERRCODE
C                  ENDSR

```

```

**
CSUACFC failed with return/reason codes 9999/9999.
Random token was successfully returned.
The Cryptographic Coprocessor successfully cleared/reset.

```

Use the Hardware Service Manager

Hardware service manager is a tool for displaying and working with system hardware from both a logical and a packaging viewpoint, an aid for debugging Input/Output (I/O) processors and devices, and is also used to reinitialize the Cryptographic Coprocessor (set it back to an un-initialized state).

When the Cryptographic Coprocessor is re-initialized, the Cryptographic Coprocessor Licensed Internal Code is reloaded into the Coprocessor. Some but not all program temporary fixes (PTFs) for the Coprocessor licensed internal code may require the use of hardware service manager to activate them. This extra step is included to allow you to prepare for recovery because reloading certain segments of the licensed internal code will cause any configuration data including master keys, retained RSA private keys, roles, and profiles to be lost.

There may be situations where the Cryptographic Coprocessor must be reset back to an uninitialized state. For example, if the Coprocessor is not configured correctly, there could be a scenario where the Coprocessor can not perform any useful function and cannot be corrected using the Cryptographic Coprocessor configuration utility or a user-written application. Another example is if the passwords for the administrative profiles are forgotten and no other profile uses a role that is authorized to change passwords.

Hardware service manager is found in System Service Tools. Use the Start System Service Tools (STRSST) CL command by typing STRSST at the CL command line and pressing enter. The System Service Tools Signon display should be shown.

Start Service Tools (STRSST) Sign On

SYSTEM: RCHSYS01

Type choice, press Enter.

Service tools user _____
 Service tools password _____

F3=Exit F9=Change Password F12=Cancel

Enter the service tools user profile name and password. The System Service Tools display should appear.

System Service Tools (SST)

Select one of the following:

1. Start a service tool
2. Work with active service tools
3. Work with disk units
4. Work with diskette data recovery
5. Work with system partitions
6. Work with system capacity

Selection
1

F3=Exit F10=Command entry F12=Cancel

Select **1** to start a service tool and press Enter. The Start a Service Tool display will be shown.

Start a Service Tool

Warning: Incorrect use of this service tool can cause damage to data in this system. Contact your service representative for assistance.

Select one of the following:

1. Product activity log
2. Trace Licensed Internal Code
3. Work with communications trace
4. Display/Alter/Dump
5. Licensed Internal Code log
6. Main storage dump manager
7. Hardware service manager

Selection
7

F3=Exit F12=Cancel F16=SST menu

Select 7 to start Hardware Service Manager. The Hardware Service Manager screen will be displayed showing the menu of available options.

```

Hardware Service Manager

Attention: This utility is provided for service representative use only.

System unit . . . . . : 9406-270 10-4314M
Release . . . . . : V5R1M0 (1)

Select one of the following:

1. Packaging hardware resources (systems, frames, cards,...)
2. Logical hardware resources (buses, IOPs, controllers,...)
3. Locate resource by resource name
4. Failed and non-reporting hardware resources
5. System power control network (SPCN)
6. Work with service action log
7. Display label location work sheet
8. Device Concurrent Maintenance

Selection
  7

F3=Exit      F6=Print configuration      F9=Display card gap information
F10=Display resources requiring attention      F12=Cancel
```

Select 2 to work with logical hardware resources.

```

Logical Hardware Resources

Select one of the following:

1. System bus resources
2. Processor resources
3. Main storage resources
4. High-speed link resources

Selection
  2

F3=Exit      F6=Print configuration      F12=Cancel
```

From the Logical Hardware Resources screen, select 1 to show system bus resources.

Logical Hardware Resources on System Bus

System bus(es) to work with *ALL *ALL, *SPD, *PCI, 1-511
 Subset by *CRP *ALL, *STG, *WS, *CMN, *CRP

Type options, press Enter.
 2=Change detail 4=Remove 5=Display detail 6=I/O Debug
 8=Associated packaging resource(s) 9=Resources associated with IOP

| Opt | Description | Type-Model | Status | Resource Name |
|-----|---------------------------|------------|-------------|---------------|
| _ | HSL I/O Bridge | 2249- | Operational | BC02 |
| _ | Bus Expansion Adapter | - | Operational | BCC02 |
| _ | System Bus | 2249- | Operational | LB01 |
| _ | Multi-Adapter Bridge | 2249- | Operational | PCI01D |
| _ | Combined Function IOP * < | 284D-001 | Operational | CMB01 |
| _ | HSL I/O Bridge | 283B- | Operational | BC01 |
| _ | Bus Expansion Adapter | - | Operational | BCC03 |

More...

F3=Exit F5=Refresh F6=Print F8=Include non-reporting resources
 F9=Failed resources F10=Non-reporting resources
 F11=Display serial/part numbers F12=Cancel

If you know which IOP contains the Cryptographic Coprocessor, type **9** next to the IOP. Otherwise, subset the list by typing ***CRP** for "Subset by" field and then type **9** next to the IOP containing the Cryptographic Coprocessor. You should then see the Logical Hardware Resources Associated with IOP display.

Logical Hardware Resources Associated with IOP

Type options, press enter.
 2=Change detail 4=Remove 5=Display detail 6=I/O Debug
 7=Verify 8=Associated packaging resource(s)

| Opt | Description | Type-Model | Status | Resource Name |
|----------|---------------------------|------------|-------------|---------------|
| _ | Combined function IOP * < | 284D-001 | Operational | CMB01 |
| _ | Cryptography Adapter | 4758-023 | Operational | CRPCTL01 |
| 6 | Cryptography Device | 4758-023 | Operational | CRP01 |
| _ | Workstation IOA | 2746-001 | Operational | CTL01 |
| _ | Display Station | 3477-0FC | Operational | DSP001 |
| _ | Display Station | 3477-0FC | Operational | DSP002 |
| _ | Communications IOA | 2745-001 | Operational | LIN01 |
| _ | Communications Port | 2745-001 | Operational | CMN01 |
| _ | Communications Port | 2745-001 | Operational | CMN02 |
| _ | Communications IOA | 2744-001 | Operational | LIN03 |
| _ | Communications Port | 2744-001 | Operational | CMN03 |

More...

F3=Exit F5=Refresh F6=Print F8=Include non-reporting resources
 F9=Failed resources F10=Non-reporting resources
 F11=Display serial/part numbers F12=Cancel

Type **6** next to the cryptography device that you want to reinitialize, and then press Enter.

Select Cryptography Debug Function

Select one of the following:

1. Reinitialize Flash Memory
2. Select IOP Debug Function

Selection

1

F3=Exit F12=Cancel

Select **1** to reinitialize flash memory (reload the Cryptographic Coprocessor Licensed Internal Code). A confirmation screen will be displayed. If you are applying a PTF ensure that you have taken the necessary precautions regarding your encrypted data and keys, and have a backup of the master key. Press Enter to continue.

Reinitialize Flash Memory Function

DANGER:

Performing this initialization of the flash memory on the cryptography device will cause ALL key information stored on the device to be **DESTROYED**. This will cause all data encrypted using this device to be rendered unusable.

WARNING:

Performing this initialization of the flash memory on the cryptography device will take an estimated 10 minutes.

Press Enter to proceed.

F3=Exit F12=Cancel

The following screen showing status of the reinitialization will be displayed and updated until reinitialization is complete.

Reinitialize Flash Memory Status

Flash memory reinitialization in progress...

Estimated time: 10.0 minutes

Elapsed time: 2.5 minutes

When reinitialization is complete, a message will be displayed.

Select Cryptography Debug Function

Select one of the following:

1. Reinitialize Flash Memory
2. Select IOP Debug Function

Selection

-

F3=Exit F12=Cancel



Reinitialization of cryptography device was successful.

After reinitialization is complete, exit all the way out of system service tools by pressing function key F3 on each screen as necessary.


Related information for cryptographic hardware

The following resources provide additional information relating to cryptographic concepts or hardware:

IBM Sources

- The IBM Cryptographic hardware  contains information on the 4764 and 4758 Cryptographic Coprocessor hardware solutions.
- The IBM PCI Cryptographic Coprocessor documentation library  contains the CCA 3.2x Basic Services Manual for the 4764 Cryptographic Coprocessor, in addition to the 2.5x CCA Basic Services

Manuals for the 4758 Cryptographic Coprocessor. These downloadable PDF documents are intended for systems and applications analysts and application programmers who will evaluate or create CCA programs.

- The CCA 2.41 Basic Services Manual  and the CCA 2.52 Basic Services Manual  are intended for systems and applications analysts and application programmers who will evaluate or create programs for the IBM 4758 Common Cryptographic Architecture (CCA) support.

Code disclaimer information

This document contains programming examples.

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you "AS IS" without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

DB2
developerWorks
eServer
IBM
i5/OS
iSeries
Operating System/400
OS/400
pSeries
Redbooks
xSeries
zSeries

SET[™] and the SET Logo are trademarks owned by SET Secure Electronic Transaction[™] LLC.

Other company, product, and service names may be trademarks or service marks of others.

Terms and conditions for downloading and printing publications

Permissions for the use of the publications you have selected for download are granted subject to the following terms and conditions and your indication of acceptance thereof.

Personal Use: You may reproduce these Publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these Publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations. IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

All material copyrighted by IBM Corporation.

By downloading or printing a publication from this site, you have indicated your agreement with these terms and conditions.

Code disclaimer information

This document contains programming examples.

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you "AS IS" without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.



Printed in USA