AIX Version 7.3

4765 PCIe Cryptographic Coprocessor AIX CCA Support Program Installation 4.4

Note

Before using this information and the product it supports, read the information in “Notices” on page 65.

This edition applies to AIX Version 7.3 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this document

This installation information describes Release 4.4 of the IBM® Common Cryptographic Architecture (CCA) Support Program (hereafter referred to as Support Program) for the IBM 4765 PCIe Cryptographic Coprocessor. The Support Program includes device drivers, utilities, and the CCA coprocessor code.

Use this information to help with the following tasks:

- Obtain the Support Program through the Internet
- Load the software onto a host computer and into the coprocessors.
- Use the utilities supplied with the Support Program to:
  - Load the coprocessor function-control vector (FCV)
  - Initialize one or more coprocessors
  - Create and manage access-control data
  - Create a master key and primary key-encrypting keys (KEKs)
  - Manage keystore at the cryptographic node
  - Create node-initialization file lists to set up and configure other cryptographic nodes
- Link your application software to the CCA libraries
- Obtain guidance for security considerations in application development and operational practices

Audience

The audience for this publication includes:

- System administrators who install the software
- Security officers responsible for the coprocessor access-control system
- System programmers and application programmers who determine how the software is to be used

Highlighting

The following highlighting conventions are used in this document:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Case-sensitivity in AIX

Everything in the AIX operating system is case-sensitive, which means that it distinguishes between uppercase and lowercase letters. For example, you can use the **ls** command to list files. If you type LS, the system responds that the command is `not found`. Likewise, `FILEA`, `FiLea`, and `filea` are three distinct file names, even if they reside in the same directory. To avoid causing undesirable actions to be performed, always ensure that you use the correct case.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Related publications

Publications for the PCIe Cryptographic Coprocessor and commercial cryptographic applications in general follow:

Cryptographic hardware publications are available at the *CryptoCards* website at http://www.ibm.com/security/cryptocards:

- *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and the IBM 4764 PCI-X Cryptographic Coprocessors*

4765 PCIe Cryptographic Coprocessor AIX CCA Support Program Installation 4.4

To use the information effectively, you must be familiar with commands, system calls, subroutines, file formats, and special files.

Support Program installation process overview

This overview of AIX CCA explains the procedure to install and operate the IBM Cryptographic Coprocessor Support Program on a host computer.

Related information

"Installing the Support Program" on page 2
Procedure to install the IBM Common Cryptographic Architecture (CCA) Support Program on the coprocessor host computer.

Obtaining coprocessor hardware and software

Information about selecting, installing, and ordering the coprocessor hardware, and to download the software.

The following sections describe how to:

- Order coprocessors
- Placing orders for IBM 4765 coprocessor
- Installing the IBM 4765 hardware
- Obtaining the coprocessor software

Ordering coprocessors

The IBM 4765-001 is ordered from IBM as a machine type and model. The coprocessor requires a PCIe slot that accepts a 2/3 length PCIe adapter.

The software supports up to eight coprocessors per system, depending on the number of PCIe slots available.

Placing orders for IBM 4765 coprocessor

To order the coprocessor hardware, contact your local IBM representative or your IBM Business Partner, and order the model and features you have selected.

Customers in the U.S.A. can also contact IBM Direct at 1-800-IBM-CALL. Specifically mention *IBM 4765* with your order to be directed to the group that processes IBM 4765 orders.

Installing the IBM 4765 hardware

The IBM 4765 is installed in a manner similar to other PCIe adapters. Follow the process described in the *IBM 4765 PCIe Cryptographic Coprocessor Installation 4.4* for detailed information.

Obtaining the coprocessor software

The software can be obtained by downloading it from the website: http://www.ibm.com/security/cryptocards/pciecc/ordersoftware.shtml.

Installing the Support Program

Procedure to install the IBM Common Cryptographic Architecture (CCA) Support Program on the coprocessor host computer.

The IBM Common Cryptographic Architecture (CCA) Support Program consists of several components, including:

- Device drivers and an operating system for the PCIe cryptographic coprocessor hardware
- Support for the IBM Common Cryptographic Architecture (CCA) application program interface (API)
- A function-control vector (FCV)

  Note: An FCV is a signed value provided by IBM. It enables the CCA application within the coprocessor to yield a level of cryptographic service consistent with applicable cryptographic implementation import and export regulations.
- Utility applications where the coprocessor must be installed that runs on the host machine

To install and configure the IBM Common Cryptographic Architecture (CCA) Support Program, complete these steps:

1. Choose the platform support packages that are appropriate to your setup:

   AIX® 6.1 or later.

   See "Obtaining coprocessor hardware and software" on page 1 for details.
2. Place an order for the hardware with IBM or your IBM Business Partner. See "Obtaining coprocessor hardware and software" on page 1 describes how to order and receive the coprocessor hardware from IBM.
3. Download the Support Program for your operating system. See "Obtaining coprocessor hardware and software" on page 1 describes how to install the embedded operating system, and the CCA application program into the PCIe Cryptographic Coprocessor.
4. Install the Support Program onto the coprocessor host computer.
5. Install the coprocessor hardware. See "Obtaining coprocessor hardware and software" on page 1 for details.
6. Load the coprocessor software. See "Loading and Unloading software into the coprocessor" on page 6 for details.
7. Set up a CCA test node. You can establish a CCA cryptographic node by using the utilities provided with the Support Program, or link your application programs to the CCA API. Also verify the access control and other setup requirements imposed by the application software you plan to use with the IBM 4765. The CCA Node Management (CNM) utility, described in "Managing the cryptographic node by using the CNM and CNI utilities" on page 15, includes setup and management functions needed to:
   - Load the FCV
   - Create and edit the access control data
   - Manage the coprocessor master key
   - Manage primary key encrypting keys (KEKs)
   - Manage the storage of data keys
   - Create lists (scripts) for the CCA Node Initialization (CNI) utility
8. Run test programs that utilize the CCA libraries. See "Building applications to use with the CCA API" on page 36 for details.

Related information

"Obtaining coprocessor hardware and software" on page 1
Information about selecting, installing, and ordering the coprocessor hardware, and to download the software.

"Loading and Unloading software into the coprocessor" on page 6

After installing the IBM Common Cryptographic Architecture (CCA) Support Program on the host computer, use the Coprocessor Load Utility (CLU) to load the coprocessor operating system and CCA application into the coprocessor.

"Managing the cryptographic node by using the CNM and CNI utilities" on page 15
A computer that provides cryptographic services, such as key generation and digital signature support, is defined here as a *cryptographic node*.

Installing the Support Program base release 4.4

Instructions for installing the Support Program on the coprocessor host computer.

Prerequisites

Before you begin the installation choose the platform support packages that are appropriate to your setup. See "Obtaining coprocessor hardware and software" on page 1 for details on software and hardware requirements for AIX.

Note: If you are not installing the program for the first time, back up your key storage files.

To install the Support Program:

1. Enter the **smitty install_all** command.
2. Enter the location of the installation images that you obtained by using the procedure described in Obtaining the coprocessor software section under"Obtaining coprocessor hardware and software" on page 1. Press Enter.
3. Enter `csufx.4765.cca csufx.4765.man` in the **SOFTWARE install** field or press F4 (Display) to select from the list. Verify that **AUTOMATICALLY install requisite software** is set to yes and that **ACCEPT new license agreements** is set to yes. Use the tab key to toggle or the F4 (Display) key to list. Press Enter and press Enter again to continue when prompted `ARE YOU SURE`.
4. Exit from **smitty** using the F10 (Exit) key.
5. Read the `/usr/lpp/csufx.4765/README` file. This file contains the latest information about the Support Program product.
6. Use the configuration utilities to configure the software as described in "Configuring the Support Program" on page 3.

Configuring the Support Program

This section describes the utilities and system command used to configure the CCA Cryptographic Coprocessor Support Program software.

csufadmin

Specifies the system-access permissions that are associated with the csufkeys, csufappl, csufclu (Coprocessor Load Utility), csufcnm (Cryptographic Node Management), and csufcni (Cryptographic Node Initialization) utilities.

Default permissions restrict the use of these utilities to the root user and to users in the system group. Use the csufadmin utility to modify these permissions.

csufappl

Specifies the system-access permissions that are associated with the CCA libraries.

The default permissions restrict the use of the CCA libraries to the root user and members of the system group. Use the csufappl utility to permit other groups to use the services furnished by the CCA API.

csufkeys

Creates and identifies the file and directory names of the locations wherein the cryptographic keys and key lists are stored. The installation program defines, in the AIX object data manager (ODM), the following default directories:

- AES key-record-list directory: `/usr/lpp/csufx.4765/csufkeys/aeslist`
- AES key-storage file: `/usr/lpp/csufx.4765/csufkeys/aes.keys`
- DES key-record-list directory: `/usr/lpp/csufx.4765/csufkeys/deslist`
- DES key-storage file: `/usr/lpp/csufx.4765/csufkeys/des.keys`
- PKA key-record-list directory: `/usr/lpp/csufx.4765/csufkeys/pkalist`
- PKA key-storage file: `/usr/lpp/csufx.4765/csufkeys/pka.keys`

Use the csufkeys utility to change the storage locations.

Note: When you initialize key storage by using the Cryptographic Node Management utility, ensure that you specify the ODM directories that are defined by this utility.

odmget

Verifies key-storage file names with the **odmget** system command. You can verify the key-storage names used by the CCA Support Program by entering the **odmget csufodm** command. The four parameter name attributes specify the following values:

- **csuaesds**: The file containing the AES key-records
- **csuaesld**: The directory containing the AES key-record-list files
- **csudesds**: The file containing the DES key-records
- **csudesld**: The directory containing the DES key-record-list files
- **csupkads**: The file containing the PKA key-records
- **csupkald**: The directory containing the PKA key-record-list files

When initializing CCA key-storage with either the CNM utility or with the csnbksi CCA verb, you must use the file names that are returned from the ODM. Use the csufkeys utility to change these file names.

The DES_Key_Record_List verb, PKA_Key_Record_List verb, and the AES_Key_Record_List verb produce list files in the `/usr/lpp/csufx.4765/csufkeys/deslist`, `/usr/lpp/csufx.4765/csufkeys/pkalist`, and `/usr/lpp/csufx.4765/csufkeys/aeslist` directories respectively. These are the default directory names. You can modify the directory names when you install the software. The list files are created under your ownership, if you request the list service. Make sure that the files are created under the group ID as required by the installation. This can also be achieved by setting the set-group-id-on-execution bit on in these three directories. See the g+s flags in the **chmod** command for more information. If this procedure is not followed, errors are returned on key-record-list verbs.

To assign a default CCA Coprocessor, use the EXPORT command to set the environment variable **CSU_DEFAULT_ADAPTER** to **CRP0***n*, where *n* = 1, 2, 3, 4, 5, 6, 7, or 8, depending on which installed CCA Coprocessor you want as the default. If this environment variable is not set when the first CCA verb of a process is called, the CCA software uses Coprocessor CRP01 as the default. If this environment variable is set to an invalid value, you will get an error until the environment variable is set to a valid value.

Related information

“Creating a key label” on page 34

CCA Support Program and AIX file permissions

The CCA Support Program relies on file permissions at the group level to function accurately.

The users and administrators of the Support Program must have the correct group file permissions on the CCA shared libraries, utilities, key-storage files, and directories to be fully functional and to run without errors.

Note: Key-storage files and directories are defined as those files and directories that are contained in the key-storage directory. This directory includes the top-level key-storage directory, that is, in the default configuration, all the files and directories under the `/usr/lpp/csufx.4765/csufkeys/deslist` directory, and the `/usr/lpp/csufx.4765/csufkeys` directory itself.

To operate the key-storage files and directories must have a group ID of the application user group, that is, the groupname parameter that is used when the `csufappl` utility was run.

Also, as a rule, all key-storage directories must have file permissions of 2770 (drwxrws---) and be owned by the root. All key-storage files must have file permissions of 660 (-rw-rw----).

The 4765 CCA software and keystore cannot exist concurrently with the 4764 CCA software and keystore because of conflicts in the libraries and ODM databases.

Reviewing coprocessor hardware errors

Errors occurring in the IBM Power Systems coprocessor hardware is recorded in the AIX error log.

To process and view the log, enter the following command:

```
errpt -a -N Cryptn,libxcrypt.a | more
```

Where *n* is 0, 1, 2, 3, 4, 5, 6 or 7 (for example, Crypt 0), depending on which CCA Coprocessor log you want to view.

Related information

“Loading and Unloading software into the coprocessor” on page 6
After installing the IBM Common Cryptographic Architecture (CCA) Support Program on the host computer, use the Coprocessor Load Utility (CLU) to load the coprocessor operating system and CCA application into the coprocessor.

Removing the Support Program

If your key-storage files are in the default directories, back them up or save them before you remove the IBM Cryptographic Coprocessor (CCA) Support Program. Removing the software deletes the key-storage files in the default directories.

To remove the IBM Cryptographic Coprocessor Support Program, follow these steps:

1. Log on as `root`.
2. Enter the **`rmdev -dl Crypt0`** command. The coprocessor device driver and other related information are removed. You can use this command for each CCA coprocessor that you plan to remove or relocate.
3. Enter the **`smitty install_remove`** command.

   Note: When prompted, enter the `csufx.4765.com` and `devices.pciex.14107a0314107b03.rte` product names.
4. Verify that the **REMOVE dependent software** value is set to NO. Also, verify that the **Preview Only** value is set to NO.
5. Press the **Enter** key.

AIX hardware and software requirements

The prerequisites that are required to install CCA.

Hardware

Install an IBM Power Systems server with an available 4765 PCIe cryptographic coprocessor.

During installation of the software, the driver interacts with the coprocessor to arbitrate interrupt settings, DMA channels, and other system resources. For installation instructions about the coprocessor hardware and device driver, see “Obtaining coprocessor hardware and software” on page 1.

Software

1. IBM AIX 6.1 and later.
2. Java Runtime Environment (JRE) 1.6.0, or later, that is required to run the CCA Node Management (CNM) utility.
3. The software package **csufx.4765** must be downloaded from the http://www.ibm.com/security/cryptocards/pcixcc/ordersoftware.shtml website. The software package contains the following filesets:
   - **csufx.4765.com** - 4765 CCA Support Program
   - **csufx.4765.cca** - 4765 Support Program - Common Utilities
   - **csufx.4765.man** - Support Program man pages

File permissions

Manage the file permission by using the CCA Node Management (CNM) utility.

The CCA Node Management (CNM) utility provides a way to manage access control points. To help protect against accidental or intentional corruption of the CNM utility's executable file, set the file permission of the `CNM.jar` file to read and execute only. Similarly, to protect the data file of access control points, set the file permission of the `csuap.def` file to read only.

Loading and Unloading software into the coprocessor

After installing the IBM Common Cryptographic Architecture (CCA) Support Program on the host computer, use the Coprocessor Load Utility (CLU) to load the coprocessor operating system and CCA application into the coprocessor.

If you obtain updates to the Support Program, use the CLU to reload the necessary program segments. You can also load vendor software by using the CLU.

This section includes:

- Instructions for using the CLU to understand which coprocessors are installed and their status, and to install and uninstall the software that runs within the coprocessor
- A reference section that describes:
  - The coprocessor memory segments
  - Validation of the coprocessor status
  - The syntax used to start the CLU utility
  - CLU return codes

For a deeper understanding of the code-loading controls and the security considerations implemented by the coprocessor, see the research paper *Building a High-Performance, Programmable Secure Coprocessor* that is available on the product website library page at http://www.ibm.com/security/cryptocards.

Notes:

1. The file locations referred to in this section are the default directory paths.
2. The error codes returned by the coprocessor device driver are presented in the form of a hexadecimal number, such as X'8040xxxx'. You might encounter the errors, especially when you first use the CLU utility and are less familiar with the product and its procedures.
3. The coprocessor function-control vector (FCV) is loaded by the CCA Node Management (CNM) utility.

Related information

“Device driver error codes” on page 42
The coprocessor device driver monitors the status of its communication with the coprocessor and the coprocessor hardware-status registers.

“Managing the cryptographic node by using the CNM and CNI utilities” on page 15

A computer that provides cryptographic services, such as key generation and digital signature support, is defined here as a *cryptographic node*.

Loading coprocessor software

Find the procedures to load software into the coprocessor in this section.

See the README file that accompanies the software distribution that you are installing for specific `.clu` file names. The README file might also provide additional information that enhances or modifies these general procedures.

Use the following subtopics, follow this sequence of tasks:

1. At a command prompt, change to the directory with the Coprocessor Load Utility (CLU) files and run the CLU.
2. Determine the software that is currently resident within the coprocessor.
3. Change the contents of software segments 1, 2, and 3, as appropriate.
4. Validate the final contents of the software segments.

Changing the default directory and running the CLU

To change the default directory, you must locate the directory that contains the coprocessor code files (*.clu) and the Coprocessor Load Utility (CLU).

Changing the default directory

At a command prompt, change to the default directory coprocessor code directory `/usr/lpp/csufx.4765/clu` to access the code files. If the CLU is not in the default directory, ensure that your operating system can locate the CLU.

Running the CLU

Note: When using CLU, applications that use CCA must not be running.

To run the CLU utility, enter the **`csufclu`** program name at the command prompt .

You can provide parameters interactively to the CLU utility, or you can include these on the command line. Each time you use CLU you must specify a log file name. This is the first parameter and can be included on the command line. In general, when working with a specific coprocessor, it is best to use the coprocessor serial number as the log file name. You can obtain the serial number from the label on the bracket at the end of the coprocessor.

CLU will append information to two log files. If the log files do not exist, they are created. One log file contains the same information that is normally displayed on your console. The other log file, to which CLU will assign MRL as the file name extension, contains a machine-readable log. The MRL file is used with an analysis utility.

Note: Subsequent instructions in this section assume that you use CLU interactively. Change to the directory that contains the coprocessor code files. Start CLU with the name appropriate to your operating system. Respond to the prompts as requested.

CLU obtains the number of installed coprocessors from the device driver. If you have more than one installed coprocessor, CLU requests the number of the coprocessor with which you intend to interact. The numbers (coprocessor_*number*) can be 0 - 2. To correlate these numbers to a particular coprocessor, use the System Status (SS) command to learn the number for each of the installed coprocessors. (For an example of the output, see Figure 2 on page 15 in the Coprocessor Load Utility commands topic.)

Note: The CLU utility can operate with a coprocessor when it obtains exclusive control of the coprocessor. If any other application such as a thread is running and has performed the CCA verb calls, the coprocessors that are loaded with CCA will be "busy" and unusable by CLU.

Related information

"Coprocessor Load Utility syntax" on page 12

Determining coprocessor software segment contents

The coprocessor has three segments: segment 1, segment 2, and segment 3. Each segment has a status, holds software and a validation public key, and an identifier of the owner (except for segment 1).

See Table 1 on page 8 for information about the segments of the coprocessor.

Table 1. Software segment contents

Segment	Content
1	Miniboot contains diagnostics and code loading controls
2	Embedded control program
3	CCA or another application

You determine the current content and status of the coprocessor segments by using the ST command. Figure 1 on page 8 shows a typical ST response.

```
======================================================================
CSUFCLU V4.1.1 st.log ST    begun Tue Sep 13 09:30:25 2011
*********** Command ST started. ---- Tue Sep 13 09:30:25 2011

 *** VPD data;  PartNum = 45D5117
 *** VPD data;  EC Num = 0G43192
 *** VPD data;  Ser Num = 99000543
 *** VPD data;  Description = IBM 4765-001 PCI-e Cryptographic Coprocessor
 *** VPD data;  Mfg. Loc. = 91
 *** ROM Status; POST0 Version 1, Release 27
 *** ROM Status; MiniBoot0 Version 1, Release 20
 *** ROM Status; INIT: INITIALIZED
 *** ROM Status; SEG2: RUNNABLE , OWNER2: 2
 *** ROM Status; SEG3: RUNNABLE , OWNER3: 2
 *** Page 1 Certified: YES
 *** Segment 1 Image: S0103 P1v0607 M1v011B P2v0706 F5180 201104151205401A000022000000000000
 *** Segment 1 Revision: 40105
 *** Segment 1 Hash: 177C AF13 C601 2276 90AA 8E20 D3BB BA58 79A6 7EBA 6C2A D68B 0A34 33E0 802C
4EA7
 *** Segment 1 Hash: 177C AF13
 *** Segment 2 Image: 4.1.7    y4_12-lnx-2011-03-04-16    201108111338401A000000000100010900
 *** Segment 2 Revision: 40107
 *** Segment 2 Hash: 698A 29DC EF8A 44D8 A025 3117 491B C552 45DA EC6F 0D0C 6671 BABE 7ABF 41E7
2FF5
 *** Segment 2 Hash: 698A 29DC
 *** Segment 3 Image: 4.1.7    CCA                        201108121155401A000000000000000000
 *** Segment 3 Revision: 40107
 *** Segment 3 Hash: EC02 B93A 309F 882A D859 031D 1F22 839D 2233 4D6A C58D D93C E43F 4A4C 1234
9F48
 *** Segment 3 Hash: EC02 B93A
 *** Query Adapter Status successful ***
Obtain Status ended successfully!
*********** Command ST ended. ---- Tue Sep 13 09:31:26 2011

  ...finishing up...
*********** Command ST exited. ---- Tue Sep 13 09:31:46 2011
```

Figure 1. Typical CLU status response

Definitions of the fields on the ST response follow:

Field
Description

PartNum
The part number (P/N) of the coprocessor.

EC Num
The engineering change number of the coprocessor.

Ser Num
The manufacturer's serial number of the coprocessor. This number is not the IBM tracking serial number that is used for warranty verification and download authorization.

Description
A statement that describes the type of coprocessor in general terms. Auditors must review this and other status information to confirm that an appropriate coprocessor is in use.

ROM Status
The coprocessor must always be in an INITIALIZED state. If the status is ZEROIZED, the coprocessor detected a possible tamper event and is in an unrecoverable, nonfunctional state. (Unintended tamper events are created if the coprocessor is not handled properly. Only remove the batteries when you follow the recommended procedure to change the battery, maintain the coprocessor in the safe temperature range, and follow the instruction.

ROM Status SEG2 / SEG3
Several status conditions for Segment 2 and Segment 3 exist, which includes:

- UNOWNED: Currently not in use, no content
- RUNNABLE: Contains code and is in an usable state

Owner identifiers are also shown. The standard CCA Support Program is assigned identifier 2 for both Segment 2 and Segment 3. **Any other owner identifier** indicates that the software is not the standard IBM CCA product code. In all cases, ensure that the software is loaded in your coprocessor. Unauthorized or unknown software can represent a security risk to your installation.

Segment 1 Image
The name and description of the software content of Segment 1. For a factory shipped coprocessor, the name includes **Factory**. This image and the associated validation key must be changed.

For a previously loaded coprocessor, the Segment 1 name probably includes CCA. Ensure that you observe the revision level.

Segment 2 and Segment 3 Images
If these segments have Owned status, observe the image name and the revision level. IBM incorporates CCA in the image name to indicate that the image is provided as part of the CCA Support Program. Be sure to observe the revision level.

Segment Hash values
The hash values for each segment must match the values that are shown in Figure 1 on page 8.

Changing software segment contents

Generally, the software within the coprocessor must be at the same release level as the CCA software in the hosting system.

Do not attempt to use various different release levels except with specific instructions from IBM.

Start the Coprocessor Load Utility (CLU) and enter the parameters interactively. For instructions, see “Changing the default directory and running the CLU” on page 7.

1. Enter the log file name (*nnnnnnn*`.LOG`, where *nnnnnnn* is the serial number of the coprocessor).
2. Enter the command, `PL`.
3. If you have multiple coprocessors, enter the coprocessor number.
4. Enter the CLU file name as indicated in the README file.

Repeat as required so that the appropriate software is loaded for Segments 1, 2, and 3.

Validating the coprocessor segment contents

The procedure to be followed to validate the contents of the coprocessor segments.

After you have loaded or replaced the code in Segments 1, 2, and 3, use the **CLU VA** command to confirm the segment contents and to validate the digital signature on the response created by the coprocessor.

Depending on the IBM 4765 coprocessor (PartNum) in use,[1] issue the following command, and substitute the class key certificate file name from Table 2 on page 10 for the data file name. Note that the data file name `v.clu` is appended to the coprocessor part number, all in lowercase characters.

```
csuxclu nnnnnnnn.log VA [coprocessor_n] datafile
```

The part number can be obtained by using the Coprocessor Load Utility (CLU) ST command.

Table 2. Class-key file for use with the CLU VA command

PartNum	Class-key certificate file
12R8565	12r8565v.clu
41U0441	41u0441v.clu

The [coprocessor_*n*] parameter is the optional designator for a particular coprocessor and defaults to zero.

Unloading coprocessor software and zeroize the CCA node

The steps to unload the coprocessor software and to zeroize the CCA node to surrender the ownership of the segments are described here.

When you use Coprocessor Load Utility (CLU) to process a file that surrenders ownership of Segment 2, both Segment 2 and the subordinate Segment 3 are cleared, and the code is removed. The validating public key for the segment is cleared, the security-relevant data items that are held within the coprocessor for the segment are zeroized. The owner identifiers are cleared, and the segment's status is set to UNOWNED.

See the README file that accompanies the software distribution you are using for the specific `.clu` file name that is used to surrender ownership of Segments 2 and 3. The README file might also provide additional information that amplifies or modifies this general procedure.

Perform these actions:

- Change to the directory that contains the CLU files.
- Start the CLU utility.
- Respond to the prompts and use the serial number of the coprocessor in the log file name.
- Use the **PL** command to surrender Segment 2 as indicated in the README file for your platform.

Notes:

1. You can also zeroize CCA without removing the software by using the CCA reinitialize process.
2. IBM does not normally make available a file to restore the factory Segment 1 validating key to put the coprocessor into a condition similar to a factory-ready product. Segment 1 can be changed to a limited number of times before the available Device Key certificate space is used and the coprocessor is potentially rendered unusable. If you require the capability to restore the validating key of Segment 1, and are willing to display your coprocessor to a possible lock-up condition, you can obtain the required file from IBM by submitting a query by using the Support Form on the product website, http://www.ibm.com/security/cryptocards. It is important to note that certificate space is a nonrenewable resource. After it is used, it cannot be recovered.

Related information

“Initializing the node” on page 23

[1] You can refer to the IBM product website (http://www.ibm.com/security/cryptocards) FAQ section for the procedure to validate coprocessor integrity. That topic carries the current list of class key certificate files.

The procedure to initialize the CCA node to its initial state.

Coprocessor Load Utility (CLU) reference

The coprocessor memory segments to which you load the software is described here. The approach the coprocessor uses to validate the software loads, the syntax used to start the CLU, and the CLU return codes.

If you do not need the details in this section, skip to "Managing the cryptographic node by using the CNM and CNI utilities" on page 15.

Coprocessor memory segments

Coprocessor memory segments are organized into different segments.

The organization of memory segments and its function follows:

Table 3. Organization of memory segments

Segment	Description
0	Basic code The basic code manages coprocessor initialization and the hardware component interfaces. This code cannot be changed after the coprocessor leaves the factory.
1	Software administration and cryptographic routines Software in this segment: • Administers the replacement of software already loaded to Segment 1. • Administers the loading of data and software to segments 2 and 3. • Is loaded at the factory, but can be replaced using the CLU utility.
2	Embedded operating system The coprocessor Support Program includes the operating system. The operating system supports applications loaded into Segment 3. Segment 2 is empty when the coprocessor is shipped from the factory.
3	Application software The coprocessor Support Program includes a CCA application program that can be installed into Segment 3. The application functions according to the IBM CCA and performs access control, key management, and cryptographic operations. Segment 3 is empty when the coprocessor is shipped from the factory.

Validating the coprocessor software loads

When the coprocessor is shipped from the factory, it has within it the public key that is needed to validate replacement software for Segment 1.

To load code into coprocessor Segment 2 and Segment 3, for each segment follow these steps:

1. Identify an owner for the segment by using an **`Establish Owner`** command. The owner identifier is only accepted if the digital signature associated with this identifier can be validated by the public key that is residing with the immediately lower segment. Once established, ownership remains in effect until a **`Surrender Owner`** command is processed by the coprocessor.
2. Load the segment to the code. Two different commands are available.
   a. Initially use the **Load** command. The **Load** command data includes a public key certificate that must be validated by the public key that is present on the next lower segment. The coprocessor

accepts the code and retains the validated public key for the segment if one of the condition is satisfied:

- The certificate is validated.
- The data of the owner identifier in the **Load** command matches the current ownership that is held by the coprocessor for the segment.
- The complete data in the **Load** command can be validated by the public key in the certificate that was used for validation.

b. If a segment already has a public key, a **Reload** command can be used to replace the code in a segment. The coprocessor actions are the same as for a **Load** command, except that the included certificate must be validated by the public key associated with the target segment rather than the key associated with the next lower segment.

The embedded operating system, working with the coprocessor hardware, can store security-relevant data items (SRDIs) on behalf of itself and an application in Segment 3. The SRDIs are zeroized upon tamper detection, loading of segment software, or processing a **Surrender Owner** command of a segment. The SRDIs for a segment are not zeroized when the **Reload** command is used. The CCA application stores the master keys, the function control vector (FCV), the access control tables, and the retained RSA private keys as SRDI information that is associated with Segment 3.

IBM signs its own software. If another vendor intends to supply software for the coprocessor, that vendor's **Establish Owner** command and the code-signing public key certificate must be signed by IBM under a suitable contract. These restrictions make sure that the following conditions are satisfied:

- Only authorized code can be loaded into the coprocessor.
- Government restrictions are met relating to the import and export of cryptographic implementations.

Coprocessor Load Utility syntax

The syntax that is used to start the Coprocessor Load Utility (CLU), and the functions of the utility are described.

CLU must be used for the following functions:

- Ensure that the coprocessors are not busy by ending any application that has used a coprocessor. For example, end all applications that use the CCA API.
- Obtain the release level and the status of software that is installed in the coprocessor memory segments.
- Confirm the validity of digitally signed messages that are returned by the coprocessor.
- Load and reload portions of the coprocessor software.
- Reset the coprocessor.

To start the utility, follow these steps:

1. Log on as required by your operating system.
2. At the command line, change directory to the directory that contains the CLU files. The default directory is `/usr/lpp/csufx.4765/clu`.
3. Enter the **csufclu** utility name followed by the applicable parameters.

If you do not supply the necessary parameters, the utility prompts when the information is required. Optional parameters are enclosed in brackets. The syntax for the parameters that follow the utility name is

```
[log_filecmd[coprocessor _#][data_file][-Q]]
```

[2] In this publication, the terms *load* and *reload* are used. Other documentation might refer to these operations as *emergency burn* (EmBurn), and *regular burn* or *remote burn* (RemBurn).

Where:

log_file
Identifies the log file name. The utility appends entries to this ASCII text file as it performs the operations that are requested. A second machine-readable log file, with a file name of logfile_name. MRL, is also created. This log file can be processed by a program and contains the binary-encoded responses from the coprocessor.

cmd
Specifies a two-letter abbreviation that represents the loader command to be run.

coprocessor *_number*
Provides the coprocessor number as established by the device driver. This parameter defaults to 0. Coprocessors are designated to the device driver as numbers 0, 1, and 2. You can use the serial number information that you obtain with the **ST** or **VA** commands and the serial number that is printed on the end-bracket of the coprocessor to correlate a particular coprocessor to the coprocessor *_number*. The utility supports up to eight coprocessors per system.

data_file
Identifies the data file (drive, directory, and file name) that is used for the requested operation. To identify the *data_file* name, use one of the following methods:

- For software loads and reloads, the *data_file* name is the file name of the software image that you are loading into the coprocessor. The Support Program README file provides the *data_file* name.
- For the coprocessor, the coprocessor status is obtained with the **VA** command. The *data_file* name is the class-key certificate file name that used to validate the coprocessor response. The FAQ section of the product website (http://www.ibm.com/security/cryptocards) contains a description of the procedure for validating the coprocessor and its code. This description also contains a list of the current class-key certificate file names. You can download the required certificate file from the website.

-Q
Suppresses (quiets) the CLU program output to the standard output device. The status information is still appended to the log files.

Example: To obtain the coprocessor status and save the results to the log file, enter:

`csufclu nnnnnnnn.log va` *`datafile_name`*`.clu`

It is suggested that you make *nnnnnnnn* the serial number of the coprocessor. It is not mandatory to use the serial number, but it is used to retain a history of all software changes made to each specific coprocessor.

Related information

“Machine-readable log contents” on page 42
The CLU utility creates two log files, one intended for reading and the other for possible input to a program.

“Coprocessor Load Utility commands” on page 13
The Coprocessor Load Utility (CLU) supports multiple loader commands.

Coprocessor Load Utility commands

The Coprocessor Load Utility (CLU) supports multiple loader commands.

The loader commands and its functions that are supported by CLU are as following:

Table 4. CLU loader commands	
Loader command	**Description**
PL: Load microcode into coprocessor Commands R1, E2, L2, R2, S2, E3, L3, R3, and S3 are inferred from information contained in the data files that you use with the PL command. A single "PL" file can incorporate information for multiple ownership and loading commands.	Processes a series of commands as directed by the contents of the data file to establish segment ownership and to load or reload segment software.
RS: Reset the coprocessor	Resets the coprocessor. Generally you will not use this command. The command causes the coprocessor to perform a power-on reset. You might find this command helpful should the coprocessor and the host-system software lose synchronization. You should end all host-system software processes that are operating with the coprocessor prior to issuing this command to enable the complete cryptographic subsystem to get to a reset state.
SS: Obtain system status	Obtains the part number, serial number, and a portion of the Segment 3 software image name for each of the installed coprocessors, provided that these are not being used by some application such as CCA. See Figure 2 on page 15.
ST: Obtain coprocessor status	Obtains the status of loaded software and the release level of other components. The status is appended to the log files.
VA: Validate coprocessor status	Obtains the status of loaded software and the release level of other components. The data is transmitted in a message signed by the coprocessor device key, and then stored in the utility log file. The utility uses its built-in public key to validate the one-or-more class-key certificates contained in *data_file* name parameter. One of these certificates should validate the public key, or chain of public keys, obtained from the coprocessor, and confirm that the coprocessor has not been tampered with.

In general, the utility can be called by a script file or a command file. When you create a script file or a command file to start the utility on an unattended system, add the "quiet" syntax, the `-q` (or `-Q`, `/q`, or `/Q`) parameter, to request that no output be sent the display. By default, the utility returns prompts and messages to the display.

The *Typical CLU system status response* figure shows the response of a CLU system.

```
====================================================================
CSUFCLU V4.00 ss.log SS    begun Tue Sep 28 10:49:36 2010
*********** Command SS started. ---- Tue Sep 28 10:49:36 2010

 Card #    P/N        S/N         Segment 3 Description
 ------    -------    --------    ----------------------------------
   0       45D6045    99000627    4.1.0    CCA
 *** Query System Status successful ***
System Status ended successfully!
*********** Command SS ended. ---- Tue Sep 28 10:50:37 2010

  ...finishing up...
*********** Command SS exited. ---- Tue Sep 28 10:50:57 2010
```

Figure 2. Typical CLU system status response

Coprocessor Load Utility return codes

This section specifies the returned code values from CLU.

When CLU finishes processing, it returns a value that can be tested in a script file or in a command file. Each of the returned values have their implications.

0
OK. This implies that the CLU finished processing properly.

1
Command line parameters are not valid.

2
Cannot access the coprocessor. In this case, ensure that the coprocessor and its driver have been properly installed.

3
Check the utility log file for an abnormal condition report.

4
No coprocessor is installed. In this case, ensure that the coprocessor and its driver have been properly installed.

5
An Invalid coprocessor number is specified.

6
A data file is required with this command.

7
The data file specified with this command is incorrect or invalid.

Managing the cryptographic node by using the CNM and CNI utilities

A computer that provides cryptographic services, such as key generation and digital signature support, is defined here as a *cryptographic node*.

The CCA Node Management (CNM) utility and the CCA Node Initialization (CNI) utility that are provided with the Support Program are tools to set up and manage the CCA cryptographic services provided by a node.

This section includes:

- Utilities and description on how to start them.
- Sample scenarios for using the utilities that you might consider.
- How to use the CNM utility administrative functions: Review this material after working through the topic “Scenario: Creating a test node” on page 17.

- How to create and manage access control data: Read details about the access control portion of the CNM utility.
- How to manage cryptographic keys: Read about some of the key management tasks that you can accomplish with the CNM utility.
- How to establish other nodes by using the CNI utility: You can automate use of the CNM utility by using encapsulated procedures.

These utilities are written in Java™ and require the use of a Java runtime environment (JRE). You can also use the Java Development Kit (JDK).

CNM and CNI overview

Typical users of the CCA Node Management (CNM) utility and the CCA Node Initialization (CNI) utility are security administration personnel, application developers, system administrators, and, in some cases, production-mode operators.

Notes:

1. The CNM utility furnishes a limited set of the CCA API services. After becoming familiar with the utility, you can determine whether it meets your needs or whether you require a custom application to achieve more comprehensive administrative control and key management.
2. Files that you create through use of the CNM utility might be dependent on the release of the Java Runtime Environment (JRE). If you change the release of the Java Runtime Environment (JRE) that you use, files that you have created with the CNM utility might not function correctly with the new release.
3. The CNM utility has been designed for use with a mouse. Use the mouse instead of the **Enter** key for consistent results.
4. No help panels are provided for the Master-Key Cloning portion of the utility.
5. These utilities use the IBM Common Cryptographic Architecture (CCA) Support Program API to request services from the coprocessor. The *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* manual contains a comprehensive list of the verbs (also known as callable services or procedure calls) provided by the CCA API. Refer to this book and the individual services described therein to understand which commands might require authorization in the various roles that you define by using the procedures described in this section.

CCA node management utility overview

The CCA Node Management utility is a Java application that provides a graphical interface to use in the setup and configuration of IBM 4765 CCA cryptographic nodes. The utility functions primarily to set up a node, create and manage access-control data, and manage the CCA master-keys that are necessary to administer a cryptographic node.

You can load data objects directly into the coprocessor or save them to disk. The data objects are usable at other IBM 4765 CCA nodes that use the same operating system and a compatible level of the Java application.

Note: Starting the CCA Node Management Utility: To start the CCA Node Management utility enter the **`csufcnm`** command The CNM utility logo and then the main window are displayed.

CCA node initialization utility overview

The CCA Node Initialization utility runs scripts that you create by using the *CNI Editor* within the CNM utility. These scripts are known as *CNI lists*. The CNI utility can run the CNM utility functions that are necessary to set up a node; for example, it can be used to load access-control roles and profiles.

As you create a CNI list, you specify the disk location of the data objects that the CNI utility will load into the target nodes. After creating a CNI list, you can distribute the CNI list and any accompanying data files (for roles, profiles, and so on) to nodes where the CNI utility will be used for an automated setup. The source node and all nodes running the distributed CNI list must employ the same operating system and a compatible level of the Java application.

Note: Starting the CCA Node Management Utility: To start the CCA Node Management utility enter the **csufcnm** command The CNM utility logo and then the main window are displayed.

Related information

“Scenario: Cloning a DES or PKA master key” on page 20
The steps to clone a data encryption standard (DES) or public key algorithm (PKA) master key from one coprocessor to another.

“Creating other nodes by using the CNI utility” on page 35
Creating a CNI list for the CCA Node Initialization (CNI) utility, allows to load keys and access control data stored on disk into other cryptographic nodes without running the CNM utility on those target nodes.

Scenarios: Using the CNM and the CNI utilities

This section describes using the CCA Node Management (CNM) utility and the CCA Node Initialization (CNI) utility to create a node and clone it to another coprocessor.

The usage of the utilities is illustrated in the scenarios, which includes:

1. Creating a test node to be used to develop applications or establish procedures for using the CNM utility. *First time users should follow this procedure to begin experimentation with the utility and the coprocessor.*
2. Creating nodes for a production environment using key parts. This scenario employs CNI lists to automate establishment of target production nodes.
3. Cloning a master key from one coprocessor to another coprocessor. This is a procedure of interest to high security installations that employ multiple coprocessors.

The purpose of the scenarios is to illustrate how the procedures described here can be used. Where appropriate, a scenario refers to other sections of this topic collection with more detailed information.

If you are not familiar with the coprocessors's CCA access control system, see “Access control overview” on page 25 and “Initial state of the access control system” on page 25. Here you can find an explanation of terms such as *role initial DEFAULT role*, and *user profile*. The scenarios assume that the access-control system is in its initial state.

Note: These scenarios are instructional only. You are encouraged to determine the procedures best suited for your specific environment. Refer to the appendix about secure operations in the IBM CCA Basic Services Reference and Guide for the *IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*.

Scenario: Creating a test node

In this scenario, a single programmer sets up a node to allow unlimited access to cryptographic services.

Important: The resulting cryptographic node must not be considered secure because under this scenario many sensitive commands are permitted with unrestricted use.

Prerequisites: You must have already installed an appropriate level of the Java Runtime Environment (JRE) or the Java Development Kit (JDK).

To create a test node, complete the following steps:

1. Install the coprocessor and the IBM Cryptographic Coprocessor Support Program as described in Installing the Support Program.
2. Start the CCA Node Management utility by entering the **csufcnm** command. The CNM utility logo and the main panel displays.
3. If you have more than one coprocessor with CCA installed, specify to the CNM utility which coprocessor you want to use. From the **Crypto Node** menu, select **Select Adapter**. A list of available adapter numbers (1 - 8) is displayed. Select an adapter (coprocessor) from the list. If you do not use the **Select Adapter** list to select an adapter, the default adapter (coprocessor) is used.
4. Synchronize the clock within the coprocessor and host computer. From the **Crypto Node** menu, click **Time**. From the resulting submenu, click **Set**. The clocks are synchronized.

5. Use the CNM utility to permit all commands in the DEFAULT role:
   a. From the `Access Control` menu, click **Roles**.
   b. Highlight the **DEFAULT** entry and click **Edit**. A window displays the commands that are enabled and those that are not enabled by the DEFAULT role.
   c. Click **Permit All**.
   d. Load the modified role back into the coprocessor by clicking **Load**, select **OK**.
   e. Save a copy of the role by clicking the **Save** button and name the role.
6. Load the function-control vector (FCV) into the coprocessor. From the **Crypto Node** menu, click **Authorization**. From the resulting submenu, click **Load** to specify and load the FCV.

   The FCV file is the one that was placed on your server during the installation process. FCVs usually have file names such as `fcv_td4kECC521.crt` and is searched using the file search utility available with your operating system.
7. Install a master key from the **Master Key** menu, click either **DES / PKA Master Keys** or **AES Master Keys**, and click **Yes**. The coprocessor generates and sets a random master key.

   The master key that was installed with the **Auto Set** option has actually passed through the main memory of your system processor as key parts. For production purposes, use a more secure method of establishing a master key, such as random generation or installation of known key parts entered by two or more individuals. These options are also accessed from the menus mentioned previously.
8. Initialize the key storage files. For information on initializing the key storage files, see "Creating or initializing key storage" on page 33

   Key storage is a CCA term that describes a place where the Support Program can store Data Encryption Standard (DES), Rivest-Shamir-Adleman algorithm (RSA), and Advanced Encryption Standard (AES) cryptographic keys under names that you (or your applications) define. If you intend to use key storage, you must initialize the key storage file or files that correspond to the type of keys that you are using: DES, RSA (PKA), or AES. For example, if you intend to use only DES keys, you must initialize the DES key storage file but not the others. If you intend to use DES and PKA keys, you must initialize the DES and PKA key storage files but not the AES key storage file. If you intend to use all three, you must initialize all three.

Related Links

"Creating a role" on page 26

"Loading the master key automatically" on page 31

Scenario: Creating nodes in a production environment

In this scenario, the responsibility for creating cryptographic nodes is divided among three individuals, namely, an access control administrator and two key management officers.

The administrator sets up the node and its access control system. Then, the key management officers load a master key and any required key encrypting keys (KEKs). The KEKs can be used as transport keys to convey other keys between nodes.

This scenario is focused on installing master keys and high level, internode data encryption standard (DES) KEKs from *key parts*. The CCA implementation supports alternatives to the key part technique such as random master-key generation and distribution of DES keys by using techniques that are based on Rivest-Shamir-Adleman (RSA) public key technology. The *key part technique* assumes that there are two *key management officers* who can be trusted to perform their tasks and to not share their key part information. This technology implements a *split knowledge* policy. The access control system is set up to enforce *dual control* by separating the tasks of the first and second officers.

In this scenario, the access-control administrator uses the cryptographic node management (CNM) utility to prepare coprocessor node initialization (CNI) lists for the target nodes. The CNI lists automate the process of using the CNM utility at the target node. The administrator prepares a CNI list for the tasks that are performed by the target node access control administrator and the two key management officers.

The administrator must know the commands require authorization in the target node under different conditions, which includes:

- Normal, limited operation (when the default role is used)
- When the access control administrator tasks are run
- When each of the key management officer tasks are run
- Under any other special circumstances by using additional roles and profiles

Note: The CNM and CNI utilities are tools that are used to set up and manage the CCA cryptographic services that are provided by a node.

The administrator authorizes commands in the various roles to ensure that only required commands are enabled. Sensitive commands, such as loading a first key part or loading subsequent key parts, are only enabled in roles for users with the responsibility and authority to use those commands. It is important to separate the responsibilities so that policies such as split knowledge and dual control are enforceable by the coprocessor's access control system.

Related information

“Creating and managing access control data” on page 24

Scenario: Preparing CNI lists for target nodes

In this task, the access control administrator uses the CCA Node Management (CNM) utility to prepare CCA Node Initialization (CNI) lists for the target nodes.

To set up the node and create its access control data, the access control administrator can:

1. On an established node, start the CNM utility.
2. Create and save to disk the access control data for the target node, which includes:
   - Supervisory roles and user profiles for the access control administrator and the key management officers
   - A default role to replace the initial default role

   a. To create a CNI list to synchronize the clock and calendar within the coprocessor and host computer.
      i) Load the access control data.
      ii) Log on as an access control administrator.
      iii) Load the replacement default role.
      iv) Load the function control vector (FCV).
      v) Log off.

   b. Create a CNI list for the first key-management officer:
      i) Log on as the first key management officer.
      ii) Load a first master key of the key part.
      iii) Load the first part key encrypting key information.
      iv) Log off.

   c. Create a CNI list for the second key management officer:
      i) Log on as the second key management officer.
      ii) Load a second master key of the key part.
      iii) Load the second part key encrypting key information.
      iv) Log off.
3. Install the coprocessor and the IBM Common Cryptographic Architecture (CCA) Support Program onto the target nodes.
4. Transport to the target nodes the access control data and the FCV specified in the CNI list.

5. With the involvement of the key management officers, on each target node run the CNI lists that you created in steps “2.a” on page 19, “2.b” on page 19, and “2.c” on page 19.

The target nodes are now ready to provide cryptographic service.

Related information

“Creating and managing access control data” on page 24

“Creating other nodes by using the CNI utility” on page 35

Creating a CNI list for the CCA Node Initialization (CNI) utility, allows to load keys and access control data stored on disk into other cryptographic nodes without running the CNM utility on those target nodes.

Scenario: Preparing and loading key parts

This section describes the procedure to prepare, load and transport the key parts.

The key management officers prepare the key parts for use at the target nodes and load the key parts at the target nodes.

Decide the method to transport the key parts from the point of generation to the point of installation. Following are a few possibilities:

- Generate the key parts at a central place and transfer these on diskettes.
- Generate the key parts at a central place and transfer these on paper forms.
- Generate the key parts at the point and time of (first) installation. If the key parts are required after the installation, to reload or to share with another node, then you must decide on the method to transport the key parts.

Review the specific capabilities of the CNM utility by working with the utility. Then review the specific approach that you select and test the CCA Node Initialization utility (CNI) list that was prepared in conjunction with the access control administrator.

Scenario: Cloning a DES or PKA master key

The steps to clone a data encryption standard (DES) or public key algorithm (PKA) master key from one coprocessor to another.

The term *cloning* is used rather than copying because the master key is split into shares for transporting between the coprocessors. The technique is explained under the topic "Understanding and managing master keys" in the IBM CCA Basic Services Reference and Guide for the *IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* manual. The section “Cloning a master key” on page 43 provides a step-by-step procedure that you can follow. The background information that allows to vary the procedure is described in this section.

Note: Cloning of an AES master key is not supported.

Cloning of the master key involves two or three nodes:

- The master key source node.
- The master key target node.
- The share administration (SA) node. The SA node can either be the source or the target node.

The CNM utility can store various data items that are involved in this process in a database that you can carry (diskette) or transfer (FTP) between the different nodes. One database `issa.db` that is the default, and contains the information about the SA key and keys that is certified. The target node where the master key is cloned also has a database that is known by default as the `csr.db`.

You can accomplish these tasks by using the CNM utility:

1. Start the CCA Node Management utility by entering the **`csufcnm`** command. The CNM utility logo and the main window are displayed.
2. Set up the nodes in a secure manner with access control roles, user profiles, and master keys.

   You need a role and one or more user profiles at the source and target nodes for each user who obtains or store shares. Processing of shares is done by a separate command so that, if you want,

your roles can ensure that independent individuals are involved with obtaining and installing the different shares.

Consider the use of random master key generation and roles that enforce a dual control security policy. For example, allow one individual or role to register a hash and another individual or role to register a public key. Select different individual or role for obtaining and installing the individual shares of the master key.

See the guidance section in the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* manual for the description of the Master_Key_Process and the Master_Key_Distribute verbs.

3. Install a unique 1 - 16 byte environment ID (EID) of your choice into each node.

   From the `Crypto Node` menu, click Set Environment ID, enter the identifier, and click Load. Use only these characters in an EID: A - Z, a - z, 0 - 9, and @, (X'40'), space character (X'20'), &, (X'26'), and =, (X'3D').

   You must enter a full 16-character identifier. For short identifiers, complete the entry with space characters.

4. Initialize the master key sharing m and n values in the source and target nodes. These values must be the same in the source and the target nodes. The value n is the maximum number of shares while m is the minimum number of shares that must be installed to reconstitute the master key in the target node.

   From the `Crypto Node` menu, click **Share Administration** > **Set number of shares**, enter the values, and click Load.

5. At the different nodes, generate these keys and have each public key that is certified by the SA key. You can use the utility's `sa.db` database to transport the keys and the certificates.

   Share administration (SA)
   This key is used to certify itself and the following keys. You must register the hash of the SA public key, and the public key itself, in the SA, source, and target nodes.

   After the SA key is created, the utility supplies an 8 byte or 16-hexadecimal character value that is a portion of the hash of the SA key. *Be sure to retain a copy of this value.* You need this value to confirm the hash value that is recorded in the database to register the SA public key at the source and target nodes.

   Coprocessor Share Signing (CSS)
   This key is used to sign shares that are distributed from the source node. The private key is retained within the source node.

   Coprocessor Share Receiving (CSR)
   This key is used to receive a share-encrypting key into the target node. The SA certified public CSR key is used at the source node to wrap (encrypt) the share encrypting key that is unique for each share. The private key is retained within the target node.

   Generate the Key Pairs: SA, CSS, and CSR
   From the `Crypto Node` menu, click **Share Administration** > **Create Keys**. Click the **Share Administration Keys**, **CSS key**, or **CSR key**. Click **Create**.

   You must supply key labels for the CSS and CSR keys that are retained in the source and target nodes, for example, `IBM4765.CLONING.CSS.KEY` and `IBM4765.CLONING.CSR.KEY`. The labels that you use must not conflict with other key labels that are used in your applications.

   To generate the CSR key at the share-receiving node, you must obtain the serial number of the coprocessor. From the **Crypto Node**, click **Status**. You must enter the serial number value to certify the CSR key.

6. Register the SA public key in the coprocessor at the SA, source, and target nodes. This process is a two-step process that must be done under a dual control security policy.

One individual installs the SA public key hash. From the **Crypto Node** menu, click **Share Administration** > **Register Share Administration** , and click SA Key hash. You must enter the hash value that is obtained during SA key creation.

The other individual installs the actual SA public key. From the `Crypto Node`menu, click **Share Administration** > **Register Share Administration** , and click SA Key. By default, the public key information is in the `sa.db` file.

7. Take the CSS key and the CSR key to the SA node and have the keys that are certified.

   From the `Crypto Node` drop-down menu, select **Share Administration Keys**, **Certify KeysCSS key**, or **CSR key**.

   For the CSR key, you must supply the serial number of the target coprocessor as a procedural check that an appropriate key is being certified. Your procedures must include communicating this information in a reliable manner.

8. At the source node, the authorized individuals must sign on to the role that allows them to obtain their shares. At least m shares must be obtained. These shares are of the current master-key.

   From the `Crypto Node` menu, click **Share Administration** > **Get Share**, and enter the share number to be obtained. **Observe the serial numbers and database identifiers**. When these shares are in agreement, click Get Share. The share information must be placed by default into the `csr.db` file and obtains the CSR key certificate, by default, from the `sa.db` file.

   Obtain current-master-key validation information for use later at the target node. From the `Master Key` menu, click **DES/PKA Master Keys** > **Verify**. Click **Current**.

9. At the target node, the authorized individuals must sign on to the role that allows each of them to install their share. At least m shares must be installed to reconstitute the master key into the new master-key register.

   From the `Crypto Node`menu, click **Share Administration** > **Load Share**, and select the share number to be installed. Verify that the serial numbers and database identifiers are correct and then click **Observe the serial numbers and database identifiers**. When these shares are agreed to be correct, click Get Share. At the target node, the authorized individuals must sign on to the role that allows the individuals to install their share. The share information is obtained by default from the `csr.db` file and the CSS key certificate is obtained by default from the `sa.db` file. If your server has multiple cryptographic coprocessors that are loaded with CCA, the coprocessors must have identical master keys that are installed for the functioning of key storage.

   When m shares are loaded, verify that the key in the new master-key register is the same as the current master key in the source node when the shares were obtained. On the target node, from the `Master Key` menu, click **DES/PKA Master Keys** > **New**.

10. When it is confirmed through master key verification that the master key is cloned, an authorized individual can set the master key. This action deletes any old master key and moves the current master key to the old master key register. Application programs that use keys encrypted by the master key can be impacted by this change, so ensure that setting of the master key is coordinated with the needs of your application programs.
11. From the `Master Key`menu, click **DES/PKA Master Keys** > **Set**.

Using the CNM utility functions

This section describes the procedure to use the various functions of the CNM utility.

Selecting a specific coprocessor

The procedure to choose a coprocessor from the multiple coprocessors available on the system.

If your system has multiple coprocessors loaded with the CCA code, you need to select the specific coprocessor to work on. If you do not make a selection, you will operate with the default coprocessor. After you make a coprocessor selection, that selection remains in effect for the current utility session or until you make a different selection within the utility session.

To select a coprocessor, click **Select Adapter** from the **Crypto Node** menu. If you do not select an adapter, the default adapter is used.

Note:

1. When using the CLU utility, coprocessors are referred to as 0, 1, and 2. Any particular coprocessor might or might not have the CCA application installed. With the CNM utility (and other applications that use the CCA API), the coprocessors loaded with the CCA application are designated as 1, 2, and 3. These new identifiers are assigned by CCA while it scans all of the installed coprocessors for those loaded with the CCA application.
2. When coding a CCA application, keywords **CRP01**, **CRP02**, and **CRP03** are used to allocate a coprocessor. These correspond to the numbers 1, 2, and 3 that are used in the CNM utility menu.

Initializing the node

The procedure to initialize the CCA node to its initial state.

You can restore the CCA node to its initial state, provided that the role you are operating under (the default role or a logged-on role) permits use of the **Reinitialize Device** command (offset X'0111').

Use of the **Reinitialize Device** command causes the following actions to occur:

- Clearing Master-key registers
- Clearing retained Public Key Algorithm (PKA) and registered PKA public keys
- Clearing roles and profiles and restoring the access control to its initial state.

To initialize the CCA node, select **Initialize** from the Crypto Node menu. You will be asked to confirm your action.

Related information

“Initial state of the access control system” on page 25
The initial state has an initial default role.

Logging on and logging off the node

A user must log on to the coprocessor in order to activate a user profile and the associated role. This is the only way to use a role other than the default role.

To log on, select **Passphrase Logon** from the **File** menu.

To log off, select **Logoff** from the **File** menu.

Note: With the exception of the DEFAULT role, access to the coprocessor is restricted by passphrase authentication.

Loading the function-control vector

The procedure to load the coprocessor FCV.

A function-control vector (FCV) is a signed value provided by IBM to enable the CCA application in the coprocessor to provide a level of cryptographic service consistent with applicable import and export regulations. Under the current regulations all users are entitled to the same level of cryptographic functionality. Therefore, IBM now supplies a single FCV with the IBM Common Cryptographic Architecture (CCA) Support Program.

You use the CNM utility to load the FCV into the coprocessor. The FCV file is named `fcv_td4kECC521.crt`.

To load the FCV:

1. From the **Crypto Node** menu, select **Authorization**.
2. From the resulting submenu, click **Load** to specify the FCV file on disk. Specify the file name and click **Update**. The utility loads the FCV.
3. Click **OK**.

Configuring the CCA Node Management utility

The procedure to configure the default values for the CNM utility.

The configuration panel of the CNM utility allows you to indicate directory paths for the files you create with the utility. However, the utility generally does not use the paths that you store in the configuration panel. Instead, the default paths are stored in the Windows environment variables. You might find the configuration panel a useful place to record where you intend to keep the various classes of data items.

Synchronizing the clock and calendars

The procedure to synchronize the clock and calendars within the coprocessor and the host computer.

The coprocessor uses its clock and calendar to record the time and date and to prevent replay attacks in the passphrase-based profile authentication. After installing the coprocessor, synchronize its clock and calendar with that of the host system.

To synchronize the clock and calendars:

1. From the **Crypto Node** menu, click **Time**.
2. From the resulting submenu, click **Set**.
3. Type **Yes** to synchronize the clock and calendars with the host.
4. Click **OK**.

Obtaining status information of the CCA application

You can use the CNM utility coprocessor to obtain the status of the CCA application.

The supported status panels on the CNM utility coprocessor are:

CCA Application:
Displays the version and the build date of the application, and also displays the status of the master-key registers.

Adapter:
Displays the coprocessor serial number, ID, and hardware level.

Command History:
Displays the five most recent commands and sub commands sent to the coprocessor.

Diagnostics:
Indicates whether any of the coprocessor tamper-sensors have been triggered, whether any errors have been logged, and reflects the status of the coprocessor batteries.

Export Control:
Displays the maximum strength of the cryptographic keys used by the node, as defined by the function-control vector (FCV) that is resident within the coprocessor.

To view the status panels:

1. From the **Crypto Node** menu, click **Status**. The CCA application status is displayed.
2. To select other status information, use the buttons at the bottom.
3. Click **Cancel**.

Related information

“Managing the master keys” on page 30
A master key is used to encrypt local-node working keys while they are stored external to the coprocessor.

Creating and managing access control data

The access control system of the IBM CCA Cryptographic Coprocessor Support Program defines the circumstances under which the coprocessor can be used. It does this by restricting the use of CCA commands.

For a list of these CCA commands, see the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*. Also, see the "Required commands" section at the end of each verb description.

An administrator can give users differing authority so that some users can use CCA services not available to others. This section includes an overview of the access control system and instructions for managing your access control data. You need to know the commands that are required and under what circumstances. Consider that some commands should be authorized only for trusted individuals or for certain programs that operate at specific times. Generally, you authorize only those commands that are required, so as not to inadvertently enable a capability that could be used to weaken the security of your installation.

You will obtain the information about command use from the documentation for the applications that you intend to support. For additional guidance, see *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*.

Access control overview

The access control system restricts or permits the use of commands based on roles and user profiles.

Use the CNM utility to create roles that correspond to the needs and privileges of assigned users.

To access the privileges assigned to a role that are not authorized for a default role, a user must log on to the coprocessor by using a unique user profile. Each user profile is associated with a role and multiple profiles can use the same role. The coprocessor authenticates logons by using the passphrase that is associated with the profile that identifies the user.

Note: The term *user* applies to both humans and programs.

The coprocessor always has at least one role, the default role. Use of the default role does not require a user profile. Any user can use the services permitted by the default role without logging on to or being authenticated by the coprocessor.

For example, a basic system might include the following roles:

- **Access control administrator**: Can create new user profiles and modify the access rights of current users.
- **Key management officer**: Can change the cryptographic keys. This responsibility is best shared by two or more individuals making use of rights to enter the first or subsequent key parts.
- **General user**: Can use cryptographic services to protect their work, but has no administrative privileges. If your security plan does not require logon authentication for general users, address their requirements in the default role.

Note: Few individuals would be assigned the roles of key-management officer or access control administrator. Generally, the larger population would not log on and thus would have rights granted in the default role.

Initial state of the access control system

The initial state has an initial default role.

After you have loaded the CCA software support into Segment 3 of the coprocessor, or after the access control system is initialized, no access control data exists except for an initial default role that allows unauthenticated users to create and load access control data.

After creating the roles and profiles needed for your environment, including the supervisory roles necessary to load access control data and to manage cryptographic keys, remove all permissions that are assigned to the default role. Then, add only those permissions you want to grant to unauthenticated users.

Important: The cryptographic node and the data it protects are not secure while the default role is permitted to load access control data.

Related information

"Initial default-role commands" on page 41
The characteristics of the default role after the coprocessor is initialized and when no other access control data exists are described. Also, the enabled access control commands are listed.

Creating a role

A role defines permissions and other characteristics of the users assigned to that role.

To create a role, complete the following steps:

1. From the **Access Control** menu, click **Roles**. A list of currently defined roles is displayed.
2. Select **New** to display the Role Management window. At any time in the process, click **List** to return to the list of currently defined roles.

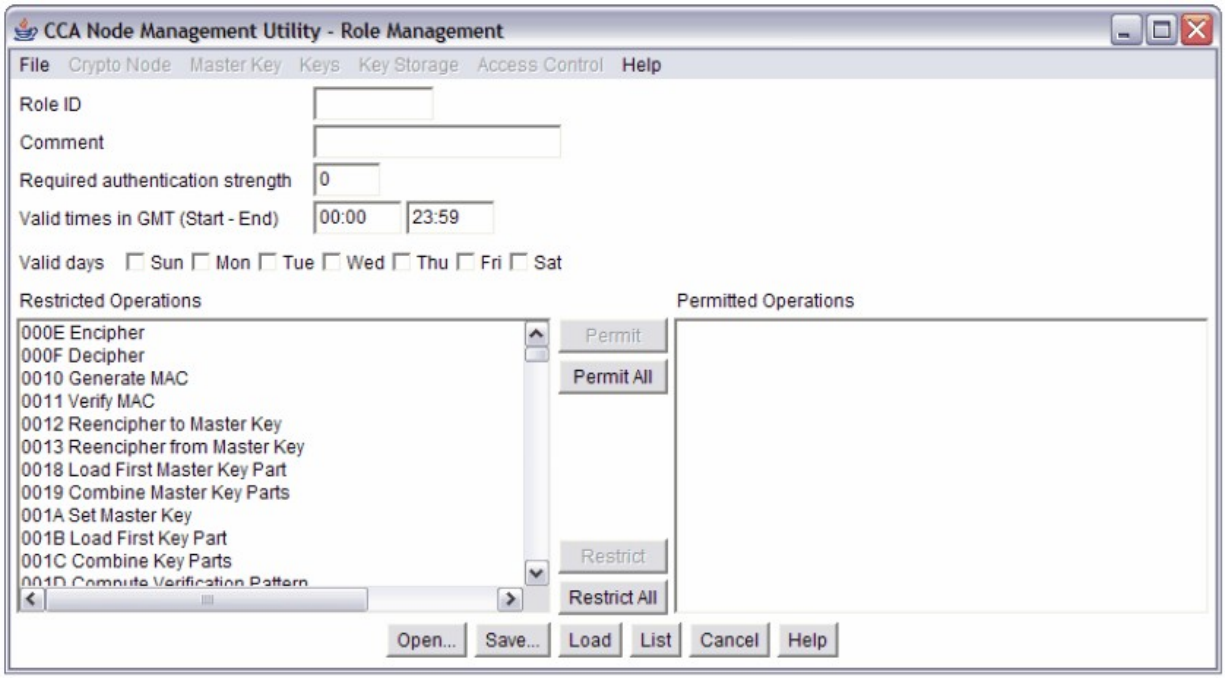

Figure 3. Role Management window

3. Define the role by using the following parameters:

Role ID
A character string that defines the name of the role. This name is contained in each user profile that is associated with this role.

Comment
An optional character string to describe the role.

Required authentication strength
When a user logs on, the strength of the authentication provided is compared to the strength level required for the role. If the authentication strength is less than that required, the user cannot log on. Currently only the passphrase authentication method is supported. Use a strength of 50.

Valid times and valid days
When the user can log on. Note that these times are Coordinated Universal Time. If you are not already familiar with the access control system, see the chapter about access control system of the IBM CCA Basic Services Reference and Guide for the *IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* manual.

Restricted operations and permitted operations
A list defining the commands the role is allowed to use.

Each CCA API verb might require one or more commands to obtain service from the coprocessor. The user requesting service must be assigned to a role that permits those commands needed to run the verb.

For more information about CCA verb calls and commands, refer to the IBM CCA Basic Services Reference and Guide for the *IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* manual.

4. Click **Save** to save the role to disk.
5. Click **Load** to load the role into the coprocessor.

Modifying existing roles

You can use the CNM utility to edit a disk stored and coprocessor stored role and delete a coprocessor stored role.

Note: Any existing role can be used as a template to create a new role. When you open a saved role, the existing information is displayed in the Role Definition window. You need only modify or enter information specific to the new role, give it a new role ID, and load or save it.

Editing a disk-stored role

This section describes the procedure to edit an existing role stored in the disk.

To edit a role stored on disk, complete the following steps:

1. From the **Access Control** menu, click **Roles**. A list of currently defined roles is displayed.
2. Click **Open**. You are prompted to select a file.
3. Open a file. Data is displayed in the Role Definition window.
4. Edit the role.
5. Click **Save** to save the role to disk.
6. Optional: Click **Load** to load the role into the coprocessor.

Editing a coprocessor-stored role

This section describes the procedure to edit the role stored in the CCA coprocessor.

To edit a role stored in the coprocessor, complete the following steps:

1. From the **Access Control** menu, click **Roles**. A list of currently defined roles is displayed.
2. Highlight the role you want to edit.
3. Click **Edit**. Data in the Role Definition panel is displayed .
4. Edit the role.
5. Click **Save**. To save the role to disk.
6. Optional: Click **Load**. To load the role into the coprocessor

Deleting a coprocessor-stored role

This section describes the procedure to delete the role from the CCA coprocessor.

Important: When you delete a role, the CNM utility does not automatically delete or reassign the user profiles associated with that role. You must delete or reassign the user profiles that are associated with a role before you delete the role.

To delete a role stored in the coprocessor, complete the following steps:

1. From the **Access Control** menu, click **Roles**. A list of currently defined roles is displayed.
2. Highlight the role you want to delete.
3. Click **Delete**. The role is deleted.

Creating a user profile

A user profile identifies a specific user to the coprocessor.

To create a user profile, complete the following steps:

1. From the **Access Control** menu, click **Profiles**. A list of currently defined profiles is displayed.
2. Select **New** to display the Profile Management window. See Figure 4 on page 28, to view the fields of the Profile Management window.

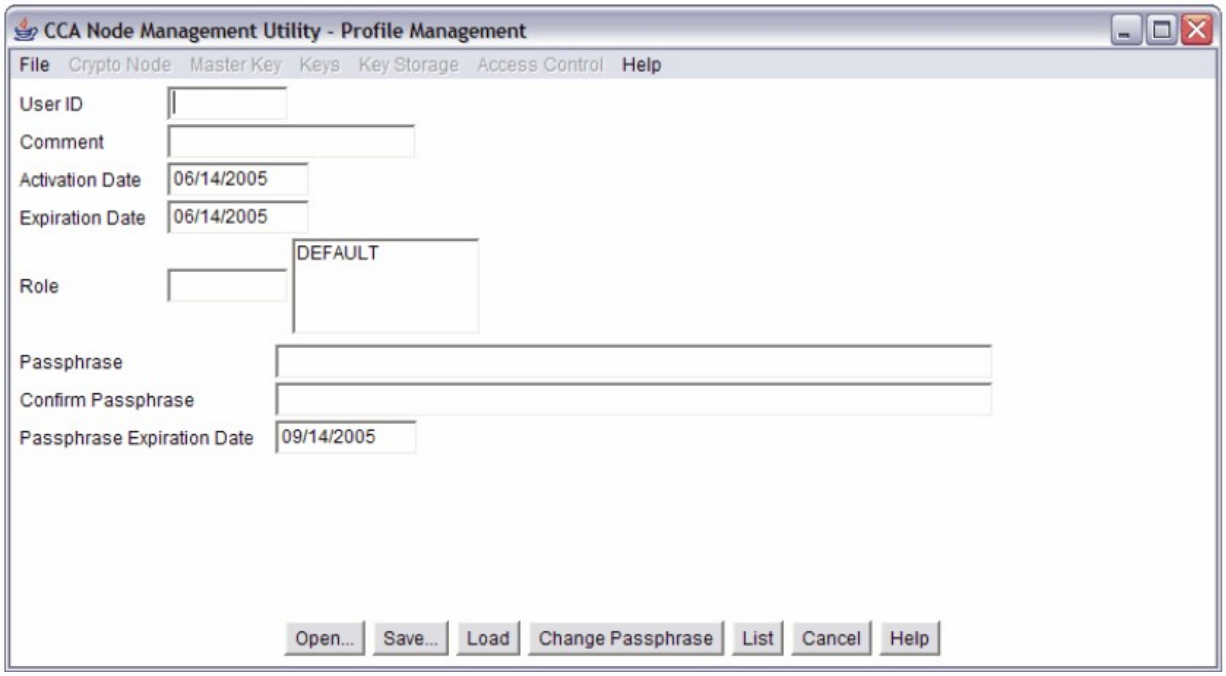

Figure 4. Profile Management panel

3. Define the user profile.

   The fields of the user profile follows:

   User ID
   The name given to a user profile of the cryptographic coprocessor.

   Comment
   An optional character string to describe the user profile.

   Activation Date and Expiration Date
   The first and last dates that the user can log on to the user profile.

   Role
   The name of the role that defines the permissions granted to the user profile.

   Passphrase and Confirm Passphrase
   The character string that the user must enter to gain access to the cryptographic node.

   Passphrase Expiration Date
   The expiration date for the passphrase. The utility will set this by default to 90 days from the current date. You can change the expiration date. Every passphrase contains an expiration date, which defines the lifetime of that passphrase. This is different from the expiration date of the profile itself.

4. Click **Save**, to save the profile to disk.
5. Optional: Click **Load**, to load the profile into the coprocessor.

Modifying existing profile

You can use the CNM utility to edit a disk stored and coprocessor stored profile and delete a coprocessor stored profile.

Note: Any existing profile can be used as a template to create a new profile. When you open a saved profile, the existing information is displayed in the Profile Definition window. You need only modify or enter information specific to the new profile, give it a new profile ID, and load or save it.

Editing a disk-stored user profile

This section describes the procedure to edit a user profile stored on a disk.

To edit a user profile stored on disk, complete the following steps:

1. From the **Access Control** menu, select **Profiles**. A list of currently defined profiles is displayed.
2. Click **Open**. You are prompted to select a file.
3. Open a file. Data is displayed in the User Profile Definition window.
4. Edit the profile.
5. Click **Save** to save the profile to disk.
6. Optional: Click **Load** to load the profile into the coprocessor.

Editing a coprocessor-stored user profile

This section describes the procedure to edit the user profile in the CCA coprocessor.

To edit a user profile stored in the coprocessor, complete the following steps:

1. From the **Access Control** menu, click **Profiles**. A list of currently defined profiles is displayed.
2. Highlight the user profile you want to edit.
3. Click **Edit**. Data in the Profile Definition window is displayed .
4. Edit the user profile.
5. Click **Save**. To save the profile to disk.
6. Optional: Click **Load**. To load the profile into the coprocessor

Deleting a coprocessor-stored user profile

This section describes the procedure to delete the user profile that is stored in the CCA coprocessor.

To delete a profile stored in the coprocessor, complete the following steps:

1. From the **Access Control** menu, click **Profiles**. A list of currently defined user profiles is displayed.
2. Highlight the user profile you want to delete.
3. Click **Delete**. The user profile is deleted.

Resetting the user-profile failure count

To prevent unauthorized logons, the access-control system maintains a logon-attempt failure count for each user profile. If the number of failed attempts for a user profile exceeds the limit defined in the profile, the offending profile is disabled.

To reset the failure count, complete the following steps:

1. From the **Access Control** menu, click **Profiles**. A list of currently defined user profiles is displayed.
2. Highlight the user profile.
3. Click **Reset FC**. A confirmation window is displayed.
4. Click **Yes** to confirm. The logon-attempt failure count is set to 0.

Initializing the access control system

When you initialize the access control system, the CNM utility clears the access control data in the coprocessor and furnishes the default role with the commands required to load access control data.

Important: The cryptographic node and the data it protects are not secure while the default role is permitted to load access control data.

Successfully performing this action removes installed access controls and keys and is therefore a sensitive operation that could render your node inoperable for production. Some installations might choose to remove authorization for this function from their coprocessor's roles. In this event, if you want to initialize the CCA cryptographic node, you must remove the CCA software from the coprocessor and reinstall the CCA software.

To initialize the access control system:

1. From the **Access Control** menu, click **Initialize**. A confirmation window is displayed.
2. Select **Yes** to confirm. The utility initializes the access control system.

Note: To start the CCA Node Management utility enter the **`csufcnm`** command. The CNM utility logo and the main window are displayed.

Managing cryptographic keys

You can use the CNM utility to manage the master keys, to manage primary key-encrypting keys (KEKs), to reset and manage data encryption standard (DES), public key algorithm (PKA), and advanced encryption standard (AES) key-stores. Key types are defined as follows:

A **master key** is a special KEK stored in clear text (not enciphered) and kept within the coprocessor secure module. Three kinds of master keys are supported: DES, PKA, and AES. They are used to wrap other keys so that those keys can be stored outside of the secure module. DES and PKA master keys are 168-bit keys formed from three 56-bit DES keys. AES master keys are 256-bit keys.

Primary KEKs are DES keys shared by cryptographic nodes and are sometimes referred to as transport keys. They are used to encipher other keys shared by the nodes. Primary KEKs, like the master key, are installed from key parts. Knowledge of the key parts can be shared in part by two people to effect a split-knowledge, dual-control security policy.

Other DES keys, PKA keys, and AES keys are enciphered keys that are used to provide cryptographic services, such as media access control (MAC) keys, DATA keys, and private PKA keys.

Note: When exchanging clear key parts, ensure that each party understands how the exchanged data is to be used, because the management of key parts varies among different manufacturers and different encryption products.

Managing the master keys

A master key is used to encrypt local-node working keys while they are stored external to the coprocessor.

CCA defines three master-key registers:

- The **current-master-key register** stores the master key currently used by the coprocessor to encrypt and decrypt local keys.
- The **old-master-key register** stores the previous master key and is used to decrypt keys enciphered by that master key.
- The **new-master-key register** is an interim location that is used to store master-key information as accumulated to form a new master key.

The IBM Common Cryptographic Architecture (CCA) Support Program uses three sets of master key registers, one set for ciphering DES (symmetric) keys, one set for ciphering PKA private (asymmetric) keys, and one set for ciphering AES (symmetric) keys.

Notes:

1. The Master_Key_Distribution master-key-administration verb does not support AES master keys. Programs that use the CCA Master_Key_Process and Master_Key_Distribution, master-key-administration verbs can use the ASYM-MK keyword to steer operations to the PKA asymmetric master-key registers, the SYM-MK keyword to steer to the DES symmetric master-key registers, or both the DES symmetric and PKA asymmetric sets of master-key registers. The CNM utility uses the B0TH option. If you use another program to load master keys and if this program specifically operates on either the SYM-MK or ASYM-MK master-key registers, in general, you will no longer be able to use the CNM utility to administer these master keys. Note that AES master keys work independently from DES and PKA master keys.
2. If your installation has multiple cryptographic coprocessors loaded with CCA, you need to independently administer the master keys in each coprocessor.
3. If your installation has a server with multiple cryptographic coprocessors that are loaded with CCA, those coprocessors need to be installed with identical master keys.

Related information

"Obtaining status information of the CCA application" on page 24
You can use the CNM utility coprocessor to obtain the status of the CCA application.

Verifying an existing master key

The CNM utility generates a verification number for each master key that is stored in the master-key registers. This number identifies the key, but does not reveal information about the actual key value.

To view a master-key verification number, follow these steps:

1. From the Load Master Key window, click **Master Key**.
2. From the **Master Key** menu, select either **DES/PKA Master Keys** or **AES Master Key**, and then click `Verify`; a submenu is displayed.
3. From the resulting submenu, select a master-key register. The verification number for the key stored in that register is displayed.

Loading the master key automatically

The CNM utility can automatically set a master key in the coprocessor. The master key value cannot be viewed from the utility.

Important: If a master key of unknown value is lost, you cannot decipher the key attached to it.

To automatically load the master key, follow these steps:

1. From the Load Master Key window, click **Master Key**.
2. From the **Master Key** menu, select either **DES/PKA Master Keys** or **AES Master Key**.
3. Select **Auto Set** or **Random**. You are prompted to verify the command.
4. Click **Yes**. The coprocessor generates and sets a master key.

Note:

1. The **Random** option is preferred because the **Auto Set** option passes clear key parts through host-system memory.
2. When you set or automatically set a master key, you must reeincipher all keys that were enciphered under the former key.

Related information

"Re-encipher the stored keys" on page 33

Loading a new master key from key parts

To set a new master key in the coprocessor, enter any part of the key in the new master-key register, and set the new master key.

To set the new master key, follow these steps:

1. From the **Master Key** menu, select either **DES/PKA Master Keys** or **AES Master Key**, and then click **Parts**. The Load Master Key window is displayed as shown in Figure 5 on page 32.

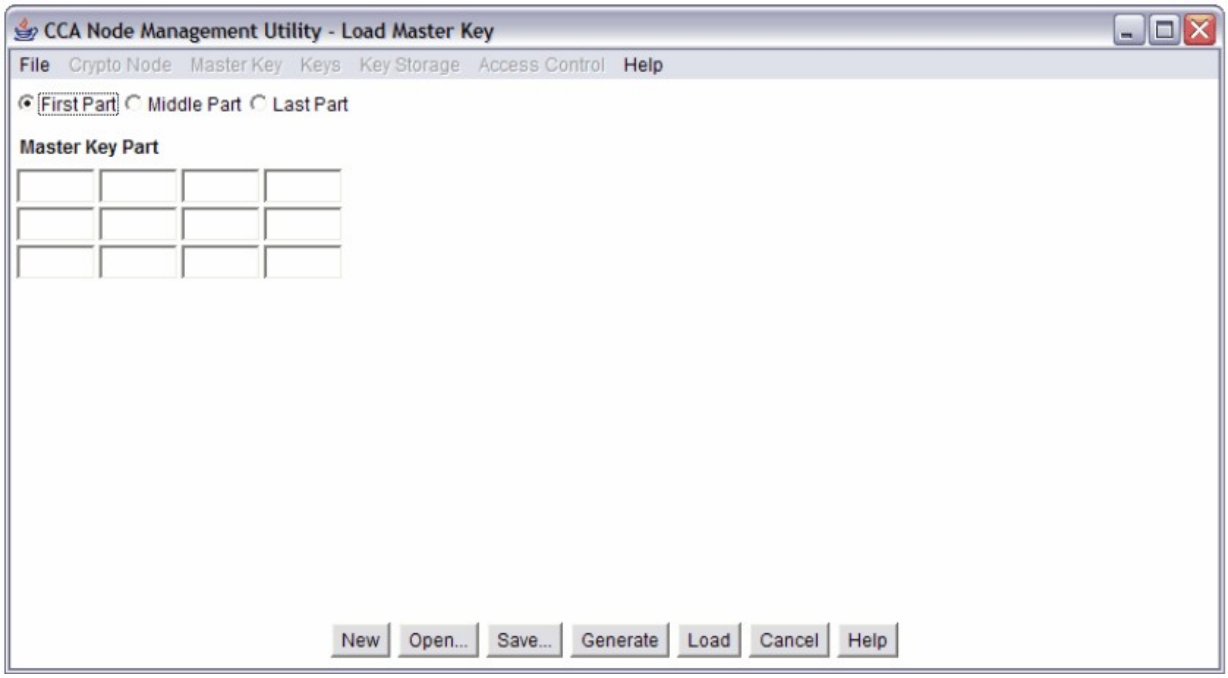

Figure 5. Load Master Key window

2. Select the radio button for the key part you are editing (**First Part**, **Middle Part**, or **Last Part**).
3. Enter data by doing one of the following actions:
   - Click **New** to clear data entered in error.
   - Click **Open** to retrieve preexisting data.
   - Click **Generate** to fill the fields with coprocessor-generated random numbers.
   - Manually enter data into the **Master Key Part** fields. Each field accepts 4 hexadecimal digits.
4. Click **Load** to load the key part into the new master-key register.
5. Click **Save** to save the key part to disk.

   Important: Key parts saved to disk are not enciphered. Consider keeping a disk with key parts on it stored in a safe or vault.

   Note: When you create a key from parts, you must have both the first and last parts. The middle part is optional.
6. Repeat the preceding steps to load the remaining key parts to the new master-key register.

   Note: For the split-knowledge security policy, different people must enter the separate key parts. To enforce a dual control security policy, the access control system must assign the right to enter the first key to one role and the right to enter subsequent key parts to another role. Then, authorized users can log on and enter their respective key part.
7. From the **Master Key** menu, select either **DES/PKA Master Keys** or **AES Master Key**.
8. Click **Set** for the utility to transfer the data:
   a. From the current master-key register to the old master-key register, and to delete the old master key
   b. From the new master-key register to the current master-key register

After setting a new master key, reencipher the keys that are currently in storage.

Related links

"Re-encipher the stored keys" on page 33

Managing key storage

The CNM utility enables basic key storage management functions for keys. These utility functions do not form a comprehensive key management system

Application programs are better suited to perform repetitive key management tasks.

Key storage is a repository of keys that you access by key label using labels that you or your applications define. Data Encryption Standard (DES) keys, Public Key Algorithm (PKA) Rivest-Shamir-Adleman (RSA) keys, and Advanced Encryption Standard (AES) keys are held in separate storage systems. Also the key storage has limited internal storage for PKA keys. The coprocessor stored keys are not considered part of key storage in this discussion.

Notes:

1. If your server has multiple cryptographic coprocessors that are loaded with CCA, those coprocessors must have identical master keys installed for key storage to work properly.
2. The CNM utility displays a maximum of 1,000 key labels. If you have more than 1,000 key labels in key storage, use an application program to manage them.

Creating or initializing key storage

To create or initialize key storage for your Data Encryption Standard (DES) keys, Public-Key Algorithm (PKA) or Advanced Encryption Standard (AES) keys, complete the following steps:

1. From the **Key Storage** menu, select **DES Key Storage**, **PKA Key Storage**, or **AES Key Storage**.
2. From the resulting submenu, click **Initialize**. The Initialize DES Key Storage, Initialize PKA Key Storage, or Initialize AES Key Storage window is displayed.
3. Enter a description for the key-storage file.
4. Click **Initialize**. You are prompted to enter a name for the key-storage data set.
5. Enter a name for the file and save it. The key-storage file is created on the host.

   Note: If a file with the same name exists, you are prompted to verify your choice because initializing the key storage modifies the file; therefore, if the file had any keys, they would be erased.

Re-encipher the stored keys

To re-encipher the keys in storage under a new master key, complete the following steps:

1. From the **Key Storage** menu, select **DES Key Storage**, **PKA Key Storage**, or AES Key Storage.
2. From the resulting submenu, click **Manage**; the DES Key Storage Management, PKA Key Storage Management, or AES Key Storage Management window is displayed. This window panel lists the labels of the keys in storage.
3. Click **Reencipher**. The keys are re-enciphered under the key in the current master-key register.

Deleting a stored key

To delete a stored key, complete the following steps:

1. From the **Key Storage**, click **DES Key Storage**, **PKA Key Storage**, or AES Key Storage.
2. From the resulting submenu, click **Manage**. The DES Key Storage Management, PKA Key Storage Management, or AES Key Storage Management window is displayed. This window lists the labels of the keys in storage.

   You can set the filter criteria to list a subset of keys within storage. For example, if you enter `*.mac` as the filter criterion and refresh the list, the subset is limited to keys with labels that end in .mac. (The asterisk is a wildcard character.)
3. Highlight the key label for the key to be deleted.
4. Click **Delete**. A confirmation message is displayed.
5. Click **Yes**. To confirm that the stored key is deleted.

Creating a key label

To create a key label, complete the following steps:

1. From the **Key Storage** menu, click **DES Key Storage**, **PKA Key Storage**, or **AES Key Storage**.
2. From the resulting submenu, click **Manage**. The DES Key Storage Management, PKA Key Storage Management, or AES Key Storage Management window is displayed. This window lists the labels of the keys in storage.

   You can set the filter criteria to list a subset of keys within storage. For example, if you enter `*.mac` as the filter criterion and refresh the list, the subset is limited to keys that have labels that end in .mac. (The asterisk is a wildcard character.)
3. Click **New**. You are prompted to enter a key label.
4. Click **Load**. The key label is loaded into storage.

Creating and storing primary DES KEKs

Key encrypting keys (KEKs) are encrypted under the Data Encryption Standard (DES) master key and stored in DES key storage for local use.

Key parts used to create a KEK can be randomly generated or entered as clear text information. The parts can also be saved to disk or diskette in clear text for transporting to other nodes or for re-creating the local KEK.

Note: The Cryptographic Node Management (CNM) utility supports only DES KEKs for the transport of keys between nodes. Applications can use the CCA API to furnish the services needed for public-key-based or Advanced Encryption Standard (AES)-based key distribution.

To create and store a primary DES KEK (or other double-length operational key), complete the following steps:

1. From the **Keys** menu, click **Primary DES Key-encrypting keys**. The Primary DES Key-encrypting keys window is displayed.

   At any time, you can click **New** to clear all data fields and reset all the radio buttons to their default settings.
2. Select the radio button for the desired key part to be entered: **First Part**, **Middle Part**, or **Last Part**.
3. Enter data in the **Key Part** fields by doing one of the following actions:
   - Click **Open** to retrieve pre-existing **Key Part**, **Control Vector**, and **Key Label** data that was previously stored on disk by using the **Save** command.
   - Click **Generate** to fill the **Key Part** fields with coprocessor generated random numbers.
   - Manually enter data into the **Key Part** fields. Each of the **Key Part** fields accepts 4 hexadecimal digits.
4. Select a control vector for the key:
   - To use a default KEK control vector, select the appropriate **Default Importer** or **Default Exporter** radio button.
   - To use a custom control vector, select the **Custom** radio button. In the **Control Vector** fields, enter the left or right half of a control vector for any double-length key. Note that the key part bit (bit 44) must be on and that each byte of the control vector must have even parity.

     For detailed information about control vectors, see IBM CCA Basic Services Reference and Guide for the *IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* manual.
5. Enter a key label to identify the key token in key storage.
6. Click **Load** to load the key part into the coprocessor and store the resulting key token into key storage.
7. Click **Save** to save the unencrypted key part and its associated control vector and key label values to disk.

8. **Save** to disk or **Load** to key storage. the remaining key part information by following steps “2” on page 34 - “7” on page 34. Be sure to use the same key label for each part of a single key.

Creating other nodes by using the CNI utility

Creating a CNI list for the CCA Node Initialization (CNI) utility, allows to load keys and access control data stored on disk into other cryptographic nodes without running the CNM utility on those target nodes.

To set up a node using the CNI utility, complete the following steps:

1. Start the CCA Node Management utility by entering the **csufcnm** command. The CNM utility logo and the main panel displays.
2. Save to the host or portable media like a diskette the access control data and keys you want to install on other nodes. When you run the CNI utility on the target node, it searches the identical directory path for each file. For example:
   - If you save a user profile to the established node directory `c:\IBM4764\profiles`, the CNI utility searches the target node directory `c:\IBM4764\profiles`.
   - If you save a user profile to the diskette directory `a:\profiles`, the CNI utility will search the target node directory `a:\profiles`.
3. From the **File** menu, click **CNI Editor**. The CCA Node Initialization Editor window displays as shown in Figure 6 on page 35.

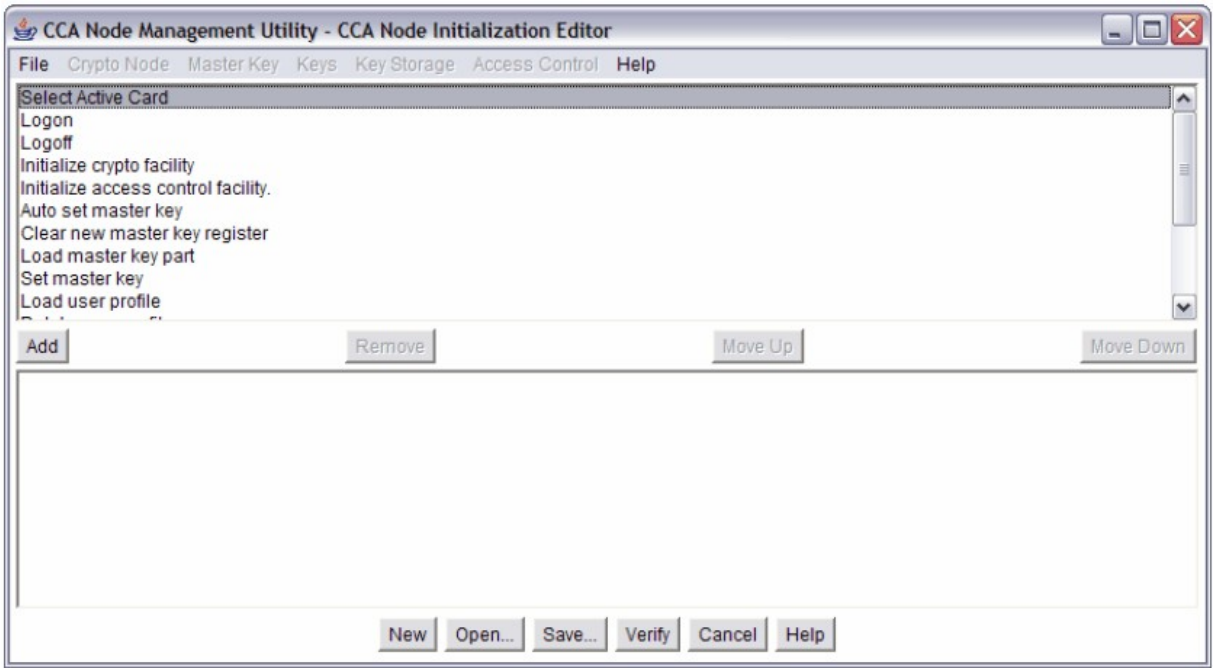

Figure 6. CCA Node Initialization Editor window

The list in the top pane of the window displays the functions that can be added to the CNI list. The bottom pane lists the functions included in the current CNI list. References to master keys in the list refer to the DES and PKA master keys.

4. Add the functions you want. To add a function to the CNI list:
   a. Highlight a function.
   b. Click **Add**. The function is added to the CNI list.

      Note: If the function you choose loads a data object, such as a key part, key-storage file, user profile, or role, you are prompted to enter the file name or the ID of the object to be loaded.
5. Using the **Move Up** and **Move Down** buttons, organize the functions to reflect the same order you follow when using the CNM utility. For example, if you are loading access control data.

6. Click **Verify** to confirm that objects have been created correctly.
7. Click **Save**. You are prompted to select a name and directory location for the CNI list file.
8. Save the CNI list file. The list file does not contain the data objects specified in the CNI list.
9. Copy the files needed by the CNI utility to target host directory locations that mirror their locations on the source host. If you saved the files to portable media, insert the media into the target node.
10. From the target node, run the list using the CNI utility by entering the **csufcni** command.

    If the CNI list includes a logon, enter `csulcni` or `csuncni` on the command line (without specifying a file name). The CNI utility help information describes the syntax for entering an ID and passphrase.

    The CNI utility loads files to the coprocessor from the host or portable media, as specified by the CNI list.

Building applications to use with the CCA API

An application can be build which can be used with the Common Cryptographic Architecture (CCA) API.

Source code for the sample routine is included with the software. You can use the sample included to test the coprocessor and the Support Program.

Note: The file locations referred to in this section are the default directory paths.

Overview of CCA verbs

Application and utility programs issue service requests to the cryptographic coprocessor by calling the CCA verbs. The term *verb* implies an action that an application program can initiate. The operating system code in turn calls the coprocessor physical device driver (PDD). The hardware and software accessed through the API are themselves an integrated subsystem.

Verb calls are written in the standard syntax of the C programming language, and include an entry-point name, verb parameters, and the variables for those parameters.

For a detailed listing of the verbs, variables, and parameters you can use when programming for the CCA security application programming interface (API), see the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors manual*.

Calling CCA verbs in C program syntax

In every operating system environment, you can code CCA API verb calls using standard C programming language syntax.

Function call prototypes for all CCA security API verbs are contained in a header file. The files and their default distribution locations are:

AIX
/usr/include/

To include these verb declarations, use the following compiler directive in your program:

AIX
```
#include "csufincl.h"
```

To issue a call to a CCA security API verb, code the verb entry-point name in uppercase characters. Separate the parameter identifiers with commas and enclose them in parentheses. End the call with a semicolon character. For example:

```
CSNBCKI (&return_code,
         &reason_code,
         &exit_data_length,    /* exit_data_length */
         exit_data,            /* exit_data        */
         clear_key,
         key_token);
```

Note: The third and fourth parameters of a CCA call, *exit_data_length* and *exit_data*, are not currently supported by the CCA Cryptographic Coprocessor Support Program. Although it is permissible to code null address pointers for these parameters, it is preferred that you specify a long integer valued to 0 with the *exit_data_length* parameter.

Compiling and linking CCA application programs

The CCA Cryptographic Coprocessor Support Program includes the C Language source code and the makefile for a sample program.

The file and its default distribution location follows:

AIX
`/usr/lpp/csufx.4765/samples/c.`

Compile application programs that use CCA and link the compiled programs to the CCA library. The library and its default distribution location follows:

AIX
`/usr/lib/libcsufcca.a.`

Sample C routine: Generating a MAC

To illustrate the practical application of CCA verb calls, this topic describes the sample C programming language routine included with the CCA Cryptographic Coprocessor Support Program.

There is also a sample program on the product Web site. That sample program can help you understand the performance of the CCA implementation.

The sample routine generates a message authentication code (MAC) on a text string and then verifies the MAC. To generate and verify the MAC, the routine:

1. Calls the **Key_Generate** (CSNBKGN) verb to create a MAC and MACVER key pair.
2. Calls the **MAC_Generate** (CSNBMGN) verb to generate a MAC on a text string with the MAC key.
3. Calls the **MAC_Verify** (CSNBMVR) verb to verify the text string MAC with the MACVER key.

A sample routine is shown in Figure 7 on page 37, see the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors manual* for the descriptions of the verbs and their parameters. These verbs are listed in the following table.

Table 5. Verbs called by the sample routine

Verb	Entry-point name
Key_Generate	CSNBKGN
MAC_Generate	CSNBMGN
MAC_Verify	CSNBMVR

Figure 7. Sample C routine: generating a MAC,

```
/*********************************************************************/
/*                                                                   */
/* Module Name: mac.c                                                */
/*                                                                   */
/* DESCRIPTIVE NAME: Cryptographic Coprocessor Support Program       */
/*                   C language source code example                  */
/*                                                                   */
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Licensed Materials - Property of IBM                              */
/*                                                                   */
/* (C) Copyright IBM Corp. 1997-2010 All Rights Reserved             */
/*                                                                   */
```

```
/* US Government Users Restricted Rights - Use duplication or       */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/*                                                                  */
/*------------------------------------------------------------------*/
/*                                                                  */
/*        NOTICE TO USERS OF THE SOURCE CODE EXAMPLES               */
/*                                                                  */
/* The source code examples provided by IBM are only intended to    */
/* assist in the development of a working software program. The      */
/* source code examples do not function as written: additional      */
/* code is required. In addition, the source code examples may      */
/* not compile and/or bind successfully as written.                 */
/*                                                                  */
/* International Business Machines Corporation provides the source  */
/* code examples, both individually and as one or more groups,      */
/* "as is" without warranty of any kind, either expressed or        */
/* implied, including, but not limited to the implied warranties of */
/* merchantability and fitness for a particular purpose. The entire */
/* risk as to the quality and performance of the source code        */
/* examples, both individually and as one or more groups, is with   */
/* you. Should any part of the source code examples prove defective, */
/* you (and not IBM or an authorized dealer) assume the entire cost */
/* of all necessary servicing, repair or correction.                */
/*                                                                  */
/* IBM does not warrant that the contents of the source code        */
/* examples, whether individually or as one or more groups, will    */
/* meet your requirements or that the source code examples are      */
/* error-free.                                                      */
/*                                                                  */
/* IBM may make improvements and/or changes in the source code      */
/* examples at any time.                                            */
/*                                                                  */
/* Changes may be made periodically to the information in the       */
/* source code examples; these changes may be reported, for the     */
/* sample code included herein, in new editions of the examples.    */
/*                                                                  */
/* References in the source code examples to IBM products, programs, */
/* or services do not imply that IBM intends to make these          */
/* available in all countries in which IBM operates. Any reference  */
/* to the IBM licensed program in the source code examples is not   */
/* intended to state or imply that IBM's licensed program must be   */
/* used. Any functionally equivalent program may be used.           */
/*                                                                  */
```

```
/*------------------------------------------------------------------*/
/*                                                                  */
/* This example program:                                            */
/*                                                                  */
/* 1) Calls the Key_Generate verb (CSNBKGN) to create a MAC (message */
/*    authentication code) key token and a MACVER key token.        */
/*                                                                  */
/* 2) Calls the MAC_Generate verb (CSNBMGN) using the MAC key token  */
/*    from step 1 to generate a MAC on the supplied text string     */
/*    (INPUT_TEXT).                                                 */
/*                                                                  */
/* 3) Calls the MAC_Verify verb (CSNBMVR) to verify the MAC for the  */
/*    same text string, using the MACVER key token created in       */
/*    step 1.                                                       */
/*                                                                  */
/********************************************************************/
#include <stdio.h>
#include <string.h>

#ifdef _AIX
  #include <csufincl.h>
#elif __WINDOWS__
  #include "csunincl.h"
#else
  #include "csulincl.h"     /* else linux */
#endif

/* Defines */
#define KEY_FORM            "OPOP"
#define KEY_LENGTH          "SINGLE  "
#define KEY_TYPE_1          "MAC     "
#define KEY_TYPE_2          "MACVER  "
#define INPUT_TEXT          "abcdefghijklmn0987654321"
#define MAC_PROCESSING_RULE "X9.9-1  "
#define SEGMENT_FLAG        "ONLY    "
#define MAC_LENGTH          "HEX-9   "
#define MAC_BUFFER_LENGTH   10
```

```
void main()
{
  static long          return_code;
  static long          reason_code;
  static unsigned char key_form[4];
  static unsigned char key_length[8];
  static unsigned char mac_key_type[8];
  static unsigned char macver_key_type[8];
  static unsigned char kek_key_id_1[64];
  static unsigned char kek_key_id_2[64];
  static unsigned char mac_key_id[64];
  static unsigned char macver_key_id[64];
  static long          text_length;
  static unsigned char text[26];
  static long          rule_array_count;
  static unsigned char rule_array[3][8];       /* Max 3 rule array elements  */
  static unsigned char chaining_vector[18];
  static unsigned char mac_value[MAC_BUFFER_LENGTH];

  /* Print a banner */
  printf("Cryptographic Coprocessor Support Program example program.\n");
```

```
  /* Set up initial values for Key_Generate call */
  return_code = 0;
  reason_code = 0;
  memcpy (key_form,        KEY_FORM,   4);     /* OPOP key pair              */
  memcpy (key_length,      KEY_LENGTH, 8);     /* Single-length keys         */
  memcpy (mac_key_type,    KEY_TYPE_1, 8);     /* 1st token, MAC key type    */
  memcpy (macver_key_type, KEY_TYPE_2, 8);     /* 2nd token, MACVER key type */
  memset (kek_key_id_1,  0x00, sizeof(kek_key_id_1));  /* 1st KEK not used   */
  memset (kek_key_id_2,  0x00, sizeof(kek_key_id_2));  /* 2nd KEK not used   */
  memset (mac_key_id,    0x00, sizeof(mac_key_id));    /* Init 1st key token */
  memset (macver_key_id, 0x00, sizeof(macver_key_id)); /* Init 2nd key token */

  /* Generate a MAC/MACVER operational key pair */
  CSNBKGN(&return_code,
          &reason_code,
          NULL,                                /* exit_data_length           */
          NULL,                                /* exit_data                  */
          key_form,
          key_length,
          mac_key_type,
          macver_key_type,
          kek_key_id_1,
          kek_key_id_2,
          mac_key_id,
          macver_key_id);

  /* Check the return/reason codes. Terminate if there is an error.          */
  if (return_code != 0 || reason_code != 0) {
    printf ("Key_Generate failed: ");             /* Print failing verb      */
    printf ("return_code = %ld, ",  return_code); /* Print return code       */
    printf ("reason_code = %ld.\n", reason_code); /* Print reason code       */
    return;
  }
  else
    printf ("Key_Generate successful.\n");
```

```
  /* Set up initial values for MAC_Generate call */
  return_code = 0;
  reason_code = 0;
  text_length = sizeof (INPUT_TEXT) - 1;          /* Length of MAC text      */
  memcpy (text, INPUT_TEXT, text_length);         /* Define MAC input text   */
  rule_array_count = 3;                           /* 3 rule array elements   */
  memset (rule_array, ' ', sizeof(rule_array));   /* Clear rule array        */
  memcpy (rule_array[0], MAC_PROCESSING_RULE, 8); /* 1st rule array element  */
  memcpy (rule_array[1], SEGMENT_FLAG,        8); /* 2nd rule array element  */
  memcpy (rule_array[2], MAC_LENGTH,          8); /* 3rd rule array element  */
  memset (chaining_vector, 0x00, 18);             /* Clear chaining vector   */
  memset (mac_value, 0x00, sizeof(mac_value));    /* Clear MAC value         */

  /* Generate a MAC based on input text */
  CSNBMGN ( &return_code,
           &reason_code,
           NULL,                                  /* exit_data_length        */
           NULL,                                  /* exit_data               */
           mac_key_id,                            /* Output from Key_Generate*/
           &text_length,
           text,
```

```
            &rule_array_count,
            &rule_array[0][0],
            chaining_vector,
            mac_value);

  /* Check the return/reason codes. Terminate if there is an error.          */
  if (return_code != 0 || reason_code != 0) {
    printf ("MAC Generate Failed: ");             /* Print failing verb      */
    printf ("return_code = %ld, ",  return_code); /* Print return code       */
    printf ("reason_code = %ld.\n", reason_code); /* Print reason code       */
    return;
  }
  else {
    printf ("MAC_Generate successful.\n");
    printf ("MAC_value = %s\n", mac_value);    /* Print MAC value (HEX-9)    */
  }
```

```
  /* Set up initial values for MAC_Verify call */
  return_code = 0;
  reason_code = 0;
  rule_array_count = 1;                        /* 1 rule array element       */
  memset (rule_array, ' ', sizeof(rule_array));/* Clear rule array           */
  memcpy (rule_array[0], MAC_LENGTH, 8);       /* Rule array element         */
                                               /*  (use default Ciphering    */
                                               /*  Method and Segmenting     */
                                               /*  Control)                  */
  memset (chaining_vector, 0x00, 18);          /* Clear the chaining vector  */

  /* Verify MAC value */
  CSNBMVR (&return_code,
           &reason_code,
           NULL,                               /* exit_data_length           */
           NULL,                               /* exit_data                  */
           macver_key_id,                      /* Output from Key_Generate   */
           &text_length,                       /* Same as for MAC_Generate   */
           text,                               /* Same as for MAC_Generate   */
           &rule_array_count,
           &rule_array[0][0],
           chaining_vector,
           mac_value);                         /* Output from MAC_Generate   */

  /* Check the return/reason codes. Terminate if there is an error.          */
  if (return_code != 0 || reason_code != 0) {
    printf ("MAC_Verify failed: ");               /* Print failing verb      */
    printf ("return_code = %ld, ", return_code);  /* Print return code       */
    printf ("reason_code = %ld.\n", reason_code); /* Print reason code       */
    return;
  }
  else                                         /* No error occurred          */
    printf ("MAC_Verify successful.\n");
}
```

Enhancing throughput with CCA and the 4765 coprocessor

When you use the CCA API, the characteristics of your host application program will affect performance and throughput of the 4765. For best performance on the 4765 coprocessor, evaluate and design your application based on multithreading and multiprocessing, and based on caching Data Encryption Standard (DES), Public-Key Algorithm (PKA), and Advanced Encryption Standard (AES) keys.

Multithreading and multiprocessing

The CCA application running inside the 4765 can process several CCA requests simultaneously. The coprocessor contains several independent hardware elements, including the Rivest-Shamir-Adleman algorithm (RSA) engine, Data Encryption Standard (DES) engine, CPU, random-number generator, and Peripheral Component Interconnect-X (PCI-X) communications interface. These elements can all be working at the same time, processing parts of different CCA verbs. By working on several verbs at the same time, the coprocessor can keep all of its hardware elements busy, maximizing the overall system throughput.

To take advantage of this capability, your host system must send multiple CCA requests to the coprocessor without waiting for each one to finish before sending the next one. The best way to send multiple requests is to design a multithreaded host application program, in which each thread can

independently send CCA requests to the coprocessor. For example, a web server can start a new thread for each request it receives over the network. Each of these threads will send the required cryptographic requests to the coprocessor, independent of what the other threads are doing. The multithreaded model guarantees that the coprocessor is not under used. Another option is to have several independent host application programs all using the coprocessor at the same time.

Caching DES, PKA, and AES keys

The CCA software for the 4765 keeps copies of recently used DES, PKA, and encrypted (not clear text) AES keys in caches inside the secure module. The keys are stored in a form that has been decrypted and validated, and is ready for use. If the same key is reused in a later CCA request, the 4765 can use the cached copy and avoid the overhead associated with decrypting and validating the key token. In addition, for retained PKA keys, the cache eliminates the overhead of retrieving the key from the internal flash Erasable Programmable Read Only Memory (EPROM) memory.

As a result, applications that reuse a common set of keys can run much faster than those that use different keys for each transaction. Most common applications use a common set of DES keys, PKA private keys, and encrypted AES keys, and the caching is effective in improving throughput. PKA public keys and AES clear keys, which have little processing overhead, are not cached.

Initial default-role commands

The characteristics of the default role after the coprocessor is initialized and when no other access control data exists are described. Also, the enabled access control commands are listed.

For the initial default role commands, the role ID is the default and the authentication strength is zero. The default role is valid at all times of the day and on all days of the week. The only functions permitted are those necessary to load access control data.

Important: The cryptographic mode is not secure when unauthenticated users can load access control data by using the default role. Restrict these commands to selected supervisory roles.

Table 6 on page 41 lists the access control commands that are enabled in the default role when the CCA software is initially loaded and when the CCA node is initialized.

Table 6. Initial default-role commands

Code	Command name
X'0107'	One-Way Hash, SHA-1
X'0110'	Set Clock
X'0111'	Reinitialize Device
X'0112'	Initialize Access Control System
X'0113'	Change User Profile Expiration Date
X'0114'	Change User Profile Authentication Data
X'0115'	Reset User Profile Logon-Attempt-Failure Count
X'0116'	Read Public Access Control Information
X'0117'	Delete User Profile
X'0118'	Delete Role
X'0119'	Load Function-Control Vector
X'011A'	Clear Function-Control Vector

Machine-readable log contents

The CLU utility creates two log files, one intended for reading and the other for possible input to a program.

The machine-readable (MRL) log file, contains the binary outputs from the coprocessor in response to various commands submitted to the coprocessor.

Detailed information about the contents of the MRL is available from IBM 4764 and IBM 4765 development. Contact IBM by using the Support and downloads tab in the IBM product website at *http://www.ibm.com/security/cryptocards*.

Device driver error codes

The coprocessor device driver monitors the status of its communication with the coprocessor and the coprocessor hardware-status registers.

Each time that the coprocessor is reset and the reset is not caused by a fault or tamper event, the coprocessor runs through a miniboot, its power-on self-test (POST), code loading, and status routines. During this process, the coprocessor attempts to coordinate with a host-system device driver. Coprocessor reset operations can occur because of power-on, a **reset** command sent from the device driver, or because of coprocessor internal activity such as completion of code updates.

The coprocessor fault or tamper-detection circuitry can also reset the coprocessor.

Programs such as the Coprocessor Load Utility (CLU) and the CCA Support Program can receive unusual status in the form of a 4-byte return code from the device driver.

The possible 4-byte codes, are of the form X'8xxxxxxx'. The codes that are frequently obtained are described in Table 7 on page 42. If you encounter codes of the form XX'8340xxxx' or X'8440xxxx', and the code is not in the table,contact the IBM cryptographic team through email from the Support page on the IBM product website at *http://www.ibm.com/security/cryptocards*.

Table 7. Device-class driver error codes in X'8xxxxxxx' class

4-byte return code (hex)	Reason	Descriptions
8040FFBF	External intrusion	The intrusion arises due to optional electrical connection to the coprocessor. This condition can be reset.
8040FFDA	Dead battery	The batteries have been allowed to run out of sufficient power or have been removed. The coprocessor is zeroized and is no longer functional.
8040FFDB	X-ray tamper or dead battery	The coprocessor is zeroized and is no longer functional.
8040FFDF	X-ray or dead battery	The coprocessor is zeroized and is no longer functional.
8040FFEB	Temperature tamper	The high or low temperature limit has been exceeded. The coprocessor is zeroized and is no longer functional.
8040FFF3	Voltage tamper	The coprocessor is zeroized and is no longer functional.
V8040FFF9	Mesh tamper	The coprocessor is zeroized and is no longer functional.
8040FFFB	Reset bit is on	Low voltage was detected, the internal operating temperature of the coprocessor went out of limits, or the host driver sent a reset command. Try removing and reinserting the coprocessor into the PCI-X bus.

Table 7. Device-class driver error codes in X'8xxxxxxx' class (continued)		
4-byte return code (hex)	**Reason**	**Descriptions**
8040FFFE	Battery warning	The battery power is marginal. For the procedure to be followed to replace the batteries, see the *IBM 4764 PCI-X Cryptographic Coprocessor Installation Manual*.
804xxxxx (for example, **80400005)**	General communication problem	Except for the prior X'8040xxxx' codes, additional conditions arose in host-coprocessor communication. Determine that the host system in fact has a coprocessor. Try removing and reinserting the coprocessor into the PCI-X bus. Run the CLU status command (ST). If problem persists, contact contact the IBM cryptographic team through email from the Support page on the IBM product website at *http://www.ibm.com/security/cryptocards*.
8340xxxx	Miniboot-0 codes	This class of return code arises from the lowest-level of reset testing. If codes in this class occur, contact the IBM cryptographic team through email from the Support page on the IBM product website at *http://www.ibm.com/security/cryptocards*.
8340038F	Random-number generation fault	Continuous monitoring of the random-number generator has detected a possible problem. There is a small statistical probability of this event occurring without indicating an actual ongoing problem. Run the CLU status (ST) command at least twice to determine whether the condition can be cleared.
8440xxxx	Miniboot-1 codes	This class of return code arises from the replaceable POST and code-loading code.
844006B2	Invalid signature	The signature on the data sent from the CLU utility to miniboot could not be validated by the miniboot. Be sure that you are using an appropriate file (for example, **CR1** xxxxx.clu versus **CE1** xxxxx.clu). If the problem persists, obtain the output of a CLU status report and forward the report with a description of the task you want to achieve to the IBM cryptographic team through email from the Support page on the IBM product website at *http://www.ibm.com/security/cryptocards*.

Cloning a master key

This section provides instructions for cloning a master key and provides access control considerations while cloning.

Overview of cloning a master key

The cloning procedure outlines how to clone a master key from one coprocessor to another coprocessor by using the Cryptographic Node Management (CNM) utility.

Note: Ensure that the CNM utility is at the same level on all systems involved in the cloning procedure.

The master-key cloning procedure makes no assumption about which server contains the coprocessors used for:

- Share administration (**SA node**)

- Master-key source (**CSS** coprocessor share-signing node)
- Master-key target (**CSR** coprocessor share-receiving node)

Note: Cloning of AES master keys is not supported.

The SA key can reside in the same coprocessor as either the CSS or the CSR key, or it can reside in a separate coprocessor node. Any of the coprocessors can reside together in the same sever if multiple coprocessors with CCA are available.

The procedure ignores operator actions to log on and log off, because these steps depend on the specific roles in use at your installation. You can switch between coprocessors when you are using more than one coprocessor within a server.

The procedure is divided into several phases as outlined in Table 8 on page 44.

Table 8. Master-key cloning procedure phase overview

Phase	Node	Task
1	SA	Establish the share administration node. Create the SA database, generate the SA key, and store its public key and hash into the SA database.
2a	Source	Establish the source node. Generate the CSS key and add the public key to the SA database. Install the SA public key.
2b	SA	Certify the CSS key and store the certificate into the SA database.
For each target node, repeat phase 3 procedures.		
3a	Target	Establish the target node. Create a CSR database, generate a CSR key, and add the public key to the CSR database for this node. Install the SA public key.
3b	SA	Certify the CSR key and store the certificate into the CSR database for the target node.
3c	Source	Obtain shares and the current master-key verification information.
3d	Target	Install shares and confirm the new master-key. Set the master key.

Before starting the master-key cloning procedure, it is suggested that you complete the forms found in table Table 9 on page 44 and Figure Figure 8 on page 46.

Table 9. Cloning responsibilities, profiles, and roles

Task	Node	Profile	Role	Responsible individual
Audit access controls	SA			
Generate SA key	SA			
Register SA-key hash	SA			
Register SA key	SA			
Audit access controls	CSS			
Generate CSS key	CSS			
Obtain CSS master key	CSS			
Register SA-key hash	CSS			

Table 9. Cloning responsibilities, profiles, and roles (continued)

Task	Node	Profile	Role	Responsible individual
Register SA key	CSS			
Certify CSS key	SA			
Audit access controls	CSR1			
Generate CSR key	CSR1			
Register SA-key hash	CSR1			
Register SA key	CSR1			
Certify CSR1 key	SA			
Obtain shares	CSS			
Install shares	CSR1			
Verify CSR new	CSR1			
Set CSR master key	CSR1			
Audit access controls	CSR2			
Generate CSR key	CSR2			
Register SA-key hash	CSR2			
Register SA key	CSR2			
Certify CSR2 key	SA			
Obtain shares	CSS			
Install shares	CSR2			
Verify CSR new	CSR2			
Set CSR master key	CSR2			

NODE INFORMATION

Node	Machine	Selector Number	Coprocessor Serial Number	Data Base Path and Name
SA Node Control				(sa.db)
CSS Node Source				(sa.db)
CSR Node Target 1				(csr1.db)
CSR Node Target 2				(csr2.db)

SA-KEY HASH

NUMBER OF SHARES

Minimum: "m"	Maximum: "n"

SHARES DISTRIBUTION

Obtained from:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Installed into CSR-1:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Obtained from:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Installed into CSR-2:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 8. Cloning information worksheet

Phase 1 for cloning a master key: Establishing the share administration node

To use the coprocessor as the share administration (SA) node, follow the steps from cloning the master key mentioned in Table 10 on page 46. This coprocessor can also serve as the master key source node or a master key target node.

Prerequisites: Before running this procedure, familiarize yourself with the steps described in the section "Scenario: Cloning a DES or PKA master key" on page 20 and the chapter about understanding and managing master keys in the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* manual.

To establish the SA node, complete the steps in the following table:

Table 10. Cloning the master key procedure: Establishing the SA node

Phase	Task	√
1.1	Audit the appropriateness of the access controls.	
1.2	Perform time synchronization and ensure that the authorization (`fcv_td4kECC521.crt`) is installed.	
1.3	Confirm (or install) the master key.	
1.4	Using the facilities of your operating system, erase any prior SA database from the SA database media.	

Table 10. Cloning the master key procedure: Establishing the SA node (continued)		
Phase	**Task**	**√**
1.5	If not already established, enter the environment ID (EID) by completing the following steps: • Click **Crypto Node** >**Set environment ID**. • Enter the EID, click **Load**.	
1.6	Generate the SA key: • Click **Crypto Node** >**Share Administration** >**Create Keys** >**Share Administration Key**. • Accept the default SA public key and private key labels, and enter the location and name of the SA database (`sa.db`). • Click **Create**. • Record the SA-key hash value for use later in the procedure.	
1.7	Register the SA public key hash: • Click **Crypto Node** >**Share Administration** >**Create Keys** >**Share Administration Key** >**Register Share Administration Key** > **SA-Key Hash**. • Enter the SA database file name and location, click **Next**. • Enter the SA public key label (or accept the default). • Enter the SA-key hash, click **Register**.	
1.8	Register the SA public key: • Click **Crypto Node** >**Share Administration** >**Create Keys** >**Share Administration Key** >**Register Share Administration Key** > **SA-Key Hash**. • Enter the SA database file name and location, click **Next**. • Enter the SA public key label (or accept the default), click Register.	

Phase 2 for cloning a master key: Establishing the source node

Using the coprocessor designated as the master key source node, follow the steps for cloning the master key mentioned in the Table 11 on page 47. This coprocessor can also serve as the SA node.

Table 11. Cloning the master key: Establishing the source (CSS) node		
Phase	**Task**	**√**
2a.1	Audit the appropriateness of the access controls.	
2a.2	Perform time synchronization and ensure that the `fcv_td4kECC521.crt`authorization is installed.	
2a.3	Confirm the coprocessor serial number: • Click **Crypto Node** >**Status**. • Click **Adapter**. • Note the coprocessor serial number, click **Cancel**.	
2a.4	Confirm (or install) the master key.	

Table 11. Cloning the master key: Establishing the source (CSS) node (continued)		
Phase	**Task**	**√**
2a.5	Obtain the current master key verification information: • Click **Master Key** > **Verify** > **Current**. • Click **Save** to transport media, click **Cancel**.	
2a.6	If not already established, enter the environment ID (EID): • Click **Crypto Node** > **Set environment ID**. • Enter the EID, click **Load**.	
2a.7	If not already established, set the number m and n shared values: • Click **Crypto Node** > **Share Administration** > **Set Number of Shares**. • Set the maximum and minimum number of required shares, click **Load**.	
2a.8	Generate the CSS key: • Click **Crypto Node** > **Share Administration** > **Create Keys** > **CSS Key**. • Enter the CSS key label (for example, CSS.KEY). • Confirm the coprocessor serial number. • Confirm or enter the SA database name and location. • Click **Create**.	
2a.9	Register the SA public-key hash: • Click **Crypto Node** > **Share Administration** > **Register Share Administration Key** > **SA-Key Hash**. • Enter the SA database file name and location, click **Next**. • Enter the SA public key label (or accept the default). • Enter the SA key hash, click **Register**.	
2a.10	Register the SA public-key: • Click **Crypto Node** > **Share Administration** > **Register Share Administration Key** > **SA Key**. • Enter the SA database file name and location, click **Next**. • Enter the SA public key label (or accept the default), click **Register**.	

Phase 3 for cloning a master key: Establishing the target node and cloning a master key

Using the designated nodes, establish the target node and clone the master key following the steps for cloning the master key mentioned in Table 12 on page 49. This coprocessor can also serve as the SA node.

Table 12. Cloning a master key: Establishing the CSR node, and cloning a master key			
Phase	**Node**	**Task**	**√**
At the target node			
3a.1	Target	Audit the appropriateness of the access controls.	
3a.2	Target	Perform time synchronization and ensure that the `fcv_td2k.crt` authorization is installed.	
3a.3	Target	Confirm the coprocessor serial number: • Click **Crypto Node** > **Status**. • Click **Adapter**. • Note the coprocessor serial number, click **Cancel**.	
3a.4	Target	Ensure the existence of a (temporary) master key.	
3a.5	Target	If not already established, enter the environment ID (EID): • Click **Crypto Node** > **Set environment ID** > **Crypto Node**. • Enter the EID (for example, CSR1 NODE and extend with spaces to 16 entered characters). • Click **Load**.	
3a.6	Target	If not already established, set the number *m* and *n* shares values: • Click **Crypto Node** > **Share Administration** > **Set Number of Shares**. • Set the maximum and minimum number of required shares. • Click **Load**.	
3a.7	Target	Using the facilities of your operating system, erase the `csr.db` data file.	
3a.8	Target	Generate the CSR key: • Click **Crypto Node** > **Share Administration** > **Create Keys** > **CSR Key**. • Enter the CSR key label (for example, CSR1.KEY). • Confirm the coprocessor serial number. • Select the key size. • Provide the CSR database name and location (for example, CSR1.DB). • Click **Create**.	
3a.9	Target	Register the SA public-key hash: • Click **Crypto Node** > **Share Administration** > **Register Share Administration** > **SA-Key Hash**. • Enter the SA database file name and location, click **Next**. • Enter the SA public key label (or accept the default). • Enter the SA key hash, click **Register**.	

Table 12. Cloning a master key: Establishing the CSR node, and cloning a master key (continued)			
Phase	**Node**	**Task**	**✓**
3a.10	Target	Register the SA public-key: • Click **Crypto Node** > **Share Administration** > **Register Share Administration** > **SA Key**. • Enter the SA database file name and location, click **Next**. • Enter the SA public key label (or accept the default), click **Register**.	
At the SA node			
3b.1	SA	Certify the CSS key (as required): • Click **Crypto Node** > **Share Administration** > **Certify Keys** > **CSS Key**. • Enter the name and path for the SA database, click **Next**. • Confirm the CSS key label, the coprocessor serial number, and the SA environment ID. • Click **Certify**.	
3b.2	SA	Certify the CSR key: • Click **Crypto Node** > **Share Administration** > **Certify Keys** > **CSS Key**. • Enter the name and path for the SA and CSR databases, click **Next**. • Confirm the SA key label, CSR key label, and the SA environment ID. • Enter the CSR serial number. • Click **Certify**.	
At the source node			
3c.1	Source	Obtain at least the number of m and n shares. Perform the following substep for each share. Note that logon and logoff might be required to obtain each share. • Click **Crypto Node** > **Share Administration** > **Get Share**. • Select the share. Note that if you are obtaining an additional set(s) of shares, the Distributed messages might not be meaningful. • Enter the name and path for the SA and CSR databases, click **Next**. • Confirm the CSS key label, CSS coprocessor serial number, and the CSR coprocessor serial number. • Click **Get Share**. Repeat as required.	

Table 12. Cloning a master key: Establishing the CSR node, and cloning a master key (continued)			
Phase	**Node**	**Task**	**✓**
At the target node			
3d.1	Target	Install the number of m and n shares. Perform the following for each share and observe the response. The response indicates when enough shares have been installed to form the new master key. Note that logon and logoff might be required to install each share. • Click **Crypto Node** > **Share Administration** > **Load Share**. • Select the share. • Enter the name and path for the CSR and SA databases, click **Next**. • Confirm the CSS key label, the CSS coprocessor serial number, and the CSR coprocessor serial number. • Click **Load Share**. Observe the response. Loading sufficient shares completes the new master-key. Repeat as required.	
3d.2	Target	Confirm the new master key: • Click **Master Key** > **Verify** > **New**. • Click **Compare** or **select the file** or click **OK** or **click Cancel**	
3d.3	Target	Erase the `csr.db` data file. This is not a security problem but rather to avoid complications while doing master key cloning operation.	
3d.4	Target	As appropriate, set the master key: • Click **Master Key** > **Set**. • Click **OK**.	

Access control considerations when cloning

There are three classes of roles to consider for cloning operations.

- Roles at the share administration (SA) node.
- Roles at the source node: coprocessor share signing (CSS) node
- Roles at the target node: coprocessor share signing (CSS) node

Your security policy must define who will have the authority to:

- Generate a random master key at the source node.
- Set the master key, the action which brings a new master key into operation. When the master key changes, the keys enciphered by the master key must be updated.
- Generate the retained Rivest-Shamir-Adleman (RSA) keys to certify the public keys of the source and target nodes (the SA key), and to generate the retained keys at the source (CSS) and target (CSR) nodes.
- Register the SA key and its hash and determine whether it will be a split responsibility.

In addition, you must decide how many nodes must cooperate to clone a master key. Of course, this must be selected to avoid collusion.

In deciding the m and n values, consider when the cloning will take place and whether you need to reconstitute the master key from a fewer number of shares than the total number obtained from the source node (perhaps because of share corruption or the unavailability of one or more individuals who can obtain or install a share).

Note: The cryptographic node management (CNM) utility places all of the shares from a node in the `csr.db` file. Each share is encrypted under a unique, triple-length data encryption standard (DES) key which itself is encrypted by the CSR public key of the target node.

Table 13 on page 52 provides guidance for selecting the permissions applicable to the roles that are related to cloning.

Table 13. CCA commands related to master key cloning

Code	Command name	Verb name	Consideration
X'001A'	Set Master Key	**Master_Key_Process**	Critical. This role must have knowledge of the contents of the new master key register and the implications of a master key change.
X'001D'	Compute Verification Pattern	**Many**	All
X'0020'	Generate Random Master Key	**Master_Key_Process**	Not critical except that it fills the new master key register.
X'0032'	Clear New Master Key Register	**Master_Key_Process**	This role is assigned to the role that can set the master key. The role can override the collected shares. It must be mutually exclusive with the Generate Random Master Key command.
X'0033'	Clear Old Master Key Register	**Master_Key_Process**	Generally not used.
X'008E'	Generate Key	**Key_Generate** **Random_Number_Generate**	All
X'0090'	Reencipher to Current Master Key	**Key_Token_Change**	This role depends on who will update the working keys encrypted by the master key.
X'0100'	PKA96 Digital Signature Generate	**Digital_Signature_Generate**	This role certifies the SA, CSS, and CSR keys.
X'0101'	PKA96 Digital Signature Verify	**Digital_Signature_Verify**	All
X'0102'	PKA96 Key Token Change	**PKA_Key_Token_Change**	This role depends on who will update the working keys encrypted by the master key.
X'0103'	PKA96 PKA Key Generate	**PKA_Key_Generate**	This role is required to generate the SA, CSS, and CSR keys.
X'0107'	One-Way Hash, SHA-1	**One_Way_Hash**	All
X'0114'	Change User Profile Authentication Data	**Access_Control_Initialization**	This role allows to change the passphrase in any profile. Use with discretion.

Table 13. CCA commands related to master key cloning (continued)			
Code	**Command name**	**Verb name**	**Consideration**
X'0116'	Read Public access control Information	**Access_Control_Maintenance**	All
X'011C'	Set EID	**Cryptographic_Facility_Control**	This role is required to set up the CSS and CSR nodes.
X'011D'	Initialize Master Key Cloning	**Cryptographic_Facility_Control**	This role is required to set up the m of n values at the CSS and CSR nodes.
X'0200'	PKA Register Public Key Hash	**PKA_Public_Key_Hash_Register**	This role must be used at the CSS and CSR nodes to ensure the SA key can be recognized. Split responsibility with X'0201'.
X'0201'	PKA Public Key Register	**PKA_Public_Key_Register**	This role must be used at the CSS and CSR nodes to ensure the SA key can be recognized. Split responsibility with X'0200'.
X'0203'	Delete Retained Key	**Retained_Key_Delete**	This role is used to remove obsolete SA, CSS, and CSR keys. Be careful about denial of service.
X'0204'	PKA Clone Key Generate	**PKA_Key_Generate**	This role is required to generate the CSS and CSR keys.
X'0211' - X'021F'	Clone-info (Share) Obtain	**Master_Key_Distribution**	This role is assigns a profile and role for each share to enforce split responsibility.
X'0221' - X'022F'	Clone-info (Share) Install	**Master_Key_Distribution**	This role is assigns a profile and role for each share to enforce split responsibility.
X'0230'	List Retained Key	**Retained_Key_List**	All

Threat considerations for a digital-signing server

Consider various threats when you deploy the IBM 4765 with the IBM Common Cryptographic Architecture (CCA) Support Program in a digital-signing application. Much of the discussion is applicable to other environments in which you might apply the coprocessor.

An organization placing a certification authority (CA), registration authority (RA), Online Certificate Status Protocol (OCSP) responder, or time-stamping service into operation needs to consider how its installation will address various threats. Table 14 on page 54 lists potential threats and presents product design and implementation solutions to many of these threats. Notes describe steps that you need to consider to further mitigate your exposure to problems.

See IBM CCA Basic Services Reference and Guide for the *IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* manual describes actions you can use in deploying the coprocessor, policies to consider, application functions to be included.

Read the contents of Table 14 on page 54 after you have made initial decisions about your installation.

Table 14. Threat considerations for a digital-signing server	
Threat discussion	**Threat mitigation**
Threats associated with physical attack on the coprocessor	
Physical probing of the coprocessor An adversary might perform physical probing of the coprocessor to reveal design information and operational contents. Such probing might include electrical functions but is referred to here as physical because it requires direct contact with the coprocessor internal functions. Physical probing might entail reading data from the coprocessor through techniques commonly employed in IC failure analysis and IC reverse-engineering efforts. The goal of the adversary is to identify such design details as hardware security mechanisms, access control mechanisms, authentication systems, data-protection systems, memory partitioning, or cryptographic programs. Determination of software design, including initialization data, passwords, PINs, or cryptographic keys might also be a goal.	The coprocessor electronics incorporate a sophisticated set of active tamper-detection sensors and response mechanisms. High and low temperature, voltage levels and sequencing, radiation, and physical penetration sensors are designed to detect unusual environmental situations. All of the sensitive electronics are enclosed in a physically shielded package. Upon detecting a potential tamper event, the coprocessor immediately clears all internal RAM memory, which also zeroizes keys used to recover sensitive, persistent data from flash memory. An independent state controller is also reset, which indicates that the coprocessor is no longer in a factory-certified condition. The various tamper sensors are powered from the time of coprocessor manufacture through the end of life of the coprocessor. The coprocessor digitally signs a query response that you can verify to confirm that the coprocessor is genuine and is not tampered with. Almost all of the software that runs on the main processor within the coprocessor is available on the web and is therefore subject to reverse engineering. However, the coprocessor validates the digital signatures on code it is requested to accept so that the code modified by an adversary cannot be loaded into the coprocessor. The public keys used to validate offered code is destroyed when a tamper event is recognized. The design and implementation is being independently evaluated and certified by the USA NIST under the FIPS PUB 140-2 Level 4 standard. **Note:** You must validate the condition of the coprocessor and the code content.
Physical modification of the coprocessor An adversary might physically modify the coprocessor to reveal design or security-related information. This modification might be achieved through techniques commonly employed in hardware failure analysis and reverse engineering efforts. The goal is to identify such design details as hardware-security mechanisms, access control mechanisms, authentication systems, data protection systems, memory partitioning, or cryptographic programs. Determination of software design, including initialization data, passwords, or cryptographic keys, might also be a goal.	The sensitive electronics are all packaged within the tamper responding package mounted on the coprocessor. In the process of altering the sensitive electronics, the coprocessor factory certification would be destroyed, rendering the device useless. **Note:** Confirm that a specific, serial numbered coprocessor is in use and audit its status-query response to confirm that it remains an unaltered IBM coprocessor loaded with appropriate software.

Table 14. Threat considerations for a digital-signing server (continued)	
Threat discussion	**Threat mitigation**
Environmental manipulation of the coprocessor An adversary might use environmental conditions beyond those of the coprocessor specification to obtain or modify data or program flow for fraudulent coprocessor use. This modification might include manipulation of power lines, clock rates, or exposure to high and low temperatures and radiation. As a result, the coprocessor might get into a situation where instructions are not correctly executed. As a result, security-critical data might get modified or disclosed in contradiction to the security requirements for the coprocessor.	The coprocessor has sensors to detect environmental stresses that might induce erroneous operation. Abnormal conditions can cause the unit to zeroize.
Substituted process Requests to, and responses from, the coprocessor might be directed to an alternative implementation enabling an adversary to influence results. An alternative implementation might be substituted with differing security features. For example, private key generation and the production of digital signatures might be performed in an alternative implementation that would enable exposure of the private key.	**Notes:** 1. Auditors need to complete the processes described for them to ensure that the signing key is indeed retained within the appropriate coprocessor. 2. Access to the host system should be supervised so that host system security measures and correct operation can be relied on.
Threats associated with logical attack on the coprocessor	
Insertion of faults An adversary might determine security critical information through observation of the results of repetitive insertion of selected data. Insertion of selected input followed by monitoring the output for changes is a relatively well-known attack method for cryptographic devices. The intent is to determine information based on how the coprocessor responds to the selected input. This threat is distinguished by the deliberate and repetitive choice and manipulation of input data as opposed to random selection or manipulation of the physical characteristics involved in input or output operations.	The electronic design of the coprocessor renders classical approaches to smart card attacks infeasible. **Note:** Supervision of the host system and controlling access to the system, both logically and physically, are important security steps to be taken by an organization.
Forced reset An adversary might force the coprocessor into a nonsecure state through inappropriate ending of selected operations. Attempts to generate a nonsecure state in the coprocessor might be made through premature ending of transactions or communications between the coprocessor and the host, by insertion of interrupt function, or by inappropriate use of interface functions.	The coprocessor is designed to always run through its initial power on sequence in the event of trap and reset conditions. Each application level request is treated as a separate unit of work and processed from a single defined set of initial conditions.

Table 14. Threat considerations for a digital-signing server (continued)

Threat discussion	Threat mitigation
Invalid input An adversary or authorized user of the coprocessor might compromise the security features of the coprocessor through the introduction of invalid input. Invalid input might take the form of operations that are not formatted correctly, requests for information beyond register limits, or attempts to find and execute undocumented commands. The result of such an attack might be a compromise in the security functions, a generation of exploitable errors in operation, or the release of protected data.	Transaction requests carry authentication information applied in the caller's domain and validated by the coprocessor. Each request is processed from a single, known state with predefined conditions. The coprocessor software validates the characteristics of each request to address misuse scenarios.
Data loading malfunction An adversary might maliciously generate errors in setup data to compromise the security functions of the coprocessor. During the stages of coprocessor preparation, which involve loading the coprocessor with special keys, identification of roles, and so forth, the data itself might be changed from the intended information or might be corrupted. Either event could be an attempt to penetrate the coprocessor security functions or to expose the security in an unauthorized manner.	**Note:** As outlined in auditor procedures, the access control setup should be verified along with confirming the installed coprocessor software.
Unauthorized program loading An adversary might use unauthorized programs to penetrate or modify the security functions of the coprocessor. Unauthorized programs might include the execution of legitimate programs not intended for use during normal operation or the unauthorized loading of programs specifically targeted at penetration or modification of the security functions.	The coprocessor only accepts digitally signed software after the signature has been validated. An independent evaluation of IBM's software build and signing procedures and the coprocessor design affirms the trust that can be placed in the identity of loaded software. **Note:** An auditor should follow procedures to affirm that specified software is in use.
Threats associated with control of access	
Invalid access A user or an adversary of the coprocessor might access information or services without having permission as defined in the role profile. Each role has defined privileges that allow access only to selected services of the coprocessor. Access beyond those specified services could result in exposure of secure information.	An auditor can confirm the permissions granted in each established role and the set of user profiles associated with each role. An independent evaluation of the coprocessor software implementation and testing has reviewed the integrity of the access control implementation.

Table 14. Threat considerations for a digital-signing server (continued)	
Threat discussion	**Threat mitigation**
Fraud on first use An adversary might gain access to coprocessor information by unauthorized use of a new, not yet installed, coprocessor. An adversary might try to get access to a coprocessor during or directly after the manufacturing process and load fraudulent software into the coprocessor or modify critical data stored within the coprocessor during the manufacturing and factory initialization process before it is shipped to the customer.	IBM's manufacturing and distribution practice ensures that prior to factory certification the end user of a coprocessor is unknown and unassigned. Factory installed software is validated through checking of digital signatures. **Notes:** 1. The standard installation bring up process replaces all of the runtime coprocessor software. 2. You should ensure that Segments 2 and 3 are unowned prior to loading coprocessor software for production. This action ensures that no residual data remains to influence subsequent operations.
Impersonation An adversary might gain access to coprocessor information or services by impersonating an authorized user of the coprocessor. The coprocessor is required to define certain roles including the required authentication mechanism and the services the role is allowed to use. An adversary might try to impersonate an authorized user, operating within a defined, to get access to information or perform services allowed for the authorized user.	The two user classes follow: 1. (IBM) Coprocessor code signer: An independent evaluation of IBM's procedure for building and signing code assures that legitimate code can be identified by a user's auditor. 2. The CCA access control design protects the integrity and confidentiality of a user access control passphrase from the domain of the user process into the coprocessor. The correct passphrase and profile identification grant use of a role. **Note:** Host system security, host system application design, and administrative policies are required to assure that a designated user's passphrase is secure.
Threats associated with unanticipated interactions	

Table 14. Threat considerations for a digital-signing server (continued)

Threat discussion	Threat mitigation
Use of disallowed application functions An adversary might exploit interactions between applications to expose sensitive coprocessor or user data. Interactions might include the execution of commands that are not required or allowed in the specific application being performed. Examples include the use of functions related to master key management or functions related to symmetric encryption or financial services. Those functions should not have any negative impact on the coprocessor functions required for the digital signing application.	The coprocessor design requires you to configure the access control setup. The CCA software has been examined to ensure that functions are disallowed when required commands are not enabled. **Notes:** 1. Your access control configuration should follow the principles discussed in Appendix H of the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* Redbooks publication such that only the functions needed for the operational phase can be invoked in this phase. 2. For the digital signing application, establish guidelines for a set of roles with very limited capabilities and a setup sequence that restricts the coprocessor functionality to that essential for digital signing. In some installations, it might be desirable to accommodate a different approach to roles or to consider the functions of additional applications, or both. In these cases, ensure that you review the guidelines and observations in Appendix H of the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* Redbooks publication for applicability to your circumstances.
Threats regarding cryptographic functions	
Cryptographic attack An adversary might defeat security functions through a cryptographic attack against the algorithm or through a brute force attack. This attack might include either signature generation and verification functions or random number generators.	The coprocessor implements well established and standardized cryptographic functions. The random-number generation implementation has been subjected to extensive evaluation under criteria published by the USA NIST and the German Information Security Agency (German Bundesamt fur f³r Sicherhert in der Informations Technik or German BSI). The secrecy afforded retained private keys is the subject of an independent evaluation. These design and implementation steps provide assurance against cryptographic attack. **Note:** For a digital signing server, see the guidelines in Appendix H of the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* Redbooks publication.
Threats regarding digital signatures	

Table 14. Threat considerations for a digital-signing server (continued)

Threat discussion	Threat mitigation
Forging signed data An adversary might modify data digitally signed by the coprocessor such that this modification is not detectable by the signatory nor a third party. This attack might use weaknesses in the secure hash function, weaknesses in the signature encoding, or weaknesses in the cryptographic algorithm used to generate a forged signature.	The coprocessor implements well established and standardized cryptographic functions. **Notes:** 1. Precautions in the use of CCA should be observed as documented in Appendix H of the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* Redbooks publication. 2. Users should maintain an awareness of vulnerabilities discussed in (open) forums regarding the strength of cryptographic algorithms and processes they employ.
Forging data before it is signed An adversary might modify data to be digitally signed by the coprocessor before the signature is generated within the coprocessor. This attack might use weaknesses in the implementation that allow an adversary to modify data transmitted for signature to the coprocessor before the coprocessor actually calculates the signature.	Requests from user host-application process memory carry an integrity check value that the coprocessor confirms prior to incorporating the hash in a digital signature. **Note:** Users must review host-system and host-application program security to ensure that authenticated hash values received into the coprocessor have not been compromised and are representative of the data to be protected.
Misuse of signature function An adversary might misuse the coprocessor signature creation function to sign data that the coprocessor is not supposed to sign. The adversary might try to submit data to the coprocessor and get it signed without passing the authorization checks of the coprocessor that perform before generating a digital signature. As an alternative, an adversary might try to modify data within the coprocessor through the use of coprocessor functions or by trying to influence the coprocessor such that the data in the coprocessor gets modified.	An independent review of the coprocessor software is expected to affirm that: • The digital signature generation service requires an appropriate permission in a role. • The processing of requests and the integrity of the design prevent data alteration. **Notes:** 1. The integrity of the coprocessor and its code must be affirmed by an auditor who reviews a coprocessor status query. 2. An auditor must confirm that appropriate access control roles and profiles have been established that exclude unauthorized users from use of the digital signing function.

Table 14. Threat considerations for a digital-signing server (continued)

Threat discussion	Threat mitigation
Forging signature-verification function An adversary might modify the function for signature verification such that a false signature is accepted as valid. This attack might try to modify the signature verification function or signed data to be verified such that the coprocessor returns a success message when this false signature is presented for verification.	The signature-verification function of primary interest here occurs in the coprocessor's code loading process (in Miniboot). With this product: • Miniboot code, like the control program and (CCA) application program code, is only accepted into the coprocessor when the coprocessor validates the signature on the signed code. • The initial Miniboot code loaded in the factory is also subject to digital signature verification. • Standardized cryptographic processes are used (SHA-1, RSA, ISO 9796) for the signature. • The code building and signing process are the subject of an independent review.
Disclosure of a private RSA signature key An adversary might use functions that disclose a private RSA signature key.	An independent evaluation is expected to affirm that the CCA Support Program does not contain any function to output or reveal the value of a retained private key. Certified evaluations are expected to demonstrate that the control program does not output data retained in coprocessor persistent storage nor is there any lower-level function to read such storage.
Deleting a private RSA signature key An adversary might use a function that deletes a private RSA signature key without being authorized to do so and without physically tampering with the coprocessor.	Independent evaluations are expected to affirm that a retained private key is only deleted in the following circumstances: 1. Under CCA control with the Retained_Key_Delete verb 2. By loading the coprocessor CCA software* 3. By removing the coprocessor CCA software 4. By causing a tamper event **Notes:** To address these exposures takes these actions: 1. Selectively enable the Delete Retained Key command, X'0203'. 2. Use host system access controls to manage use of the CLU. 3. Manage physical access to the coprocessor. * Reloading the coprocessor software with a file such as CEXxxxxx.clu does not zeroize the contents of persistent storage. The file CNWxxxxx.clu will zeroize persistent storage. See “Loading and Unloading software into the coprocessor” on page 6.
Threats that monitor information	

Table 14. Threat considerations for a digital-signing server (continued)

Threat discussion	**Threat mitigation**
Information leakage An adversary might make use of information that is leaked from the coprocessor during normal use. Information leakage might occur through emanations, variations in power consumption, I/O characteristics, clock frequency, or by changes in processing time requirements. This leakage might be interpreted as a covert channel transmission but is more closely related to measurement of operating parameters, which might be derived either from direct (contact) measurements or measurement of emanations and can then be related to the specific operation being performed.	Practical means to interpret information leakage are the subject of ongoing research in commercial and governmental laboratories. An in-depth defense should include limiting access to the cryptographic environment and restrictions on the use of specialized equipment in and near the cryptographic environment.
Linkage of multiple observations An adversary might observe multiple uses of resources or services and, by linking these observations, deduce information that would reveal critical security information. The combination of observations over a period of many uses of the coprocessor, or the integration of knowledge gained from observing different operations, might reveal information that allows an adversary to either learn information directly or to formulate an attack that could further reveal information that the coprocessor is required to keep secret.	**Notes:** 1. Use of the cryptographic equipment should be controlled, including following the guidelines in Appendix H of the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* Redbooks publication. 2. An adversary might well have access to the signed data and signatures, so controls should be put in place to limit a user's ability to submit arbitrary signing requests. 3. The use of standardized cryptographic procedures and monitoring of the cryptographic community's understanding of the vulnerabilities of these processes (SHA-1, RSA, ISO 9796, X9.31, HMAC, and triple-DES) can provide assurance of secure operation.
Miscellaneous threats	
Linked attacks An adversary might perform successive attacks with the result that the coprocessor becomes unstable or some aspect of the security functions is degraded. A following attack might then be successfully executed. Monitoring outputs while manipulating inputs in the presence of environmental stress is an example of a linked attack.	**Notes:** 1. Use of the cryptographic system should be limited to authorized situations enforced through the coprocessor access controls and through use of host system controls. 2. Host-system controls and organizational policies should restrict the access to the system for monitoring and the submission of arbitrary requests.

Table 14. Threat considerations for a digital-signing server (continued)	
Threat discussion	**Threat mitigation**
Repetitive attack An adversary might utilize repetitive undetected attempts at penetration to expose memory contents or to change security critical elements in the coprocessor. Repetitive attempts related to some or all of the other threats discussed herein might be used to iteratively develop an effective penetration of the coprocessor security. If these attacks can, in all cases, remain undetected, there will be no warning of increased vulnerability.	**Note:** Use of the cryptographic system should be limited to authorized situations enforced through the coprocessor access controls and through use of host system controls. Host system controls and organizational policies should restrict the access to the system for monitoring and the submission of arbitrary requests.
Cloning An adversary might clone part or all of a functional coprocessor to develop further attacks. The information necessary to successfully clone part or all of a coprocessor might derive from a detailed inspection of the coprocessor itself or from illicit appropriation of design information.	**Note:** Auditors must confirm that the digital signing key, appropriate code, and access control regime is resident in the authorized coprocessor.
Threats addressed by the operating environment	
Coprocessor modification and reuse An adversary might use a modified coprocessor to masquerade as an original coprocessor so that information assets can be fraudulently accessed. Removal, modification, and re-insertion of that coprocessor into a host system could be used to pass such a combination as an original. This might then be used to access or change the private signature keys or other security critical information to be protected.	**Notes:** 1. An auditor must confirm through examination of a coprocessor signed query response that the device is genuine and that the appropriate code is loaded. 2. The auditor must also confirm that the digital signing key is a retained key in the coprocessor.
Abuse by privileged users A careless, willfully negligent, or hostile administrator or other privileged user might create a compromise of the coprocessor assets through execution of actions that expose the security functions or the protected data. A privileged user or administrator could directly implement or facilitate attacks based on any of the threats described here.	**Note:** An organization must establish, enforce, and audit policies that limit the access that a single individual has to the cryptographic system. The setup procedure must ensure that a single user does not have the opportunity to bring an inappropriate system into production.
Data modification Data to be signed by the coprocessor might be modified by an adversary or by faults in the operational environment after it has been approved by the legitimate user, but before the data is submitted to the coprocessor to be signed. Data that has been approved by the legitimate user to be signed might be modified by an adversary, by false or malicious programs, or by environmental errors (for example, transmission errors) after the data has been approved by the legitimate user and before the data is transferred to the coprocessor to be signed.	**Note:** Host system security precautions and organization policies must be defined, enforced, and audited to thwart such attacks.

Table 14. Threat considerations for a digital-signing server (continued)	
Threat discussion	**Threat mitigation**
Data verification Signed data to be verified by the coprocessor might be modified by an adversary or by faults in the operational environment before it is submitted to the coprocessor for signature verification such that the response of the coprocessor does not reflect the validity of the signature. Signed data submitted by a user might be modified within the coprocessor environment before it is passed to the coprocessor for verification. This might result in a response from the coprocessor that does not reflect the actual validity of the digital signature that should be verified. There is also the possibility that the response of the coprocessor is modified in the coprocessor environment before it is passed to the user that requested the signature verification.	The coprocessor verifies the signature on code and certain code loading commands. An independent evaluation is expected to confirm that this cannot be bypassed. The CCA design supports validation of the integrity of requests and responses between the coprocessor and the top layer of CCA code in the host system. **Note:** Host-system security measures must address blocking the modification of request inputs and outputs.

IBM Cryptographic Coprocessor notices

IBM Cryptographic Coprocessor notices includes 3 notices that provide guidelines for safe disposal of electronic components.

Product recycling and disposal

This unit contains materials such as circuit boards, cables, electromagnetic compatibility gaskets and connectors that might contain lead and copper/beryllium alloys that require special handling and disposal at end of life. Before this unit is disposed of, these materials must be removed and recycled or discarded according to applicable regulations. IBM offers product-return programs in several countries. Information on product recycling offerings can be found on IBM Internet site at http://www.ibm.com/ibm/environment/products/prp.shtml IBM encourages owners of information technology (IT) equipment to responsibly recycle their equipment when it is no longer needed. IBM offers a variety of programs and services to assist equipment owners in recycling their IT products. Information on product recycling offerings can be found on IBM's Internet site at:

http://www.ibm.com/ibm/environment/products/prp.shtml

Notice: This mark applies only to countries within the European Union (EU) and Norway. Appliances are labeled in accordance with European Directive 2002/96/EC concerning waste electrical and electronic equipment (WEEE). The Directive determines the framework for the return and recycling of used appliances as applicable throughout the European Union. This label is applied to various products to indicate that the product is not to be thrown away, but rather reclaimed upon end of life per this Directive.

Battery return program

This product may contain sealed lead acid, nickel cadmium, nickel metal hydride, lithium, or lithium ion battery. Consult your user manual or service manual for specific battery information. The battery must be recycled or disposed of properly. Recycling facilities may not be available in your area. For information on disposal of batteries outside the United States, go to http://www.ibm.com/ibm/environment/products/batteryrecycle.shtml or contact your local waste disposal facility. In the United States, IBM has established a return process for reuse, recycling, or proper disposal of used IBM sealed lead acid, nickel cadmium, nickel metal hydride, and other battery packs from IBM Equipment. For information on proper disposal of these batteries, contact IBM at 1-800-426-4333. Please have the IBM part number listed on the battery available prior to your call.

For Taiwan: Please recycle batteries.

IBM Cryptographic Coprocessor card return program

This machine may contain an optional feature, the cryptographic coprocessor card which includes a polyurethane material that contains mercury. Please follow Local Ordinances or regulations for disposal of this card. IBM has established a return program for certain IBM Cryptographic Coprocessor cards. More information can be found at:

http:/www.ibm.com/ibm/environment/products/prp.shtml

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide 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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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 actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year).

Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as the customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Index

N

O

P

R

S

T

U

V

Z