AIX Version 7.1

IBM

4765 PCIe Cryptographic Coprocessor AIX CCA Support Program Installation Manual 4.1

SC23-6772-00

AIX Version 7.1

IBM

4765 PCIe Cryptographic Coprocessor AIX CCA Support Program Installation Manual 4.1

SC23-6772-00

Note

Before using this information and the product it supports, read the information in “Notices” on page 59.

First Edition (September 2010)

This edition applies to AIX Version 6.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to Information Development, Department 04XA-905-6B013, 11501 Burnet Road, Austin, Texas 78758-3400. To send comments electronically, use this commercial Internet address: pserinfo@us.ibm.com. Any information that you supply may be used without incurring any obligation to you.

Contents

About this document v

Audience v
Organization of this publication v
Related Publication vi
IBM 4765 PCI-e Cryptographic Coprocessor publications vi

4765 PCIe Cryptographic Coprocessor AIX CCA Support Program Installation Manual 4.1 1

Installation process overview 1
  Summary 1
Obtaining coprocessor hardware and software . . . 2
  Ordering and Installing coprocessor hardware . . 2
Installing the Support Program 3
  Support Program components 3
  Installing and Removing coprocessor host software 3
  Installing and Uninstalling the Support Program for AIX 3
  AIX requirements. 4
  Special considerations 6
Loading software into the coprocessor 7
  Loading coprocessor software. 7
  Unloading coprocessor software and zeroize the CCA node 10
  Coprocessor Load Utility reference 11
Using the CNM and CNI utilities to manage the cryptographic node. 15
  Overview 16
  Using the utilities, sample scenarios 16
  Using CNM Utility Administrative Functions . . 20
  Creating and Managing access-control data. . . 21
  Managing Cryptographic Keys 26
  Using the CNI utility to establish other nodes . . 33
Building applications to use with the CCA API . . 35
  Overview 35
  Calling CCA verbs in C program syntax. . . . 36
  Compiling and Linking CCA application programs 36
  Sample C routine 36
  Enhancing throughput with CCA and the 4765 coprocessor 40
Initial DEFAULT-role commands 41
Machine-readable log contents 41
Device driver error codes. 42
Master-key cloning 43
  Master-key cloning procedure 43
  Access-control considerations when cloning . . 49
Threat considerations for a digital-signing server . . 50
  Product recycling and disposal 57
  Battery return program 57
  IBM Cryptographic Coprocessor card return program 58

Notices 59

Trademarks 61

Index 63

About this document

This installation manual describes Release 4.10 of the IBM® Common Cryptographic Architecture (CCA) Support Program (Support Program) for the IBM 4765 PCI-e Cryptographic Coprocessor. The Support Program includes device drivers, utilities, and the CCA coprocessor code.

Use this manual to help with the following tasks:
- Obtain the Support Program through the Internet
- Load the software onto a host computer and into the coprocessor(s)
- 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 key storage 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 consideration 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

Organization of this publication

- "Installation process overview" on page 1 summarizes the installation and the operation of the .
- "Obtaining coprocessor hardware and software" on page 2 describes how to obtain the PCI-e cryptographic coprocessor hardware and the .
- "Installing the Support Program" on page 3 describes how to install the software onto the host computer.
- "Loading software into the coprocessor" on page 7 describes how to load the operating system and the CCA software into the PCI-e cryptographic coprocessor.
- "Using the CNM and CNI utilities to manage the cryptographic node" on page 15 describes how to use the and the utilities to set up and manage cryptographic nodes.
- "Building applications to use with the CCA API" on page 35 explains how to build applications for CCA, and how to link them to CCA libraries.
- "Initial DEFAULT-role commands" on page 41 details the permissions granted to the DEFAULT role when the is initialized.
- "Machine-readable log contents" on page 41 details the content of the machine-readable log created by the Coprocessor Load Utility.
- "Device driver error codes" on page 42 provides error-code information that can be observed when operating the CLU utility.
- "Master-key cloning" on page 43 provides a procedure for master-key cloning.
- "Threat considerations for a digital-signing server" on page 50 addresses threats to consider when employing the IBM 4765 and the Support Program in a digital-signing application.
- "Notices" on page 59 provides legal notices.

Related Publication

The list below reflects source information regarding the PCI-e Cryptographic Coprocessor and commercial cryptographic applications in general.

IBM 4765 PCI-e Cryptographic Coprocessor publications

For availability of these publications, check the Library page of the Web site at *http://www.ibm.com/security/cryptocards/pcixcc/ordersoftware.shtml*. From the Web site, you can download, view, and print publications available in the Adobe® Acrobat** portable document format (PDF):

- *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCI-e and the IBM 4764 PCI-X Cryptographic Coprocessors*
- *IBM 4765 PCI-e Cryptographic Coprocessor Installation Manual*

4765 PCIe Cryptographic Coprocessor AIX CCA Support Program Installation Manual 4.1

To use the information effectively, you must be familiar with commands, system calls, subroutines, file formats, and special files. This topic is also available on the documentation CD that is shipped with the operating system.

To view or download the PDF version of this topic, select 4765 PCIe Cryptographic Coprocessor AIX CCA Support Program Installation Manual 4.1.

Downloading Adobe Reader: You need Adobe Reader installed on your system to view or print this PDF. You can download a free copy from the Adobe website (www.adobe.com/products/acrobat/readstep.html).

Installation process overview

This topic summarizes the installation and operation procedures discussed in this manual and provides a checklist for you to use while installing the PCIe Cryptographic Coprocessor and the CCA Cryptographic Coprocessor Support Program. See Table 1 on page 2.

Summary

The IBM CCA Cryptographic Coprocessor Support Program consists of several components, and includes:
- 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)
- Utility applications that run on the host machine into which the coprocessor has been installed

An FCV is a signed value provided by IBM. Its use originated to enable the CCA application within the coprocessor to yield a level of cryptographic service consistent with applicable cryptographic implementation import and export regulations.

To install these components and to establish a CCA cryptographic node, perform the following steps described in this manual:
1. **Obtain the hardware and software**: “Obtaining coprocessor hardware and software” on page 2 describes how to order the hardware from IBM, how to download the software through the Internet, and how to install the downloaded files.
2. **Install the software on the host**: “Installing the Support Program” on page 3 describes how to install the downloaded software onto the coprocessor host computer.
3. **Load the coprocessor software**: “Loading software into the coprocessor” on page 7 describes how to load both the embedded operating system and the CCA application program.
4. **Set up the cryptographic node**: You can establish a CCA cryptographic node using the utilities provided with the Support Program, or by linking your application programs to the CCA API. You should also verify the access control and other setup requirements imposed by application software you plan to use with the IBM 4765. The CCA Node Management (CNM) utility described in “Using the CNM and CNI utilities to manage the cryptographic node” 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 KEKs
- Manage the storage of data keys
- Create lists ("scripts") for the CCA Node Initialization (CNI) utility

5. **Link application programs to the CCA libraries**: "Building applications to use with the CCA API" on page 35 describes how to build applications for CCA and how to link them to the CCA libraries.

Table 1. Activity checklist, CCA Cryptographic Coprocessor Support Program installation

Step	Task	Reference	✓
1	Decide which platform support packages are appropriate to your setup: ( ) AIX 5.3 or AIX 6.1	"Ordering and Installing coprocessor hardware"	
2	Place an order with IBM or your IBM Business Partner.	"Placing orders for IBM 4765 coprocessor" on page 3	
3	Receive coprocessor hardware.		
4	Download the Support Program for your operating system.	"Obtaining the software" on page 3	
5	Install the Support Program onto the coprocessor host computer.	"Installing the Support Program" on page 3	
6	Install coprocessor hardware.	"Installing your IBM 4765 hardware" on page 3	
7	Load coprocessor software.	"Loading software into the coprocessor" on page 7	
8	Set up a CCA test node. Review the first pages in "Using the CNM and CNI utilities to manage the cryptographic node" on page 15, and then set up a test node.	"Establishing a test node" on page 17	
9	Run test programs that utilize the CCA libraries.		

Obtaining coprocessor hardware and software

This topic describes how to obtain coprocessor hardware and software.

The IBM CCA Cryptographic Coprocessor Support Program features are available for download through the Internet on the product order page. This topic describes:
- How to choose, order, and install coprocessor hardware
- How to download the software

Ordering and Installing coprocessor hardware

The following sections describe how to:
- Order coprocessors
- Replace batteries
- Place orders for the IBM 4765 hardware
- Install your IBM 4765 hardware

Ordering coprocessors

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

The software will support up to eight coprocessors per system (dependent 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 chosen.

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

Installing your IBM 4765 hardware

The IBM 4765 is installed in a manner similar to other PCI-e boards. Follow the process described in the *IBM 4765 PCIe Cryptographic Coprocessor Installation Manual*

Obtaining the software: The software can be obtained by downloading it from this Web site: *http://www.ibm.com/security/cryptocards/pcixcc/ordersoftware.shtml*

Installing the Support Program

This topic describes how to install CCA Cryptographic Coprocessor Support Program onto the coprocessor host computer.

After downloading the software as described in "Obtaining coprocessor hardware and software" on page 2, follow the procedures in this topic to install the CCA Cryptographic Coprocessor Support Program onto the coprocessor host computer. (Loading software into the coprocessor is described in "Loading software into the coprocessor" on page 7 and initializing the CCA application within the coprocessor is described in "Using the CNM and CNI utilities to manage the cryptographic node" on page 15.)

This chapter:

- Lists the Support Program components that you are installing
- Describes how to install and uninstall coprocessor host software
- Discusses special security considerations

Support Program components

The Coprocessor Load Utility and software files necessary to load the embedded operating system and the CCA application program into the coprocessor is described in "Using the CNM and CNI utilities to manage the cryptographic node" on page 15.

Installing and Removing coprocessor host software

For each operating system, the following sections:

- Lists system requirements and procedures to install the Support Program.
- Describe how to install the Support Program.
- Describe how to remove the Support Program.

After you have installed the software as described in this chapter, you are ready to install the coprocessor software; see Chapter 4,"Loading software into the coprocessor" on page 7.

Note that all components of the installed CCA package might not indicate the same version number. The version number for a software component is updated only when the component is changed for a release. Unchanged components retain the version number from the last time that they were modified. For example, the Cryptographic Coprocessor Support Software might be at a different level from the coprocessor device driver.

Installing and Uninstalling the Support Program for AIX

This section includes a description of the Support Program system requirements and procedures necessary to install and uninstall the Support Program.

Important

The installation process requires root-level authority and must be performed by a system administrator with that authority.

AIX requirements

Before you install the Support Program, ensure that your system meets the following requirements:

Hardware

An IBM System p® (RS/6000®) server on the IBM ServerProven® list with an available 4765 PCI-e cryptographic coprocessor feature.

During installation of the software, the driver interacts with the coprocessor to arbitrate interrupt settings, DMA channels, and other system resources. For installation instructions regarding the coprocessor hardware and device driver, refer to the IBM 4765 PCI-e Cryptographic Coprocessor Installation Manual.

Software

1. Install IBM AIX Version 5.3 or 6.1™
2. Install a Java™ Runtime Environment (JRE) 1.4.2 or later that is required to run the CCA Node Management (CNM) utility.
3. Install the software package **(csufx.4765)** that is downloaded from the Web site. The software package contains the following file sets:
   - **csufx.4765.com** - 4765 CCA Support Program
   - **csufx.4765.cca** - 4765 Support Program - Common Utilities
   - **csufx.4765.man** - Support Program Man Pages

Installing the Support Program base release 4.1

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

To install the Support Program:

1. Enter the command **smitty install_all.**
2. Enter the location of the install images you obtained using the procedure described in "Obtaining the software" on page 3. Press Enter.
3. Enter **csufx.4765.cca csufx.4765.man** in the SOFTWARE install field or press F4 to select from the list. Verify that AUTOMATICALLY install requisite software is set to yes and ACCEPT new license agreements is set to yes. Use the tab key to toggle or the F4 key to list. Press Enter and press Enter again to continue when prompted ARE YOU SURE.
4. Exit from **smitty** using the F10 key.
5. Read or print `/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."

Configuring the Support Program

The following utilities and system command are available to configure the software. For more details, refer to the AIX man page.

csufadmin

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

Default permissions restrict 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 associated with the CCA libraries.

The default permissions restrict 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 install program defines, in the AIX object data manager (ODM), the following default directories:

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

Use the csufkeys utility to change the storage locations.

Note: When you initialize key storage using the Cryptographic Node Management utility, ensure that you specify the ODM directories defined by this utility; see How to create or initialize key storage on page 5-20

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 parmname attributes specify the following values:

- csudesds - The file containing the DES key-records
- csupkads - The file containing the PKA key-records
- csuaesds - The file containing the AES key-records
- csudesld - The directory containing the DES key-record-list files
- csupkald - The directory containing the PKA key-record-list files
- csuaesld - The directory containing the AES 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 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**, and **/usr/lpp/csufx.4765/csufkeys/pkalist,** **/usr/lpp/csufx.4765/csufkeys/aeslistdirectories**, respectively. These are the default directory names. You can modify the directory names when installing the software. The list files are created under the ownership, if you request for the list service. Make sure 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 two directories. See the g+s flags in the **chmod** command for more information. If this procedure is not followed, then 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 CRP0n, 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.

CCA Support Program and AIX file permissions

The CCA Support Program relies on file permissions at the group level to function correctly. This means that 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 in order to be fully functional and run without errors.

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

For proper operation, the key-storage files and directories must have a group ID of the application user group; that is, the groupname parameter used when the **csufappl** utility was run.

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

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

Reviewing System p (RS/6000) coprocessor hardware errors

Errors occurring in the coprocessor hardware are 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, Crypt0), depending on which CCA Coprocessor log you wish to view. The Coprocessor Load Utility is explained in Chapter 4, ôLoading software into the coprocessor.ö

Removing the Support Program

If your key-storage files are located in the default directories, back up or save them before you remove the support program; removing the software deletes those key-storage files located in the default directories. For a list of the default directories, see ôHow to configure the Support Programö on page 3-2.

To remove the Support Program:

1. Log on as **root**.
2. Enter the command **rmdev -dl Crypt0**; the coprocessor device driver and otherrelated information are removed. Repeat this command for each CCA coprocessor you plan to remove or relocate, changing **Crypt0** to **Crypt1**, Crypt2, and so forth as needed.
3. Enter the command **smitty install_remove**; you are prompted to enter the product names.
4. Enter the product names csufx.4765.com and devices.pciex.14107a0314107b03.rte.
5. Verify that the REMOVE dependent software value is NO. Also, verify that the Preview Only value is NO.
6. Press the Enter key.

Special considerations

The 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, you should set the file permission of HIKM.zip to read and execute only. Likewise, to protect the data file of access-control points, set the file permission of csuap.def to read only.

Loading software into the coprocessor

This topic describes how to use the Coprocessor Load Utility (CLU) to load the coprocessor operating system and CCA application into the coprocessor

After installing the Support Program on the host computer, as described in "Installing the Support Program" on page 3, 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 utility to reload the necessary program segments. You can also load software from other vendors using the CLU utility.

This chapter includes:

- Instructions for using the CLU utility to understand what coprocessors are installed and their status, and to install and uninstall the software that runs within the coprocessor
- A reference section describing:
  - The coprocessor memory segments
  - Validating the coprocessor status
  - The syntax used to invoke the CLU utility
  - CLU-utility 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 Web site Library page at *http://www.ibm.com/security/cryptocards*.

Note:

1. The file locations referenced in this chapter are the default directory paths.
2. "Device driver error codes" on page 42 describes error codes returned by the coprocessor device driver. These are often presented in the form of a hexadecimal number such as X'8040xxxx'. You might encounter some of these error situations, 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 utility described in "Using the CNM and CNI utilities to manage the cryptographic node" on page 15.

Loading coprocessor software

This section provides the procedures you use in loading software into the coprocessor. Refer to the readme.txt file that accompanies the software distribution you are installing for specific .clu file names. The readme.txt file might also provide additional information that amplifies or modifies these general procedures.

You will be instructed to follow this sequence of steps:

1. At a command prompt, change to the directory with the CLU files
2. Determine the software 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 CLU

You will need to locate the directory that contains the coprocessor code files (*.clu) and possibly the CLU utility. At a command prompt, change to the directory for the code files. If the CLU utility is not in this directory, ensure that your operating system can locate the CLU utility. The default coprocessor code directory is /usr/lpp/csufx.4765/clu.

To run the CLU utility, you enter the program name at the command prompt, csuclu.

You can provide parameters interactively to the CLU utility, or you can include these on the command line input. (See "Coprocessor Load Utility syntax" on page 12.) 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 2 log files. If the log files do not exist, they will be 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 you for the "number" of the coprocessor with which you intend to interact. These numbers ("coprocessor_#") range from 0 to 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. (See Figure 2 on page 14.)

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

Important: When using CLU, no applications that use CCA should be running.

Determining coprocessor software segment contents

The coprocessor has three "segments": Segment 1, Segment 2, and Segment 3 (see Table 2). Each segment:

- Has a status
- Holds software
- Holds a validation public key
- Has an owner identifier (except Segment 1)

Table 2. 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 using the **ST** command. Figure 1 on page 9 shows a typical ST response.

```
======================================================================
CSULCLU V3.10 st.log ST 0 begun Tue Sep 16 14:44:00 2008
*********** Command ST started. ---- Tue Sep 16 14:44:30 2008

 *** VPD data; PartNum = 12R8565
 *** VPD data; EC Num = 0J14210
 *** VPD data; Ser Num = 95000329
 *** VPD data; Description = IBM 4764-001 PCI-X Cryptographic Coprocessor
 *** VPD data; Mfg. Loc. = 91
 *** ROM Status; PIC ver: 28, ROM ver: 8a59
 *** ROM Status; INIT: INITIALIZED
 *** ROM Status; SEG2: RUNNABLE , OWNER2: 2
 *** ROM Status; SEG3: RUNNABLE , OWNER3: 2
 *** Page 1 Certified: YES
 *** Segment 1 Image: 3.30 POST1v2.16 MB1v1.25 FPGAv78    20050215060133000000220000000000000000000000
 *** Segment 1 Revision: 330
 *** Segment 1 Hash: C16F 4102 0989 51FE 4FA8 44B6 3B3A 274E 04F4 3DF0
 *** Segment 2 Image: 3.30.05 Linux OS MCP v1.6          20080906210733000000000000000000000000000000
 *** Segment 2 Revision: 330
 *** Segment 2 Hash: D5FB 0F2C E64F EB7E B7A9 ED72 C648 610A D71E 3CA2
 *** Segment 3 Image: 3.30.05 CCA                        20080907122933000000000000000000000000000000
 *** Segment 3 Revision: 330
 *** Segment 3 Hash: 85E0 045F 21AC AA93 6280 600B 1FBE DD1E 78F1 75EB
 *** Query Adapter Status successful ***
Obtain Status ended successfully!
*********** Command ST ended. ---- Tue Sep 16 14:45:29 2008
```

Figure 1. Typical CLU status response

Definitions of the fields on the ST response are:

Item **Discussion**

PartNum
The part number (P/N) of the coprocessor. It can be either 12R8565 or 41U0441.

EC Num
The engineering change number of the coprocessor.

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

Description
A statement that describes the type of coprocessor in general terms. Auditors should 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 has detected a possible tamper event and is in an unrecoverable, non-functional state. (Unintended "tamper" events are created if the coprocessor is not handled properly. Only remove the batteries when following the recommended battery changing procedure, maintain the coprocessor in the safe temperature range, and so forth. See the *IBM 4764 PCI-X Cryptographic Coprocessor Installation Manual*.)

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

- UNOWNED: currently not in use, no content
- RUNNABLE: contains code and is in a generally 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 proper 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-fresh coprocessor, the name will include **Factory**. This image and associated validation key will need to be changed.

For a previously loaded coprocessor, the Segment 1 name will probably include **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 should match the values shown in Figure 1 on page 9.

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 mix-and-match different release levels except with specific instructions from IBM.

Start the CLU utility and enter the parameters interactively (see "Changing the default directory and running CLU" on page 7).

- Enter the log file name (**########.LOG**, where ######## is the serial number of the coprocessor).
- Enter the command, **PL**.
- If there are multiple coprocessors, enter the coprocessor number.
- Enter the CLU file name as indicated in the readme.txt file.

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

Validating the coprocessor segment contents

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 validate the digital signature on the response created by the coprocessor. Depending on the IBM 4764 coprocessor (PartNum) in use,[1] issue this command and substitute the class-key certificate file name from Table 3 for the datafile file name. Note that the datafile name is "v.clu" appended to the coprocessor part number, all in lower case.

```
csuxclu ########.log VA [coprocessor_#] datafile
```

The part number can be obtained using the CLU utility ST command.

Table 3. Class-key file for use with the CLU VA command

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

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

Unloading coprocessor software and zeroize the CCA node

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

1. You can refer to the IBM product Web site (*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.

Refer to the readme.txt file that accompanies the software distribution you are using for the specific .clu file name used to surrender ownership of Segments 2 and 3. The readme.txt 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 logfile name
- Use the **PL** command to surrender Segment 2 as indicated in the readme.txt file for your platform

Note:

1. You can also zeroize CCA without removing the software by using the CCA reinitialize process. See "Initializing (zeroize) the node" on page 20.
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-fresh product. Segment 1 can only be changed a limited number of times before the available Device Key certificate space is exhausted and the coprocessor is potentially rendered unusable. If you require a capability to restore the Segment 1 factory validating key, and are willing to expose your coprocessor to a possible lock-up condition, you can obtain the required file from IBM by submitting a query using the Support Form on the product Web site, *http://www.ibm.com/security/cryptocards*. It is important to note that certificate space is a non-renewable resource. Once used, it cannot be recovered.

Coprocessor Load Utility reference

If you are interested in additional details related to the coprocessor code loading process, continue reading this section. Otherwise, continue reading at "Using the CNM and CNI utilities to manage the cryptographic node" on page 15.

This reference section describes:

- The coprocessor memory segments to which you load the software
- The approach the coprocessor uses to validate software loads
- The syntax used to invoke the CLU utility
- CLU utility return codes

Coprocessor memory segments

Coprocessor memory segments are organized as shown in the following table:

Table 4. 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.

Table 4. Organization of memory segments (continued)

Segment	Description
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.

Validation of coprocessor software loads

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

Loading code into coprocessor Segment 2 and Segment 3 is a two-step process for each segment.

1. Identify an owner for the segment 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 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 using “code load”. Two different commands are available.
   a. Initially use the **Load** command. Load-command data includes a public-key certificate that must be validated by the public key already residing with the next-lower segment. If the certificate is validated, and if the owner identifier in the Load-command data matches the current ownership held by the coprocessor for the segment, and if the complete Load-command data can be validated by the public key in the just-validated certificate, the coprocessor will accept the code and retain the validated public-key for the segment.
   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 a **Surrender Owner** of a segment. Note that the SRDIs for a segment are not zeroized when using the **Reload** command. The CCA application stores the master keys, the FCV, the access-control tables, and retained RSA private keys as SRDI information associated with Segment 3.

IBM signs its own software. Should another vendor intend to supply software for the coprocessor, that vendor's **Establish Owner** command and code-signing public-key certificate must have been signed by IBM under a suitable contract. These restrictions ensure that:

- 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

This section details the syntax used to invoke the Coprocessor Load Utility (CLU), and describes each function available in it. Use CLU to:

- Ensure that the coprocessor(s) is not “busy” by ending any application(s) that might have used a coprocessor. For example, end all applications that use the CCA API.
- Obtain the release level and the status of software currently installed in the coprocessor memory segments.

2. In this publication, the terms “load” and “reload” are employed. Other documentation might refer to these operations as “emergency burn” (EmBurn), and “regular burn” or “remote burn” (RemBurn), respectively.

- Confirm the validity of digitally signed messages returned by the coprocessor.
- Load and reload portions of the coprocessor software.
- Reset the coprocessor.

To invoke the utility:

1. Log on as required by your operating system.
2. Go to the command line.
3. Change directory to the directory containing the CLU utility files. The default directory is */usr/lpp/csufx.4765/clu*.
4. Enter the utility name followed by the parameters described below. The utility name is **csulclu**.

If you do not supply the necessary parameters, the utility will prompt you as information is required. Optional parameters are enclosed in brackets. The syntax for the parameters following the utility name is:

[*log_file cmd* [*coprocessor _#*] [*data_file*] [-*Q*]]

where:

log_file
: Identifies the logfile name. The utility appends entries to this ASCII text file as it performs the operations 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. For information about the contents of this log file, see "Machine-readable log contents" on page 41.

cmd
: Specifies a two-letter abbreviation representing the loader command to be run. See "Coprocessor Load Utility commands" on page 14.

coprocessor_#
: 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, 2. You can use the serial number information that you obtain with the **ST** or **VA** commands and the serial number printed on the end-bracket of the coprocessor to correlate a particular coprocessor to the coprocessor_#. The utility supports up to eight coprocessors per machine.

data_file
: Identifies the data file (drive, directory, and file name) used for the operation requested.
  - For software loads and reloads, it is the file name of the software image you are loading into the coprocessor. The CCA Support Program readme.txt file provides the data_file name,
  - When obtaining coprocessor status with the **VA** command, it is the class-key certificate file name used to validate the coprocessor response. The product Web site (*http://www.ibm.com/security/cryptocards*) Frequently Asked Questions (FAQ) area contains a description of the procedure for validating the coprocessor and its code. This description also contains a list of all the current class-key certificate file names. You can download any required certificate file from the Web site.

-*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:

csunclu ########.log va datafile_name**.clu**

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

Coprocessor Load Utility commands: The Coprocessor Load Utility (CLU) supports the following loader commands:

Table 5. 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/box 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.
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 *datafile_name*. 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 invoked by a script file or a command file. When creating a script file or a command file to invoke the utility on an unattended system, add "quiet" syntax **-q** (or **-Q, /q,** or **/Q**) to request that nothing be output to the display. By default, the utility returns prompts and messages to the display.

```
======================================================================
CSULCLU V3.10 ssall.log SS ALL    begun Wed Sep 17 15:31:11 2008
*********** Command SS started. ---- Wed Sep 17 15:31:11 2008

 Card #    P/N       S/N        Segment 3 Description
 ------    -------   --------   -----------------------------------
   0       41U0441   96004664   3.30.04 CCA
   1       12R8565   94000568   3.30.04 CCA
   2       12R8565   94000613   3.30.04 CCA
 *** Query System Status successful ***
System Status ended successfully!
*********** Command SS ended. ---- Wed Sep 17 15:32:43 2008
```

Figure 2. Typical CLU system status response

Coprocessor Load Utility return codes

When CLU finishes processing, it returns a value that can be tested in a script file or in a command file. The returned values and their meanings are:

0 OK.

1 Command line parameters not valid.

2 Cannot access the coprocessor. 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 installed. Ensure that the coprocessor and its driver have been properly installed.

5 Invalid coprocessor number specified.

6 A data file is required with this command.

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

Installing and Removing coprocessor host software

For each operating system, the following sections:

- Lists system requirements and procedures to install the Support Program.
- Describe how to install the Support Program.
- Describe how to remove the Support Program.

After you have installed the software as described in this chapter, you are ready to install the coprocessor software; see Chapter 4,"Loading software into the coprocessor" on page 7.

Note that all components of the installed CCA package might not indicate the same version number. The version number for a software component is updated only when the component is changed for a release. Unchanged components retain the version number from the last time that they were modified. For example, the Cryptographic Coprocessor Support Software might be at a different level from the coprocessor device driver.

Using the CNM and CNI utilities to manage the cryptographic node

This topic describes the CCA Node Management (CNM) utility and the CCA Node Initialization (CNI) utility provided with the Support Program to set up and manage the CCA cryptographic services provided by a node.

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 provided with the Support Program are tools to set up and manage the CCA cryptographic services provided by a node.

This chapter includes:

- What the utilities are and how to start them
- Three sample scenarios on how to use the utilities that you should consider
-
- How to use the CNM utility administrative functions: Things that you should be aware of in the CNM utility. You should review this material after working through the topic "Establishing a test node" on page 17.
- How to create and manage access-control data: Some details about the access-control portion of the CNM utility.
- How to manage cryptographic keys: Some of the key management things you can accomplish with the CNM utility.
- Using the CNI utility to establish other nodes: How you can automate use of the CNM utility using encapsulated procedures.

Note: This chapter describes the major functions of the CNM utility. For additional information about specific panels and fields, refer to the online help panels included with the utility.

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

Overview

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

Note:

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. If you change the release of the Java runtime environment 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 click instead of the Enter key for consistent results.
4. No help panels are provided for the Master-Key Cloning portion of the utility. See "Cloning a DES or PKA master key" on page 29.
5. These utilities use the IBM Common Cryptographic Architecture (CCA) API to request services from the coprocessor. The *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* 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 herein to understand which commands might require authorization in the various roles that you will define using the procedures described in this chapter.

CCA Node Management utility overview

The CCA Node Management utility is a Java application that provides a graphical user 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 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 Java.

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 panel are displayed.

CCA Node Initialization utility overview

The CCA Node Initialization utility runs scripts that you create using the *CNI Editor* within the CNM utility. These scripts are known as *CNI lists*. The CNI utility can run the CNM utility functions 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 forth) 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 Java.

The CNI utility is further explained in "Using the CNI utility to establish other nodes" on page 33.

Using the utilities, sample scenarios

The following scenarios illustrate how to use the utilities:

1. Establish 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. Establish nodes for a production environment using key parts. This scenario employs CNI lists to automate establishment of “target” production nodes.
3. Clone a master key from one coprocessor to another coprocessor. This is a procedure of interest to very high-security installations that employ multiple coprocessors.

The purpose of the scenarios is to illustrate how the procedures described in this chapter can be used. Where appropriate, a scenario cross-refers to sections with more detailed information.

If you are not familiar with the coprocessors's CCA access-control system\, see “Access-control overview” on page 22 and “Initial state of the access-control system” on page 22. Here you will find an explanation of terms like *role, initial-DEFAULT role,* and *user profile*. The scenarios assume that the access-control systemis in its initial state.

Note: These scenarios are instructional only. You are encouraged to determine the procedures best suited for your specific environment. Review the contents of Appendix H in the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*. This appendix describes observations on secure operations.

Establishing a test node

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

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

1. Install the coprocessor and the IBM CCA Cryptographic Coprocessor Support Program as described in the previous chapters. Start the CNM utility as described in “Starting the CCA Node Management Utility” on page 16.

   Remember that you must have installed an appropriate level of the Java Runtime Environment (JRE) or the Java Development Kit (JDK).
2. 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** drop down menu, select **Select Adapter**. You will see a drop-down list of available adapter numbers (ranging from one up to a maximum of eight). Choose an adapter (coprocessor) from the list. If you do not use the Select Adapter drop down to choose an adapter, the default adapter (coprocessor) is used.
3. Synchronize the clock within the coprocessor and host computer. From the **Crypto Node** drop down menu, select **Time**; a submenu is displayed. From the submenu, select **Set**; the clocks are synchronized.
4. Use the CNM utility to permit all commands in the DEFAULT role. From the **Access Control** drop down menu, select **Roles**. Highlight the **DEFAULT** entry and select **Edit**. You will see a screen that shows which commands are already enabled and which commands are not enabled by the DEFAULT role. Select **Permit All**. Then load the modified role back into the coprocessor by selecting **Load** and then **OK**.

   Before selecting **Cancel**, you could have saved a copy of this “all-commands-enabled” role to your file system using the **Save** button and assigning a file name. You must also select the folder (directory) where you will save the role.

   For more detail, see “Defining a role” on page 23.

   Finish this task by selecting **Cancel**.
5. Load the function-control vector (FCV) into the coprocessor. From the **Crypto Node** drop down menu, select **Authorization**; a submenu is displayed. From the submenu, select **Load** to specify and load the FCV.

The FCV file that you need to specify is the one that was placed on your server during the installation process. FCVs usually have file names such as "fcv_td4kaes256.crt" and can be found by using the file search utility available with your operating system.

6. Install a master key. From the **Master Key** pull-down menu, select either **DES/PKA Master Keys** or **AES Master Keys**, and then select **Yes**; the coprocessor generates and sets a random master key.

   The master key installed with Auto Set has actually passed through the main memory of your system processor as key parts. For production purposes, you should 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 pull-down menus mentioned above.

   For more detail, see "Loading the master key automatically" on page 28.

7. *Key storage* is a CCA term that describes a place where the Support Program can store DES, RSA, and 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. See "Creating or Initializing key storage" on page 32.

Establishing nodes in a production environment

In this scenario, the responsibility for establishing 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 key(s). The key-encrypting keys can be used as transport keys to convey other keys between nodes.

Note that this scenario is focused on installing master keys and high-level, inter-node DES key-encrypting keys from *key parts*. The CCA implementation supports alternatives to the key-part technique such as random master-key generation and distribution of DES keys using techniques based on 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 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 CNM utility to prepare CNI lists for the target node(s). The CNI lists automate the process of using the CNM utility at the target node. The administrator prepares a CNI list for the tasks 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, including:

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

The administrator authorizes commands in the various roles to ensure that only required command are enabled. Sensitive commands, such as loading a first key-part or loading subsequent key-part(s), are only enabled in roles for users with the responsibility and authority to utilize 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.

For more detail, see "Creating and Managing access-control data" on page 21.

Access-control administrator procedure: In this task, the access-control administrator uses the CNM utility to prepare CNI lists for the target node(s). 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, including:
   - Supervisory roles and user profiles for the access-control administrator and the key-management officers
   - A DEFAULT role to replace the initial-DEFAULT role

   For more detail, see "Creating and Managing access-control data" on page 21. For information about creating a CNI list, see "Using the CNI utility to establish other nodes" on page 33.
   a. Create a CNI list to:
      1) Synchronize the clock-calendar within the coprocessor and host computer
      2) Load the access-control data.
      3) Log on as an access-control administrator
      4) Load the replacement DEFAULT role
      5) Load the FCV
      6) Log off
   b. Create a CNI list for the first key-management officer:
      1) Log on as the first key-management officer
      2) Load a first master-key key-part
      3) As required, load first-part key-encrypting-key information
      4) Log off
   c. Create a CNI list for the second key-management officer:
      1) Log on as the second key-management officer
      2) Load a second master-key key-part
      3) As required, load second-part key-encrypting-key information
      4) Log off
3. Install the coprocessor and the Support Program onto the target node(s).
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 developed in steps 2a, 2b, and 2c. See "Using the CNI utility to establish other nodes" on page 33.

The target nodes are now ready to provide cryptographic service.

Key-management officer procedures: The key-management officers have two tasks:

- Prepare the key parts for use at the target node(s)
- Load the key parts at the target nodes

You have to decide how the key parts will be transported from the point of generation to the point of installation. The following are few scenarios:

1. Generate the key parts at a central place and transfer these on diskettes
2. Generate the key parts at a central place and transfer these on paper forms
3. Generate the key parts at the point and time of (first) installation. If the key parts are required at another time, either to reload or to share with another node, then how the key parts will be transported has to be decided.

You should 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 that has been prepared in conjunction with the access-control administrator.

Using CNM Utility Administrative Functions

This section describes how to use the CNM utility administrative functions to:

- Optionally choose among multiple coprocessors
- Initialize (or "zeroize") the coprocessor
- Log on to and off of the coprocessor node
- Load the coprocessor FCV
- Configure the CNM utility defaults
- Synchronize the clock-calendars within the coprocessor and the host computer
- Poll status information about the coprocessor in the CCA application

Choosing a specific coprocessor

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. Once 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.

Choose Select Adapter from the **Crypto Node** menu to select an adapter (coprocessor). If you do not choose an adapter, the default adapter is used.

Note:

1. When using the CLU utility, coprocessors are referenced as 0, 1, 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 1, 2, 3. These new identifiers are assigned by CCA as 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, 3 used in the CNM utility pull-down.

Initializing (zeroize) the node

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 this command causes clearing of all:

- Master-key registers
- Retained PKA and registered PKA public keys
- Roles and profiles and restoring the access control to its initial state (see "Initial state of the access-control system" on page 22).

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

Logging on and to Logging off the node

To log on, select **Passphrase Logon...** from the **File** pull-down menu. To log off, select **Logoff** from the **File** pull-down menu.

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

Loading the function-control vector

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 CCA Support Program.

You use the CNM utility to load the FCV into the coprocessor. The FCV file is named "fcv_td4kaes256.crt". You can locate this file using the filename search tool provided with your operating system.

To load the FCV:

1. From the **Crypto Node** pull-down menu, select **Authorization**; a submenu is displayed.
2. From the submenu, select **Load** to specify the FCV file on disk. Specify the file name and select **Update**. The utility loads the FCV.
3. Select **OK** to finish the task.

Configuring the CCA Node Management 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-calendars

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

To synchronize the clock-calendars:

1. From the **Crypto Node** pull-down menu, select **Time**; a submenu is displayed.
2. Select **Set** from the submenu.
3. Answer **Yes** to synchronize the clock-calendars with the host.
4. Select **OK** to finish this task.

Obtaining status information

You can use the CNM utility coprocessor to obtain the status of the CCA application. The following status panels are available:

- *CCA Application:* Displays the version and the build date of the application. Also displays the status of the master-key registers. For information about these registers, see "Managing the master keys" on page 27.
- *Adapter:* Displays the coprocessor serial number, ID, and hardware level.
- *Command History:* Displays the five most recent commands and subcommands 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 FCV resident within the coprocessor.

To view the status panels:

1. From the **Crypto Node** pull-down menu, select **Status**. The CCA application status is displayed.
2. To select other status information, use the buttons at the bottom. The new panel is displayed.
3. Select **Cancel** to finish this task.

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 commands, refer to Appendix G of the *IBM CCA Basic Services Reference and Guide frefbook 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 commands that are required and under what circumstances. You also need to consider that some commands should be authorized only for trusted individuals, or for certain programs that operate at specific times. Generally, you should 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(s). You will obtain the information about command use from the documentation for the applications that you intend to support. Refer to Appendix H in the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* for additional guidance on this topic.

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 (those that are not authorized in the default role), a user must log on to the coprocessor using a unique user profile. Each user profile is associated with a role. (Multiple profiles can use the same role.) The coprocessor authenticates logons using the passphrase 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 onto 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 “first” or “subsequent” key parts.)
- **General User**: Can use cryptographic services to protect his or her 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

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. For a full description of this role, see “Initial DEFAULT-role commands” on page 41.

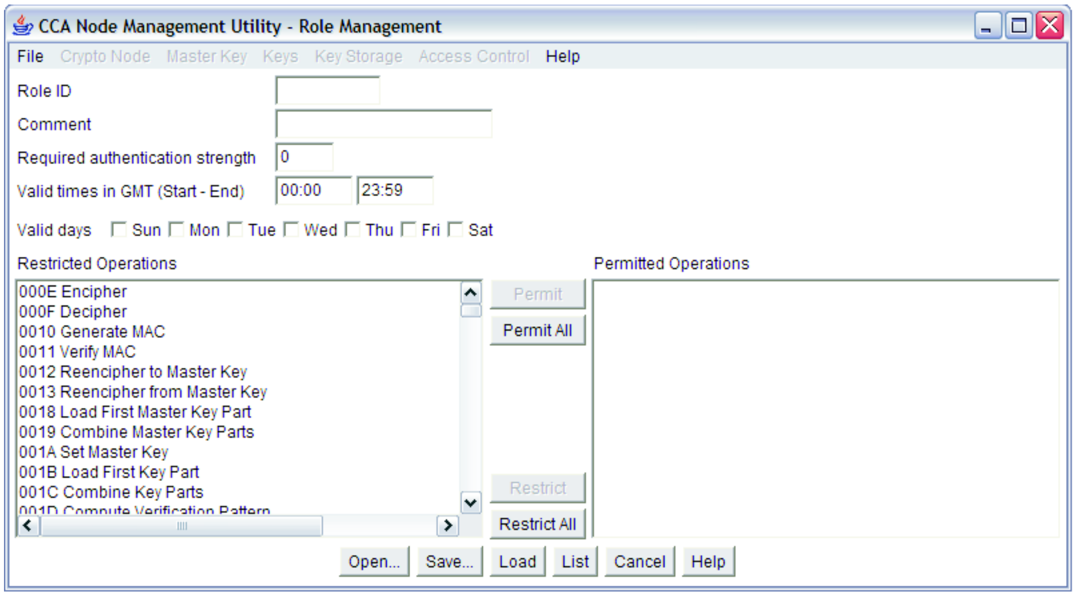
After creating the roles and profiles needed for your environment—including the supervisory roles necessary to load access-control data and to manage cryptographic keys—remove all permissions assigned to the DEFAULT role. Then, add only those permissions you want to grant to unauthenticated users.

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

Defining a role

A role defines permissions and other characteristics of the users assigned to that role. To define a role:

1. From the **Access Control** pull-down menu, select **Roles**; a list of currently defined roles is displayed.
2. Select **New** to display the Role Management panel. At any time in the process, select **List** to return to the list of currently defined roles.


CCA Node Management Utility - Role Management
File Crypto Node Master Key Keys Key Storage Access Control Help
Permit
Permit All
Restrict
Restrict All
Open... Save... Load List Cancel Help

Role ID

Comment

Required authentication strength 0

Valid times in GMT (Start - End) 00:00 23:59

Valid days ☐ Sun ☐ Mon ☐ Tue ☐ Wed ☐ Thu ☐ Fri ☐ Sat

Restricted Operations

000E Encipher
000F Decipher
0010 Generate MAC
0011 Verify MAC
0012 Reencipher to Master Key
0013 Reencipher from Master Key
0018 Load First Master Key Part
0019 Combine Master Key Parts
001A Set Master Key
001B Load First Key Part
001C Combine Key Parts
001D Compute Verification Pattern

Permitted Operations

Figure 3. Role Management panel

3. Define the role:

Role ID
A character string that defines the name of the role. This name is contained in each user profile 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
These values determine 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, refer to Chapter 2 of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*.

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 frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*.

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

Modifying Existing Roles

Use the CNM utility to:

- Edit a disk-stored role
- Edit a coprocessor-stored role
- Delete a coprocessor-stored role

Tip: 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 panel. You need only modify or enter information specific to the new role, then give it a new Role ID and load or save it.

Editing a disk-stored role: To edit a role stored on disk:

1. From the **Access Control** pull-down menu, select **Roles**; a list of currently defined roles is displayed.
2. Select **Open**; you are prompted to choose a file.
3. Open a file; data is displayed in the Role Definition panel.
4. Edit the role.
5. Select **Save** to save the role to disk; select **Load** to load the role into the coprocessor.

Editing a coprocessor-stored role: To edit a role stored in the coprocessor:

1. From the **Access Control** pull-down menu, select **Roles**; a list of currently defined roles is displayed.
2. Highlight the role you want to edit.
3. Select **Edit**; data is displayed in the Role Definition panel.
4. Edit the role.
5. Select **Save** to save the role to disk; select **Load** to load the role into the coprocessor.

Deleting a coprocessor-stored role: **Important**: When you delete a role, the CNM utility does not automatically delete or re-assign the user profiles associated with that role. You should delete or re-assign the user profiles associated with a role *before* you delete the role.

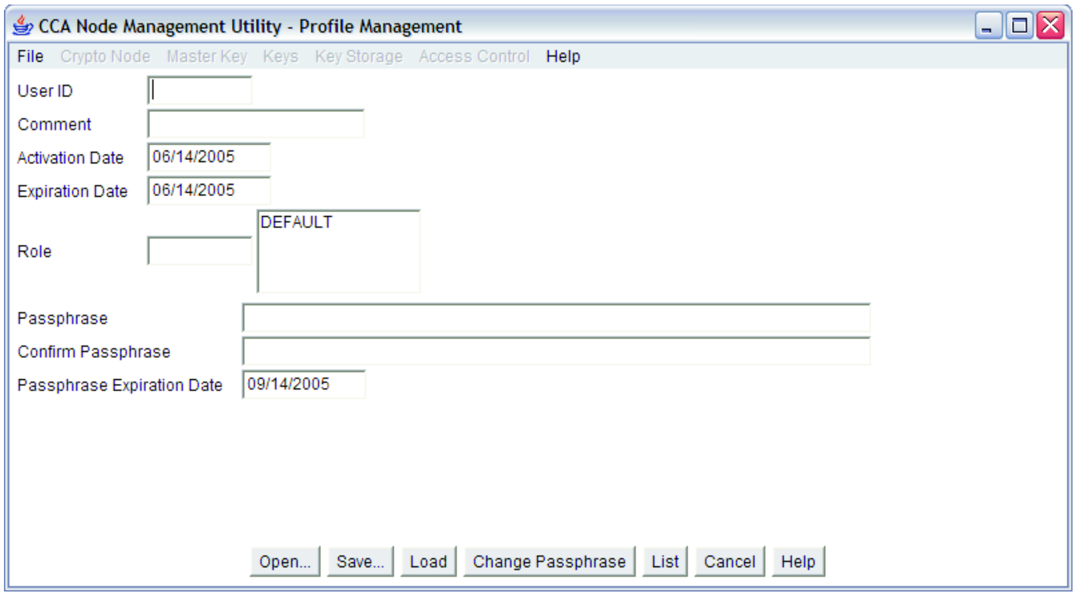
To delete a role stored in the coprocessor:

1. From the **Access Control** pull-down menu, select **Roles**; a list of currently defined roles is displayed.
2. Highlight the role you want to delete.
3. Select **Delete**; the role is deleted.

Defining a user profile

A user profile identifies a specific user to the coprocessor. To define a user profile:

1. From the **Access Control** pull-down menu, select **Profiles**; a list of currently defined profiles is displayed.
2. Select **New** to display the Profile Management panel; see Figure 4 on page 25.



User ID

Comment

Activation Date 06/14/2005

Expiration Date 06/14/2005

Role DEFAULT

Passphrase

Confirm Passphrase

Passphrase Expiration Date 09/14/2005

Figure 4. Profile Management panel

3. Define the user profile:

   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
   These values determine the first and last dates when 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. Select **Save** to save the profile to disk; select **Load** to load the profile into the coprocessor.
5. Select **List** to return to the list of currently defined profiles.

Modifying Existing Roles

Use the CNM utility to:

- Edit a disk-stored role
- Edit a coprocessor-stored role
- Delete a coprocessor-stored role

Tip: 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 panel. You need only modify or enter information specific to the new role, then give it a new Role ID and load or save it.

Editing a disk-stored user profile: To edit a profile stored on disk:

1. From the **Access Control** pull-down menu, select **Profiles**; a list of currently defined profiles is displayed.
2. Select **Open**; you are prompted to choose a file.
3. Open a file; data is displayed in the User Profile Definition panel.
4. Edit the profile.
5. Select **Save** to save the profile to disk; select **Load** to load the profile into the coprocessor.

Editing a coprocessor-stored role: To edit a role stored in the coprocessor:
1. From the **Access Control** pull-down menu, select **Roles**; a list of currently defined roles is displayed.
2. Highlight the role you want to edit.
3. Select **Edit**; data is displayed in the Role Definition panel.
4. Edit the role.
5. Select **Save** to save the role to disk; select **Load** to load the role into the coprocessor.

Deleting a coprocessor-stored user profile: To delete a profile stored in the coprocessor:
1. From the **Access Control** pull-down menu, select **Profiles**; a list of currently defined profiles is displayed.
2. Highlight the profile you want to delete.
3. Select **Delete**; the 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 profile exceeds the limit defined in the profile, the offending profile is disabled. To reset the failure count:
1. From the **Access Control** pull-down menu, select **Profiles**; a list of currently defined profiles is displayed.
2. Highlight the profile.
3. Select **Reset FC**; a confirmation dialog box is displayed.
4. Select **Yes** to confirm; the logon-attempt failure count is set to zero.

Initializing the access-control system

When you initialize the access-control system, the CNM utility:
- Clears the access-control data in the coprocessor.
- 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 installation-installed access controls and keys and is therefore a very sensitive operation that could render your node inoperable for production. Some installations will choose to remove authorization for this function from their coprocessor's roles. In this event, if you wish to initialize the CCA cryptographic node you must remove the CCA software from the coprocessor and re-install the CCA software.

To initialize the access-control system:
1. From the **Access Control** pull-down menu, select **Initialize**; a confirmation dialog box is displayed.
2. Select **Yes** to confirm; the utility initializes the access-control system.

Managing Cryptographic Keys

This section describes how to use the CNM utility to:
- Manage the master keys
- Manage primary key-encrypting keys (KEKs)

- Reset and manage DES, PKA, and AES key-storage

Key types are defined as follows:

A **master key** is a special KEK stored in the clear (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 used to provide cryptographic services, such as 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, since 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 used to store master-key information as accumulated to form a new master-key.

The 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.

For information about checking the contents of these registers, see “Obtaining status information” on page 21.

Note:

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 **BOTH** 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 will 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 will need to be installed with identical master keys.

This section describes how to:

- Verify an existing master key
- Load a master key automatically
- Load a new master key from parts

- Clone a master key

Verifying an existing master key: The utility generates a verification number for each master key 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:

1. From the **Master Key** pull-down menu, select either **DES/PKA Master Keys** or **AES Master Key**, and then select **Verify**; a submenu is displayed.
2. From the 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 auto-set a master key into 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:

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

Note:

1. Use of Random is preferred since the Auto-Set option passes clear key-parts through host-system memory.
2. When you set or auto-set a master key, you must re-encipher all keys enciphered under the former key. See "Reencipher the stored keys" on page 32.

Loading a new master key from key parts: To set a new master-key into the coprocessor, enter any part of the key into the new-master-key register , and set the new master-key. To achieve this:

1. From the **Master Key** drop down menu, select either **DES/PKA Master Keys** or **AES Master Key**, and then select **Parts**; the Load Master Key panel is displayed; see Figure 5.

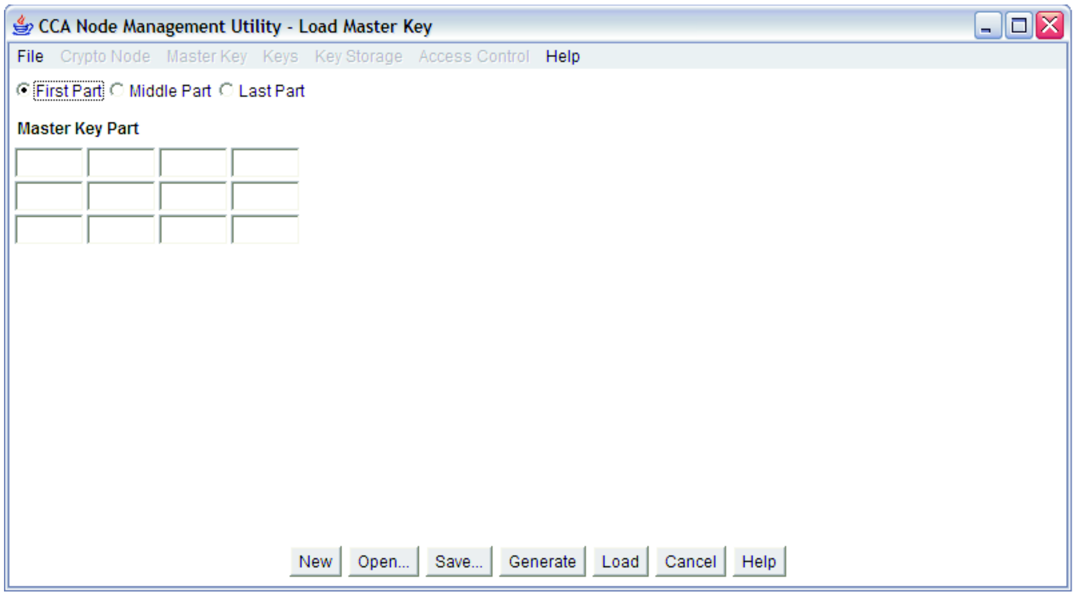


Figure 5. Load Master Key panel

2. Select the radio button for the key part you are editing (First Part, Middle Part, or Last Part).
3. Enter data by one of the following:
   - Select **New** to clear data entered in error.
   - Select **Open** to retrieve preexisting data.
   - Select **Generate** to fill the fields with coprocessor-generated random numbers.
   - Manually enter data into the "Master Key Part" fields; each field accepts four hexadecimal digits.
4. Select **Load** to load the key part into the new-master-key register; select **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 first and last part. The middle part is optional.
5. Repeating the preceding steps, load the remaining key parts to the new-master-key register.

   Note: For 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 first key to one role and the right to enter subsequent key part(s) to another role. Then, authorized users can log on enter their respective key part.
6. From the **Master Key** drop down menu, select either **DES/PKA Master Keys** or **AES Master Key**. Select **Set...** for the utility to transfer the data:
   a. From the current-master-key register to the old-master-key register, and deletes 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. See "Reencipher the stored keys" on page 32.

Cloning a DES or PKA master key: This scenario explains the steps involved in *cloning* a DES or PKA master key from one coprocessor to another. (Cloning of an AES master key is not supported.) The term cloning is used rather than copying since the master key will be split into *shares* for transport between the coprocessors. The technique is explained at some length in "Understanding and managing master keys" in Chapter 2 of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*. "Master-key cloning" on page 43 provides a step-by-step procedure that you can follow. The material in this chapter provides background information that permits you to vary the procedure.

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 also be either the source or the target node.)

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

You can accomplish these tasks using the CNM utility:
1. Set up the nodes in a secure manner with access-control roles and profiles and master keys.

   You will need a role and profile(s) at the source and target nodes for each user who will obtain or store share $_i$, 1≥i≥n. Processing of share $_i$ is a separate command so that, if you wish, your roles can insure that independent individuals are involved with obtaining and installing the different shares.

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

See the guidance portion of Chapter 2 in the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* for the description of the Master_Key_Process and the Master_Key_Distribute verbs.

2. Install a unique 1- to 16-byte Environment ID (EID) of your choice into each node.

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

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

3. 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. "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** drop down menu, select **Share Administration**, and then select **Set number of shares**, enter the values, and select **Load**.

4. At the different nodes, generate these keys and have each public key certified by the share-administration (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.

   When the SA key is created, the utility will supply an 8-byte/16-hex-character value that is a portion of the hash of the SA key. *Be sure to retain a copy of this value.* You will need this value to confirm the hash value 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 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** drop down menu, select **Share Administration**, select **Create Keys**, and one of **Share Administration Key**, **CSS key**, or **CSR key**, then select **Create**.

   You also will need to supply key labels for the CSS and CSR keys that are retained in the source and target nodes. For example, 'IBM4764.CLONING.CSS.KEY' and 'IBM4764.CLONING.CSR.KEY'; the labels that you use must not conflict with other key labels used in your applications.

   When generating the CSR key at the share receiving node, also obtain the serial number of the coprocessor. From the **Crypto Node** drop down menu, select **Status**. You must enter the serial number value when certifying the CSR key.

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

One individual should install the SA public-key hash. From the **Crypto Node** drop down menu, select **Share Administration**, select **Register Share Administration**, and select **SA key hash**. You will enter the hash value obtained during SA key creation.

The other individual should install the actual SA public-key. From the **Crypto Node** drop down menu, select **Share Administration**, select **Register Share Administration**, and select **SA Key**. By default, the public-key information is in the sa.db file.

6. Take the CSS key and the CSR key to the SA node and have the keys certified.

   From the **Crypto Node** drop down menu, select **Share Administration**, select **Certify Keys**, and one of **CSS key** or **CSR key**.

   For the CSR key, you will need to supply the serial number of the target coprocessor as a procedural check that an appropriate key is being certified. Your procedures should include communicating this information in a reliable manner.
7. At the source node, have authorized individuals sign on to the role that permits them to obtain their shares. At least “m” shares must be obtained. These will be shares of the current master-key.

   From the **Crypto Node** drop down menu, select **Share Administration**, select **Get Share**, and select the share number to be obtained. Observe the serial numbers and database identifiers. When these are agreed to be correct, select **Get Share**. The share information will be placed by default into the csr.db file and will obtain 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** drop down menu, select **DES/PKA Master Keys**, then select **Verify**, then select **Current**.
8. At the target node, have authorized individuals sign on to the role that permits each of them to install his or her share. At least “m” shares must be installed to reconstitute the master key into the new-master-key register.

   From the **Crypto Node** drop down menu, select **Share Administration**, select **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 are agreed to be correct, select Get Share. At the target node, have authorized individuals sign on to the role that permits each of them the individuals to install his or her 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 installed for the proper functioning of key storage to work properly.

   When “m” shares have been 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** pull-down menu, select **DES/PKA Master Keys**, then select **New**.
9. When it is confirmed through master-key verification that the master key has been 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. From the **Master Key** pull-down menu, select **DES/PKA Master Keys**, then select **Set...**.

Managing key storage

The CNM utility allows basic key-storage management 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. DES keys, PKA (RSA) keys, and AES keys are held in separate storage systems. Also, the has a very limited internal storage for PKA keys. The coprocessor-stored keys are not considered part of key storage in this discussion.

Note: 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.

This section describes how to:
- Create or initialize key storage
- Reencipher stored keys
- Delete a stored key
- Create a key label

Note: The 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 DES, PKA, or AES keys:
1. From the **Key Storage** pull-down menu, select **DES Key Storage**, **PKA Key Storage**, or **AES Key Storage**; a submenu is displayed.
2. From the submenu, select **Initialize**; the Initialize DES Key Storage, Initialize PKA Key Storage, or Initialize AES Key Storage panel is displayed.
3. Enter a description for the key-storage file, if desired.
4. Select **Initialize**; you are prompted to enter a name for the key-storage dataset.
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, and if it had any keys, these would be erased.

Reencipher the stored keys: To reencipher the keys in storage under a new master-key:
1. From the **Key Storage** pull-down menu, select **DES Key Storage**, **PKA Key Storage**, or **AES Key Storage**; a submenu is displayed.
2. From the submenu, select **Manage**; the DES Key Storage Management, PKA Key Storage Management, or AES Key Storage Management panel is displayed. The panel lists the labels of the keys in storage.
3. Select **Reencipher...**; the keys are reenciphered under the key in the current master-key register.

Deleting a stored key: To delete a stored key:
1. From the **Key Storage** pull-down menu, select **DES Key Storage**, **PKA Key Storage**, or **AES Key Storage**; a submenu is displayed.
2. From the submenu, select **Manage**; the DES Key Storage Management, PKA Key Storage Management, or AES Key Storage Management panel is displayed. The panel lists the labels of the keys in storage.

   You can set the filter criteria to list a subset of keys within storage. For example, entering "*.mac" as the filter criterion and refreshing the list limits it 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. Select **Delete...**; a confirmation dialog box is displayed.
5. Select **Yes** to confirm; the stored key is deleted.

Creating a key label: To create a key label:
1. From the **Key Storage** pull-down menu, select **DES Key Storage**, **PKA Key Storage**, or **AES Key Storage**; a submenu is displayed.
2. From the submenu, select **Manage**; the DES Key Storage Management, PKA Key Storage Management, or AES Key Storage Management panel is displayed. The panel lists the labels of the keys in storage.

You can set the filter criteria to list a subset of keys within storage. For example, entering "*.mac" as the filter criterion and refreshing the list limits it to keys with labels that end in ".mac". (The asterisk is a wildcard character.)

3. Select **New**; you are prompted to enter a key label.
4. Select **Load**; the key label is loaded into storage.

Creating and Storing primary DES KEKs

Key-encrypting keys (KEKs) are encrypted under the 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 information. The parts can also be saved to disk or diskette in the clear for transport to other nodes or for re-creating the local KEK.

Note: The 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 AES-based key distribution.

To work with a KEK (or other double-length operational key):

1. From the **Keys** pull-down menu, select **Primary DES Key-encrypting keys**; the Primary DES Key-encrypting keys panel is displayed.
2. At any time you can select **New** to clear all data fields and reset all the radio buttons to their default settings.
3. Select the radio button for the desired key-part to be entered: **First Part**, **Middle Part**, or **Last Part**.
4. Enter data in the **Key Part** fields by using one of the following processes:
   - Select **Open** to retrieve pre-existing Key Part, Control Vector, and Key Label data previously stored on disk using the **Save** command.
   - Select **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 four hexadecimal digits.
5. 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, refer to Appendix C of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*.
6. Enter a key label to identify the key token in key storage.
7. Select **Load** to load the key part into the coprocessor and store the resulting key token into key storage; select **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 Step 3 to Step 7. Be sure to use the same key label for each part of a single key.

Using the CNI utility to establish other nodes

By creating a CNI list for the CCA Node Initialization (CNI) utility, you can 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:

1. Start the CNM utility on an established node.

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 will search 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** pull-down menu, select **CNI Editor**; the CCA Node Initialization Editor panel is displayed. See Figure 6.

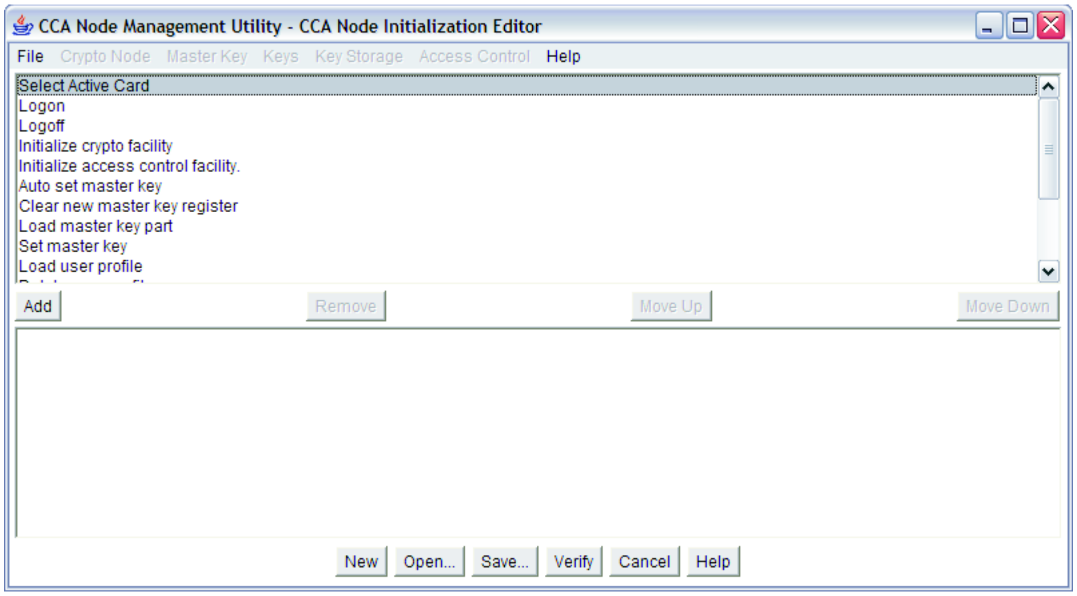


Figure 6. CCA Node Initialization Editor panel

The list in the top portion of the panel displays the functions can be added to the CNI list. The bottom portion lists the functions included in the current CNI list. References to 'master keys' in the list refer to the DES/PKA master keys. The AES master key references are explicitly indicated. The CNI list can perform the following functions:
- Select the active card
- Logon to and logoff of the cryptographic node
- Initialize the cryptographic facility (coprocessor)
- Initialize access control facility
- Auto set the master key
- Clear the new master key register
- Load master key parts
- Set the master key
- Auto set AES master key
- Clear the new AES master key register
- Load AES master key parts
- Set AES master key
- Load or delete user roles and user profiles
- Synchronize the clock calendars
- Initialize storage for DES keys, PKA keys, and AES keys
4. Add the functions you want. To add a function to the CNI list:

   a. Highlight it.
   b. Select **Add**; the function is added to the CNI list.

      Note: If the function you choose loads a data object—like 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 ccess-control data.
6. Select **Verify** to confirm that objects have been created correctly.
7. Select **Save**; you are prompted to choose 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 onto target host directory locations that mirror their location 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 using the **csufcni** command.

    If the CNI list includes a logon, enter **csulcni** or **csuncni** on the command line (without specifying a filename). The utility Help text 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

This topic describes building applications to use with CCA API.

This topic includes the following:
- An overview of the way in which applications obtain service from the Common Cryptographic Architecture (CCA) application program interface (API)
- The procedure for calling a CCA verb in the C programming language
- The procedure for compiling applications and linking them to the CCA API security application programming interface
- A sample routine written in the C programming language
- Enhancing throughput with CCA and the 4765

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

Note: The file locations referenced in this chapter are the default directory paths.

Overview

Application and utility programs issue service requests to the cryptographic coprocessor by calling the CCA verbs.

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 API security application programming interface, refer to the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*.

3. 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.

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 API security application programming interface verbs are contained in a header file. The files and their default distribution locations are:

AIX AIX/usr/include/

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

AIX `#include "csulincl.h"`

When you issue a call to a CCA API security application programming interface 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 Support Program. Although it is permissible to code null address pointers for these parameters, it is recommended that you specify a long integer valued to 0 with the *exit_data_length* parameter.

Compiling and Linking CCA application programs

The Support Program includes the C Language source code and the make file for a sample program. The file and its default distribution location is:

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 is:

AIX */usr/lib/libcsufcca.a.*

Sample C routine

To illustrate the practical application of CCA verb calls, this section describes the sample C programming language routine included with the Support Program. For reference, a hard copy of the sample routine is shown in Figure 7 on page 37. (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 effect this, the routine:

1. Calls the **Key_Generate**(CSNBKGN) verb to create a MAC/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, refer to the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* for the descriptions of the verbs and their parameters. These verbs are listed in Table 6 on page 37.

Table 6. Verbs called by the sample routine

Verb	Entry-point name
Key_Generate	CSNBKGN
MAC_Generate	CSNBMGN
MAC_Verify	CSNBMVR

Figure 7. Syntax, sample routine

```
/*********************************************************************/
/*                                                                   */
/* Module Name: mac.c                                                */
/*                                                                   */
/* DESCRIPTIVE NAME: Cryptographic Coprocessor Support Program       */
/*                   C language source code example                  */
/*                                                                   */
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Licensed Materials - Property of IBM                              */
/*                                                                   */
/* (C) Copyright IBM Corp. 1997-2001 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          "abcdefhgijklmn0987654321"
#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 multi-threading and multiprocessing, caching DES, PKA, and AES keys.

Multi-threading and multiprocessing

The CCA application running inside the 4765 can process several CCA requests simultaneously. The coprocessor contains several independent hardware elements, including the RSA engine, DES engine, CPU, random-number generator, and 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.

In order 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 accomplish this is to design a multi-threaded 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. By doing this, you guarantee that the coprocessor is not under-utilized. 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) AES keys in caches inside the secure module. The keys are stored in a form that has been decrypted, 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 EPROM memory.

As a result, applications that reuse a common set of keys can run much faster than those which 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 very effective in improving throughput. PKA public-keys and AES clear keys, which have very little processing overhead, are not cached.

Initial DEFAULT-role commands

This appendix describes the characteristics of the DEFAULT role after the coprocessor is initialized and when no other access-control data exists.

This appendix describes the characteristics of the DEFAULT role after the coprocessor is initialized and when no other access-control data exists: For the initials DEFAULT role commands the role ID is 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 using the DEFAULT role. Restrict these commands to selected supervisory roles.

Table 7 lists the access-control commands enabled in the DEFAULT role when the CCA software is initially loaded and when the CCA node is initialized.

Table 7. 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

This topic describes about the 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 input 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 link on the IBM product Web site at *http://www.ibm.com/security/cryptocards*.

Device driver error codes

This topic describes about various device driver error codes.

Each time that the coprocessor is reset, and the reset is not caused by a fault or tamper event, the coprocessor runs through 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 resets 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. [coprocessor is an inanimate object. Do not personify inanimate object.

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

There are a very large number of possible 4-byte codes, all of which are of the form X'8xxxxxxx'. The most likely codes that might be encountered are described in Table 8. If you encounter codes of the form XX'8340xxxx' or X'8440xxxx', and the code is not in the Table,8, contact the IBM Crypto Team through email from the Support page on the IBM product Web site (http://www.ibm.com/security/cryptocards).*http://www.ibm.com/security/cryptocards*).

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

4-byte return code (hex)	Reason	Descriptions
8040FFBF	External 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	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	Either 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.
8040FFFE	Battery warning	Battery power is marginal. For the procedure to be followed to replace the batteries, refer to the *IBM 4764 PCI-X Cryptographic Coprocessor Installation Manual*.
804xxxxx (for example, **80400005)**	General communication problem	Except for the prior X'8040xxxx' codes, there are additional conditions that arise 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 IBM Crypto Support via the Web site.
8340xxxx	Miniboot-0 codes	This class of return code arises from the lowest-level of reset testing. If codes in this class occur, contact IBM Crypto Support via the Web site.

Table 8. Device driver error codes in the class X'8xxxxxxx' (continued)

4-byte return code (hex)	Reason	Descriptions
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. The CLU status (ST) command should be run at least twice to determine if 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 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 this and a description of what you are trying to accomplish to IBM Crypto Support via the Web site.

Master-key cloning

This topic describes about Master-key cloning procedure and access-control considerations while cloning.

This appendix describes the following:
- Master-key cloning
- Access-control considerations when cloning

Master-key cloning procedure

The following procedure outlines how to clone a master key from one coprocessor to another coprocessor using the CNM utility. Before running this procedure, familiarize yourself with steps described in the section "Cloning a DES or PKA master key" on page 29 and "Understanding and managing master keys" in Chapter 2 of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*.

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

The master-key cloning procedure that follows makes no assumption about which computer contains the coprocessors used for:
- Share administration (**A 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 computer if multiple coprocessors with CCA are available.

The procedure ignores operator actions to log on and log off, as 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 computer.

The procedure is broken down into several phases as outlined in Table 9 on page 44.

Table 9. 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 current master-key-verification information.
3d	Target	Install shares and confirm new master-key; set the master key.

Before undertaking the master-key cloning procedure, it is recommended that you complete the forms found on the following pages

Table 10. 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			
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			

Table 10. Cloning responsibilities, profiles and roles (continued)

Task	Node	Profile	Role	Responsible individual
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: Establish the share administration node

Using the coprocessor designated as the Share Administration (SA) *node*, follow the steps in Table 11. Note that this coprocessor can also serve as the master-key source or a master-key target node.

Table 11. Master-key cloning procedure: establish SA node

Phase	Task	✓
1.1	Audit the appropriateness of the access controls.	
1.2	Perform time synchronization and ensure that the authorization (fcv_td4kaes256.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.	
1.5	If not already established, enter the Environment ID (EID): • **Crypto Node**, **Set Environment ID**. • Enter the EID, *Load*.	

Table 11. Master-key cloning procedure: establish SA node (continued)

Phase	Task	✓
1.6	Generate the SA key: • **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"). • *Create*. • Record the SA-key hash value for use later in the procedure.	
1.7	Register the SA public-key hash: • **Crypto Node**, **Share Administration**, **Register Share Administration Key**, **SA-Key Hash**. • Enter the SA database file name and location, *Next*. • Enter the SA public-key label (or accept the default). • Enter the SA-key hash, *Register*.	
1.8	Register the SA public-key: • **Crypto Node**, **Share Administration**, **Register Share Administration**, **SA Key**. • Enter the SA database file name and location, *Next*. • Enter the SA public-key label (or accept the default), *Register*.	

Phase 2: Establish the source node

Using the coprocessor designated as the master-key source node, follow the steps in Table 12. Note that this coprocessor can also serve as the SA node.

Table 12. Master-key cloning procedure: establish source (CSS) node

Phase	Task	✓
2a.1	Audit the appropriateness of the access controls.	
2a.2	Perform time synchronization and insure that the authorization (fcv_td4kaes256.crt) is installed.	
2a.3	Confirm the coprocessor serial number: • **Crypto Node**, **Status**. • *Adapter*. • Note the coprocessor serial number, *Cancel*.	
2a.4	Confirm (or install) the master key.	
2a.5	Obtain the current master-key-verification information: • **Master Key**, **Verify**, **Current**. • *Save* to transport media, *Cancel*.	
2a.6	If not already established, enter the Environment ID (EID): • **Crypto Node**, **Set Environment ID**. • Enter the EID, *Load*.	
2a.7	If not already established, set the number of shares values, "m" and "n": • **Crypto Node**, **Share Administration**, **Set Number of Shares**. • Set the maximum and minimum number of required shares, *Load*.	
2a.8	Generate the CSS key: • **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. • *Create*.	

Table 12. Master-key cloning procedure: establish source (CSS) node (continued)

Phase	Task	✓
2a.9	Register the SA public-key hash: • **Crypto Node**, **Share Administration**, **Register Share Administration Key**, **SA-Key Hash**. • Enter the SA database file name and location, *Next*. • Enter the SA public-key label (or accept the default). • Enter the SA-key hash, *Register*.	
2a.10	Register the SA public-key: • **Crypto Node**, **Share Administration**, **Register Share Administration**, **SA Key**. • Enter the SA database file name and location, *Next*. • Enter the SA public-key label (or accept the default), *Register*.	

Phase 3: Establish target node and clone master key

Using the designated nodes, establish the target node and clone the master key following the steps in Table 13. Note that this coprocessor can also serve as the SA node.

Table 13. Master-key cloning procedure: establish CSR node, clone 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 insure that the authorization (fcv_td2k.crt) is installed.	
3a.3	Target	Confirm the coprocessor serial number: • **Crypto Node**, **Status**. • *Adapter*. • Note the coprocessor serial number, *Cancel*.	
3a.4	Target	Ensure the existence of a (temporary) master key.	
3a.5	Target	If not already established, enter the Environment ID (EID): • **Crypto Node**, **Set Environment ID**. • Enter the EID (for example, “CSR1 NODE” and extend with spaces to 16 entered characters). • *Load*.	
3a.6	Target	If not already established, set the number of shares values, “m” and “n”: • **Crypto Node**, **Share Administration**, **Set Number of Shares**. • Set the maximum and minimum number of required shares. • *Load*.	
3a.7	Target	Using the facilities of your operating system, erase the csr.db data file.	
3a.8	Target	Generate the CSR key: • **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”). • *Create*.	
3a.9	Target	Register the SA public-key hash: • **Crypto Node**, **Share Administration**, **Register Share Administration**, **SA-Key Hash**. • Enter the SA database file name and location, *Next*. • Enter the SA public-key label (or accept the default). • Enter the SA-key hash, *Register*.	

Table 13. Master-key cloning procedure: establish CSR node, clone master key (continued)

Phase	Node	Task	✓
3a.10	Target	Register the SA public-key: • **Crypto Node**, **Share Administration**, **Register Share Administration Key**, **SA Key** • Enter the SA database file name and location, *Next*. • Enter the SA public-key label (or accept the default), *Register*.	
At the SA node...			
3b.1	SA	Certify the CSS key (as required): • **Crypto Node**, **Share Administration**, **Certify Keys**, **CSS Key**. • Enter the name and path for the SA database, *Next*. • Confirm the CSS key label, the coprocessor serial number, and the SA Environment ID. • *Certify*.	
3b.2	SA	Certify the CSR key: • **Crypto Node**, **Share Administration**, **Certify Keys**, **CSR Key**. • Enter the name and path for the SA and CSR databases, *Next*. • Confirm the SA key label, CSR key label, and the SA Environment ID. • Enter the CSR serial number. • *Certify*.	
At the source node...			
3c.1	Source	Obtain at least “m” of “n” shares. Perform the following for each share. Note that logon and logoff might be required to obtain each share. • **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, *Next*. • Confirm the CSS key label, CSS coprocessor serial number, and the CSR coprocessor serial number. • *Get Share*. Repeat as required.	
At the target node...			
3d.1	Target	Install “m” (of “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. • **Crypto Node**, **Share Administration**, **Load Share**. • Select the share. • Enter the name and path for the CSR and SA databases, *Next*. • Confirm the CSS key label, the CSS coprocessor serial number, and the CSR coprocessor serial number. • *Load Share*. Observe the response. Loading sufficient shares completes the new master-key. Repeat as required.	
3d.2	Target	Confirm the new master-key: • **Master Key**, **Verify**, **New**. • *Compare*, select the file, *OK*, *Cancel*	
3d.3	Target	Erase the **csr.db** data file. This is not a security issue but rather to avoid complications while another master-key cloning operation.	
3d.4	Target	As appropriate, set the master key: • **Master Key**, **Set** • *OK*	

Access-control considerations when cloning

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

- Roles at the share-administration node
- Roles at the source node, CSS
- Roles at the target node, CSR

You must also consider your security policy and 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 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 if this will be a split responsibility.

In addition, it must be decided 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, you must consider when the cloning will take place and if there is the 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 CNM utility places all of the shares from a node in the **CSR.DB** file. Each share is encrypted under a unique, triple-length DES key which itself is encrypted by the CSR public key of the target node.

Table 14 provides guidance for selecting the permissions applicable to the roles related to cloning.

Table 14. CCA commands related to master-key cloning

Code	Command name	Verb name	Consideration
X'001A'	Set Master Key	**Master_Key_Process**	Critical, must be knowledgeable 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**	Probably assigned to the role that can set the master key. This can override the collected shares. Probably should 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**	Consider who will update working keys encrypted by the master key.
X'0100'	PKA96 Digital Signature Generate	**Digital_Signature_Generate**	Certifier of the SA, CSS, and CSR keys.
X'0101'	PKA96 Digital Signature Verify	**Digital_Signature_Verify**	All

Table 14. CCA commands related to master-key cloning (continued)

Code	Command name	Verb name	Consideration
X'0102'	PKA96 Key Token Change	**PKA_Key_Token_Change**	Consider who will update working keys encrypted by the master key.
X'0103'	PKA96 PKA Key Generate	**PKA_Key_Generate**	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**	Allows changing the passphrase in ANY profile, use with discretion.
X'0116'	Read Public Access-Control Information	**Access_Control_Maintenance**	All
X'011C'	Set EID	**Cryptographic_Facility_Control**	Required to set up the CSS and CSR nodes.
X'011D'	Initialize Master Key Cloning	**Cryptographic_Facility_Control**	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**	Use 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**	Use 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**	Use to remove obsolete SA, CSS, and CSR keys, be careful about denial of service.
X'0204'	PKA Clone Key Generate	**PKA_Key_Generate**	Required to generate the CSS and CSR keys.
X'0211' - X'021F'	Clone-info (Share) Obtain	**Master_Key_Distribution**	Consider a profile and role for each share to enforce split responsibility.
X'0221' - X'022F'	Clone-info (Share) Install	**Master_Key_Distribution**	Consider 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

This topic describes various the threats that should be considered when deploying the IBM 4765 with the Support Program in a digital-signing application.

This appendix addresses threats which should be considered when employing the IBM 4765 with the 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 should consider how their installation will address various threats. Table 15 on page 51 lists potential threats and presents product design and implementation solutions to many of these threats. Notes® are included describing steps that you should consider to further mitigate your exposure to problems.

Appendix H of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* describes actions you should use in deploying the coprocessor, policies you should consider, application functionality to be included, and so forth.

Read the contents of Table 15 on page 51 after you have made first-pass decisions about your installation.

Table 15. 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 since it requires direct contact with the coprocessor internals. 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 mechanism. 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 which you can verify to confirm that the coprocessor is genuine and is untampered with. Note that 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 in order 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.
Environmental manipulation of the coprocessor An adversary might utilize 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 an effect, 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 which might induce erroneous operation. Abnormal conditions can cause the unit to zeroize.

Table 15. Threat considerations for a digital-signing server (continued)

Threat discussion	**Threat mitigation**
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.	**Note:** 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 proper operation can be relied upon.
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 inputs 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 inputs. 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/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 non-secure state through inappropriate termination of selected operations. Attempts to generate a non-secure state in the coprocessor might be made through premature termination of transactions or communications between the coprocessor and the host, by insertion of interrupts, 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.
Invalid input An adversary or authorized user of the coprocessor might compromise the security features of the coprocessor through introduction of invalid inputs. Invalid input might take the form of operations which 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, generation of exploitable errors in operation, or 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 utilize 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 will only accept 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 which 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	

Table 15. Threat considerations for a digital-signing server (continued)

Threat discussion	**Threat mitigation**
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 which 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 profiles (users) associated with each role. An independent evaluation of the coprocessor software implementation and testing has reviewed the integrity of the access-control implementation.
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 end 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. **Note:** 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 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 role to get access to information or perform services allowed for the authorized user.	There are two "user" classes: 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 an end-user auditor. 2. The CCA access-control design protects the integrity and confidentiality of an end-user access-control passphrase from the domain of the end-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	
Use of disallowed application functions An adversary might exploit interactions between applications to expose sensitive coprocessor or user data. Interactions might include execution of commands that are not required or allowed in the specific application being performed. Examples include 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. **Note:** 1. Your access-control configuration should follow the principles discussed in Appendix H of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* 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 to digital signing. In some installations it might be desirable to accommodate a different approach to roles and/or to address additional application(s) functionality. In these cases, ensure that you review the guidelines and observations in Appendix H of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors* for applicability to your circumstances.
Threats regarding cryptographic functions	

Table 15. Threat considerations for a digital-signing server (continued)

Threat discussion	**Threat mitigation**
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 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, observe the guidelines in Appendix H of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*.
Threats regarding digital signatures	
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. **Note:** 1. There are precautions in the use of CCA which should be observed as documented in Appendix H of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*. 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 which 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 the coprocessor is not supposed to sign. The adversary might try to submit data to the and get it signed without passing the authorization checks of the coprocessor it has to 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 **Note:** 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 which exclude unauthorized users from use of the digital-signing functionality.

Table 15. 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 a function(s) that discloses 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: 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 **Note:** To address these exposures: 1. Selectively enable the Delete Retained Key command, X'0203' 2. Use host-system access controls to manage usage of the CLU Utility 3. Manage physical access to the coprocessor * *Re*loading 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 software into the coprocessor" on page 7.
Threats that monitor information	
Information leakage An adversary might exploit information that is leaked from the coprocessor during normal usage. 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.

Table 15. Threat considerations for a digital-signing server (continued)

Threat discussion	**Threat mitigation**
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. Usage of the cryptographic equipment should be controlled, including following the guidelines in Appendix H of the *IBM CCA Basic Services Reference and Guide frefbook for the IBM 4765 PCIe and 4764 PCI-X Cryptographic Coprocessors*. 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 functionality 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.
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 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 which 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 which 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.

Table 15. Threat considerations for a digital-signing server (continued)

Threat discussion	Threat mitigation
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.
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.

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. In the Netherlands the following applies:

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

Trademarks

Index

A

application programs
  compile 36
  link to CCA 36
auditor 9
auto-set, master key 28

B

batteries, coprocessor
  status 21

C

C programming language
  sample routine 37
  verb calls 36
choosing among coprocessors 20
clock-calendars, synchronization 21
CNI list 16
CNI utility (CCA node initialization utility)
  using, node setup 33
CNM (CCA node management utility)
  configure 21
  defaults 21
compile, application programs 36
configure
  CNM utility 21
coprocessor
  installation 3
  status, batteries 21
create
  key label 32
  master key 28
cryptographic key management 26

D

DEFAULT role
  description 22
  initial use 41
defaults
  CNM utility 21
delete
  user profile 26
description
  DEFAULT role 22
  KEKs 27
  master key 27

E

edit
  role 24, 25
establish owner command 12

F

function-control vector
  load 20

I

initial use, DEFAULT role 41
initialization of the CCA node 20

K

KEKs
  description 27
  primary 27
key label, create 32
key management, cryptographic 26
key storage
  delete keys 32
  key label, create 32
  reencipher 32

L

link to CCA, application programs 36
load command 12
load coprocessor software
  establish owner command 12
  load command 12
  reload command 12
  surrender owner command 12
logon-attempt-failure count, reset 26

M

machine readable log 13
machine-readable log 41
make-file 36
management
  cryptographic key 26
  master key 27
master key
  auto-set 28
  description 27
  management 27
  registers 27
  verification 28
master-key administration 27
master-key cloning 43

N

node
  setup, production-environment 18
  setup, test 17

O

ordering
  overview 2
  placing orders 3

P

performance, enhancing 40
permit, access-control commands 23
production-environment, node setup 18

R

reencipher stored keys 32
registers, master key 27
reload command 12
reset logon-attempt-failure count 26
restrict, access-control commands 23
role
  modify 24, 25

S

sample routine, C programming language
  make-file 36
  source code 36
  syntax 36
security-relevant data item (SRDI) 12
setup
  production-environment node 18
  test node 17
SRDI, security-relevant data item 12
status, batteries 21
stored keys, reencipher 32
surrender owner command 12
synchronization, clock-calendars 21
syntax
  verb calls, C programming language 36

T

test setup, node 17
threat considerations, digital-signing server 50
throughput, enhancing 40

U

user profile
  delete 26
  reset logon-attempt-failure count 26
utilities
  CNI 33

V

verb calls, C programming language 36
verification, master key 28

Z

zeroization of the CCA node 20

Printed in USA

SC23-6772-00