Security Policy for the Security Module with CP/Q++

part of the IBM 4758 PCI Cryptographic Coprocessor Models 002 and 023 (PCICC)

Revised 2003.08.05

Document level: 2.0, 2003.08.05

This paper provides the Security Policy for the Security Module with the CP/Q++ control program, security-sensitive portions of the IBM 4758 PCI Cryptographic Coprocessor Models 002 and 023.

IBM does not stock publications at the address given below. This and other publications related to the IBM 4758 PCI Cryptographic Coprocessor can be obtained in PDF format from the Library page reachable from http://www.ibm.com/security/cryptocards.

Readers' comments can be communicated to IBM by using the Comments and Questions form located on the product website at http://www.ibm.com/security/cryptocards, or by sending a letter to:

IBM Corporation
Department VM9A, MG81/204-3
Security Solutions and Technology
8501 IBM Drive
Charlotte, NC 28262-8563 USA

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Chapter 1. Security Policy

This paper presents the Security Policy for the Security Module with the CP/Q++ control program, security-sensitive portions of the the IBM 4758 PCI Cryptographic Coprocessors, Models 002 and 023. The policy addresses the requirements of the FIPS 140-1 standard in these sections:

- Product family
- Applicable documents
- Cryptographic module overview
- Included cryptographic algorithms
- Cryptographic module security levels
- Roles
- System objects
- Services
- External-user services
- Authentication
- Self test
- Security rules
- Cryptographic module configuration for FIPS 140-1 compliance
- Security relevant data items.

IBM 4758 Product Family

The IBM 4758 PCI Cryptographic Coprocessor (the *Coprocessor*) is a programmable PCI-board subsystem that off-loads computationally intensive cryptographic processes from the hosting server and performs sensitive tasks unsuitable for less secure general purpose computers. It is a key product for enabling secure e-business transactions and is suited for a wide variety of cryptographic applications. The CP/Q++ control program, which is the primary focus of this security policy, operates in the Coprocessor and provides a multi-tasking environment in which “user” applications can employ validated cryptographic and storage services.

Figure 1-1. Typical IBM 4758 Model 002 or 023

Figure 1-1 shows a typical, complete Coprocessor. The large steel enclosure on the left, the embedded *Security Module*, contains the security-sensitive electronics and tamper sensors and defines the cryptographic boundary of the evaluated module. The Security Module hardware, and the resident Power-On Self-Test and Miniboot software, have previously been validated under FIPS 140. The secure Security Module hardware provides the environment in which CP/Q++ is loaded and operates. To

complete a productive Coprocessor, you must add an application program. The application program must also be validated under FIPS 140 for the Security Module with CP/Q++ to operate in a validated mode.

IBM manufactures two secure Security Modules distinguished by “model number.” These modules are attached to either of two PCI boards. This results in four Coprocessor product offerings as listed in Figure 1-2. The battery backup power systems ensure operation of the tamper-detection system from the time of factory Security Module certification until the end of useful life of the Coprocessor. Each product variation employs the same Miniboot, CP/Q++, and application software.

Figure 1-2. IBM 4758 PCI Cryptographic Coprocessor Part Numbers

IBM's Product Designation	Voltage Option	Security Module FIPS 140 Validation Level	Product Part Number	Security Module Part Number
IBM 4758 Model 002	3.3 and 5 volts	4	40H9951	04K9131
IBM 4758 Model 002	5 volts	4	40H9952	04K9131
IBM 4758 Model 023	3.3 and 5 volts	3	40H9858	04K9036
IBM 4758 Model 023	5 volts	3	40H9950	04K9036

The secure Security Module contains a random number source, a general purpose CPU, specialized DES, SHA-1, and modular-exponentiation processors, storage facilities utilizing battery-protected memories, flash memory, dynamic RAM, and a microcontroller to manage state and mediate storage access.

Through a scheme of digital-signature validations, Miniboot controls the subsequent loading of software (CP/Q++ and an application program) into the Security Module. This approach to software-integrity permits customers to employ the latest function available from IBM, and/or extensions and new applications provided by the customer or third parties. Such software is generally loaded once the Coprocessor is installed in its productive environment.

The Security Module with CP/Q++ FIPS validation builds on and extends:

- The FIPS 140-1 Level 4 validation of the IBM 4758 Model 002 Security Module
- The FIPS 140-1 Level 3 validation of the IBM 4758 Model 023 Security Module

In both cases, with the addition of CP/Q++, the validation is for FIPS 140-1, level 3, complementing the Level 4 and Level 3 physical hardware and Miniboot evaluations of the IBM 4758 PCI Cryptographic Coprocessor Model 002 and Model 023 Security Module hardware.

IBM offers the Coprocessor as an end-product for use in personal computer servers. IBM also offers the Coprocessor in IBM eServer pSeries (RS/6000), iSeries (AS/400), and zSeries (S/390) servers as the *IBM PCI Cryptographic Coprocessor* (PCICC) feature.

Applicable Documents

The Library page on the Coprocessor Web site, *http://www.ibm.com/security/cryptocards*, and the IBM Research security and cryptography Web site, *http://www.research.ibm.com/secure_systems/scop.htm*, provide background and reference information for the IBM 4758 PCI Cryptographic Coprocessor.

Design goals overview:

Dyer, Perez, Smith, Lindemann. *Application Support Architecture for a High-Performance, Programmable Secure Coprocessor*, 22nd National Information Systems Security Conference, October 1999.

Programming reference for the services discussed in this security policy:

IBM 4758 PCI Cryptographic Coprocessor Custom Software Interface Reference, Version 2

Programming reference material for the CP/Q kernel:

- *IBM 4758 PCI Cryptographic Coprocessor CP/Q Operating System Overview*
- *IBM 4758 PCI Cryptographic Coprocessor CP/Q Operating System Application Programming Reference*
- *IBM 4758 PCI Cryptographic Coprocessor CP/Q Operating System C Runtime Library Reference*.

Background information on Outbound Authentication and verifying the integrity of the Coprocessor:

Verifying Type and Configuration of an IBM 4758 Device: A White Paper, IBM T.J. Watson Research Center, February 2000.

Cryptographic Module Overview

Background

The traditional notion of a cryptographic module is a black box that performs cryptographic operations. With the IBM 4758 PCI Cryptographic Coprocessor product family, the concept is extended to high-performance *secure coprocessor* devices that can perform cryptography as well as other sensitive computation beyond the reach of adversaries who may have physical access to the device.

The Security Module is built as a generic secure-coprocessor subsystem in order to enable external developers (and IBM) to build and deploy secure-coprocessor applications. Consequently, it consists of three different components:

- An application software layer (segment-3)
- A system software layer (segment-2)
- Both guarded by the Security Module's foundational physical security package and the Miniboot secure configuration-control software.

What the Coprocessor does in the field, after booting, depends on how it has been configured: what software has been loaded into the system and application layers.

CP/Q++: In order to facilitate application development, IBM created the *CP/Q++* control program for Coprocessor segment-2. CP/Q++ runs at supervisor-privilege within the Coprocessor, and offers services that simplify the development process for user-level segment-3 applications. Segment-3 applications do not have supervisor privilege and must employ the validated services of CP/Q++. These services include:

- A programming environment
- Communication with the outside world
- Secure data storage
- Cryptographic services
- And, when appropriate, debugging tools.

Validation: Previously completed FIPS 140 validations have reviewed the foundational subsystem: the Security Module guarded by Miniboot. Incorporating CP/Q++ in a FIPS evaluation enables a vendor who wishes to validate an application program written over CP/Q++ to focus on the internal behavior of his application program, rather than on internal details of CP/Q++ services and the Security Module. In short, the goal here is to do *once* what each such developer would otherwise need to do separately for each validation.

The Delta Module: For the validation with CP/Q++, the cryptographic module consists of an embedded, Security Module configured with the secure version[1] of CP/Q++ in segment-2. Note that this module can operate in FIPS 140-1 validated mode *only when* the segment-3 application loaded on top of it has also been FIPS validated.

Software Structure

The CP/Q++ control program executes at supervisor privilege, and provides a programming environment and specialized hardware services to the application program running in segment-3. Figure 1-3 on page 1-5 illustrates this structure.

As the name indicates, CP/Q++ consists of: the IBM CP/Q kernel (a mature OS for industrial embedded systems), extended with a collection of *managers*:

- The *COM Manager* handles communications
- The *DES Manager* provides DES and SHA-1 services
- The *OA Manager* provides RSA ciphering support based on certified keys managed by Miniboot and the hardware
- The *PKA Manager* provides public key services and modular math services
- The *PPD Manager* provides secure data storage services
- The *RNG Manager* provides random number services
- The *SCC Manager* handles names and associations
- The *SP Manager* provides an unfiltered conduit for the segment-3 application to communicate via the serial-port interface.

Each of these managers is implemented as an independent CP/Q *process*. Additional details of this software structure can be found in the paper *Application Support Architecture for a High-Performance, Programmable Secure Coprocessor*, 22nd National Information Systems Security Conference, October 1999.

Included Cryptographic Algorithms

CP/Q++ supports use of DES (ECB and CBC), triple-DES (ECB, Inner and Outer CBC), DES MAC (based on ANSI X9.9), SHA-1, DSS, RSA X9.31-compliant support, modular-math primitives, and IBM DES-to-CDMF key transformation.

Commercial Data Masking Facility (CDMF) Some IBM products support the *Commercial Data Masking Facility* (CDMF) encryption algorithm. Implementation of CDMF was motivated by former USA 40-bit-key export restrictions. With CDMF, a 56-bit key (in a 64-bit, 8-byte structure) is weakened so that it has the strength of a 40-bit key. The weakened key is then used with the DES algorithm to encrypt or decrypt data. Installations permitted only 40-bit encryption always weaken the key provided as part of a DES encipher or decipher call, thus performing only CDMF ciphering. With installations that enable normal DES encryption, the key provided to the encipher and decipher services can be tagged for weakening using the CDMF key-weakening process. A DES-to-CDMF key weakening service (sccTransformCDMFKey) is also provided to support transformation of keys for their use in systems only capable of normal DES encryption. With the relaxation of USA export regulations, CDMF should be avoided in new applications. CDMF key-weakening is not a FIPS-approved cryptographic operation.

[1] Version deployed without the debug probe.

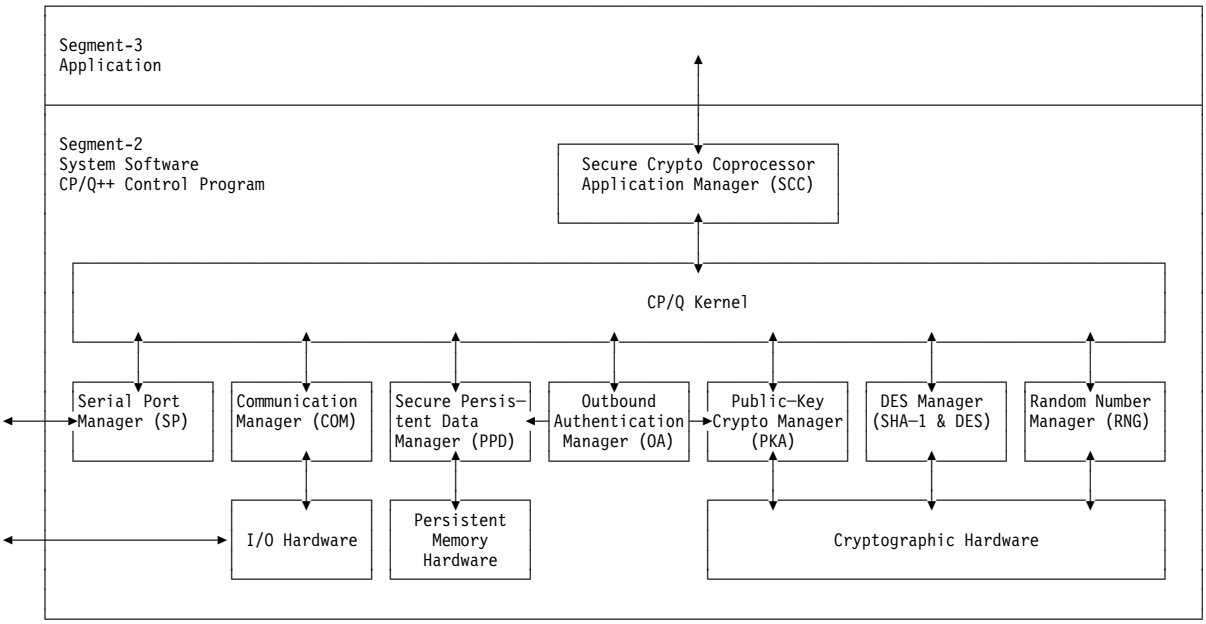

Figure 1-3. The Segment-2 Application Support Architecture in CP/Q++

Cryptographic Module Security Levels

With CP/Q++, the Security Module is evaluated at FIPS 140-1, level 3. For the entire Security Module to rate at that level, the application in segment-3 must have passed a FIPS 140 validation at that level or higher.

Figure 1-4, and Figure 1-5 on page 1-6, provide an evaluation-level summary.

Security Requirements Section	Satisfies Levels
Cryptographic module	3
Module interfaces	3
Roles and services	3
Finite state machine	3
Physical security	4
Software security	3
Operating system security	N/A
Key management	3
Cryptographic algorithms	3
EMI/EMC	3
Self test	4
Overall	3

Figure 1-4. IBM 4758 PCI Cryptographic Coprocessor Model 002 Module Security Level Specification

Security Requirements Section	Satisfies Levels
Cryptographic module	3
Module interfaces	3
Roles and services	3
Finite state machine	3
Physical security	3
Software security	3
Operating system security	N/A
Key management	3
Cryptographic algorithms	3
EMI/EMC	3
Self test	4
Overall	3

Figure 1-5. IBM 4758 PCI Cryptographic Coprocessor Model 023 Module Security Level Specification

Roles

The roles for the Security Module with Miniboot consist of Officer 1 through Officer 3, and a space of not-authenticated users. CP/Q++ partitions this set of users into two additional roles:

- The *External User* accesses CP/Q++ services from the host.
- The *Internal User* accesses CP/Q++ services from inside the application in segment-3.

Officer 2 or Officer 3 authorize a segment-3 application program. The internal and external users obtain services by formulating their service requests to the authorized segment-3 program.

System Objects

These system objects are introduced:

Application An *application* is the set of computational resources inside the Coprocessor belonging to some external Officer 3.

Processes The system can have zero or more *processes*. A process is the CP/Q construct representing “owner of resources”; informally, one can think of this as an address space. The set of processes may change over time, depending on how the application program interacts with CP/Q++.

The structure of the application in segment-3 designates one process as *FirstProc*.

Tasks Each process has zero or more *tasks* (the CP/Q construct for “thread of execution”). Each task is owned by exactly one process.

Agents An *agent* is a name, chosen at run time, that designates some specific resources. The set of agents breaks down into two overlapping subsets:

- *Process agents* are the names chosen by processes for themselves.
- *Request agents* are the names chosen by a process to represent “mailboxes” to receive communication requests from external users.

This breakdown yields three regions: an agent may be process-only, or request-only, or both.

Request A *request* is a particular communication session between an external user and the internal user, through a particular request agent which the internal user had registered.

Requests have some associated states:

- During the interval that the session exists, we say that the request is *open*.
- An open request initially has an *unread header*. Its header may then be read.
- After the session is complete, a request becomes *closed*.

Request Queue A *request queue* is a special CP/Q object created and maintained by the system software to receive messages from the External User.

Services

Services are described in three categories:

- Officer services
- Internal user services
- External user services.

Officer Services

IBM Development personnel serve as Security Officers 0, 1, and 2. The officers are responsible for the content and integrity of the code that is loadable into segments 0, 1, and 2, respectively the unchangeable basic firmware, Miniboot, and CP/Q++. The officers oversee the administration of the code-signing keys and signing facility(s) as well as the preparation of the code and the digital-signing of the code.

In addition, Officer 2 is responsible for maintaining a list of *Segment-Owner Identifiers* for segments 2 and 3, and for the certification of public keys used to validate segment-3 applications to be loaded into segment-3. Officer 2 and Officer 3 are responsible for the content and integrity of applications that they cause to be signed for loading into segment-3.

An individual in IBM Development serves as Officer 2 and Officer 3 when signing IBM's applications: PKCS#11 and CCA. Under contract, IBM Development will certify the public key of another developer and assign him a unique segment-3 owner-identifier. The party holding a certified identifier and key-pair can then perform as an Officer 3 for signing segment-3 code.

Organizations that employ the Coprocessor are responsible for loading (and reloading) of code into segments 1, 2, and 3, and for ensuring that code running in Coprocessors under their supervision is appropriate. A status query to the Coprocessor (for example, using the IBM-provided Coprocessor Load Utility) returns the name, hash-value of the code, and owner-identifier for the code segments. The query response is signed by Miniboot in the Security Module using its device key which permits the response to be externally validated using certificates from the Security Module and from IBM.[2]

Authentication of the Internal User: Since the application in segment-3 embodies the Internal-User service requests, Officer 2 and Officer 3 take on some additional responsibilities. When one of these officers prepare a program for loading into segment-3, the officer must:

- Authenticate that, within this program, the entity making requests as a specific Internal User is indeed that user
- Verify that the code has indeed been validated at the appropriate FIPS level
- And, if this is a SRDI-preserving “ordinary burn” code-load, Officer 3 should verify that the Internal user whose service requests are embodied in the new program is the same entity whose service requests were embodied in the current program.

---

[2] Note that Officer 2 could inadvertently, or maliciously, create a signed segment-3 code-load having the same name and segment-owner identifier as a legitimate code-load. However, it is computationally infeasible to provide an alternative code-load that has the same hash-value as the legitimate code-load.

Internal User Services

With CP/Q++ in segment-2, an *Internal User* accesses the "++" services of CP/Q++. The services are described next and are summarized in Figure 1-6 on page 1-17. The discussion of these services generally follows the same sequence as the full service descriptions found in the *IBM 4758 PCI Cryptographic Coprocessor Custom Software Interface Reference, Version 2* manual. Note that the SCC software libraries linked with the application in segment-3 are not formally part of the CP/Q++ evaluation. The SCC libraries perform the CP/Q mechanics to effect the service interchange between the described API and the CP/Q++ manager code. When accessing one of these services, the Internal User implicitly specifies the task and process from which the service is being called.

Signing On: The Internal User may register its existence:

- sccSignOn For this service, the Internal User provides an *agent* name and a *request queue* option. The named agent must not be in the current set of active agents.

  The service adds an agent with this name to the set of active agents, and links it as follows:

  – The new agent is added to the *request agent* subset, and is linked to the request queue specified by the queue option. (Depending on the option, this queue will either be a particular preexisting queue, or a new one created as part of the service.)
  – The caller's task is recorded as the *signer-on* of this agent.
  – If no process agent is currently associated with the caller's process, then the new agent is also added to the *process agent* subset, and is associated with the caller's process.

Host Communication Request Services: Request services permit the Internal User to handle communication sessions (*requests*) between an External User and a request agent.

These requests work as follows:

- The External User *opens* a request to a request agent. This request consists of a *request header* and specifications for *data buffers*, each of which may be *to-host* or *to-Coprocessor*.

  (Note that the product documentation uses the terms "input" and "output" for these buffers, but *from the perspective of the host;* from the Coprocessor's perspective, the terms are backwards.)

- When the Internal User receives the request, these two parties may exchange data through these buffers.
- When the request is complete, the Internal User *closes* the request.

An External User may open more than one request to the same agent at the same time; however, each data exchange must occur within the context of exactly one request.

The Internal User's communications are supported via the following services:

- sccGetNextHeader This service enables the Internal User to read the next open request for any one of a specific set of request agents.

  The caller specifies a request queue and a timeout option.

  The service then provides the caller with the first request header on that queue (and removes it from the queue). If the queue remains empty throughout the timeout preference, an error will be returned (unless an "infinite" timeout has been requested).

- sccGetBufferData This service permits the Internal User to read a "to-Coprocessor" buffer that is part of a specific open request.
- sccPutBufferData This service permits the Internal User to write a "to-host" buffer that is part of a specific open request.

- sccEndRequest This service permits the Internal User to close a specific open request.

Hashing Services: Hashing services are provided by the DES Manager to the Internal User:[3]

- sccSHA1 performs SHA-1 hashing for the Internal User subject to:
  - the data length cannot exceed 0x01FFFFFF bytes
  - the data length must be a multiple of 64 bytes for the first and middle portions of a segmented calculation.

DES Services: DES services are provided by the DES Manager to the Internal User:

- sccDES8bytes Performs DES on one 8-byte block.
- sccDES Performs DES in ECB or CBC mode on an arbitrary length of data subject to:
  - the data length must be a multiple of 8 bytes
  - the data length cannot exceed 0x01FFFFFF bytes
  - if the MAC flag is not set, the destination count must be a multiple of 8 and the source count plus plus the padding count must be less than or equal to the destination count.
- sccDES3Key Performs three successive DES operations on one 8-byte block, each with a unique key.
- sccTDES Performs Triple-DES in ECB or CBC (outer) on an arbitrary length of data subject to:
  - the data length must be a multiple of 8 bytes.
- sccEDE3_3DES Performs inner-CBC 3DES on an arbitrary length of data subject to:
  - the data length must be a multiple of 8 bytes
  - the data length cannot exceed 0x01FFFFFF bytes.
  - The data length cannot be zero.

  Note: The sccEDE3_3DES service provides a variation of TDES that is not included in the FIPS 3 DES standard.
- sccTransformCDMFKey Transforms an arbitrary DES key into a CDMF[4] key (whose effective length is only 40 bits).

Public-Key Services: Public Key services are provided by the PKA Manager to the Internal User:

- sccRSAKeyGenerate Generates an RSA keypair. The X931 option is ANSI X9.31 compliant.
- sccRSA Performs basic RSA encipher and decipher operations. (Note that these operations are not sign/verify, but basic encipher/decipher operations. These basic operations are outside the scope of the ANSI X9.31 standard by themselves; however, an application program can use them to build an ANSI X9.31-compliant RSA implementation.)
- sccComputeBlindingValues Generates blinding values for use in sccRSA to ensure that operation time leaks no private information.[5]
- sccDSAKeyGenerate Generates a DSA keypair.
- sccDSA Signs a block of data using the DSA algorithm, or verifies a DSA signature on a block of data. Optionally accepts or returns the hash of the data block.

---

[3] In the IBM 4758 PCI Cryptographic Coprocessor Model 001 and 013, SHA-1 hashing service is provided by library code which segment-3 application developers link in at build-time. For Models 002 and 023, this service is provided by hardware and the DES Manager, via the same CP/Q++ API as available for the earlier models.

[4] See “Commercial Data Masking Facility” on page 1-4 for a discussion of CDMF. Segment-3 applications using this service must clearly delineate its use as this would render the application non-usable in a FIPS-approved mode.

[5] This service is unnecessary for the IBM 4758 PCI Cryptographic Coprocessor Model 002 and 023 hardware, because the modular-exponentiation hardware itself resists such timing attacks. The service is provided for upward compatibility for programs written for the IBM 4758 PCI Cryptographic Coprocessor Models 001 and 013.

- sccModMath Provides basic modular math services (which, for example, the application programmer might use to implement elliptic curve crypto systems).

Random-Number Generation Services: Random number services are provided by the RNG Manager to the Internal User:

- sccGetRandomNumber Provides 64-bits of random data from either the internal hardware random number generator, the pseudo random number generator seeded from the hardware, or a mixture. The mixture option provides hardware numbers until the hardware-number-pool is exhausted, at which time the manager provides pseudo random numbers until more hardware numbers become available.

  The hardware-seeded pseudo random number generator is the FIPS-140-approved option.

  This service also provides options:

  – Even parity, odd parity, and full-random. The low-order bit in each byte is set to even or odd considering all bits within the byte, or random as requested.

  – Not weak. This option filters out the values that represent weak, semi-weak, and possibly semi-weak DES keys as listed at the end of the Master_Key_Process verb description in the *IBM 4758 CCA Basic Services Reference and Guide*.[6]

- sccTestRandomNumberAsync Performs the full complement of FIPS 140-1 specified statistical tests on the Internal User's selection of random-number output, either the raw hardware output or the output of the pseudo random number generator (which is seeded periodically from the hardware). The tests are: poker, runs, and monobit. The “long runs” test operates continuously and issues a failure should four consecutive equal-valued 64-bit numbers occur. This is not a specific additional part of this service.

Persistent Memory Services: Secure Persistent Memory services are provided by the PPD Manager to the Internal User. These services permit the Internal User to create, read, and write data items in the device's nonvolatile storage: FLASH and battery-backed-up RAM (BBRAM).

Names for these items are chosen by the Internal User who creates them, and are unique for all calls linked to the same process agent. (That is, the space of PPD items can be thought of as a file system, with a separate directory for each process agent.) This means that a caller's process must be associated with a process agent in order to actually use PPD services. This association is established in one of two ways:

- Currently, within an application, the process designated as *firstproc* is automatically associated with a default *firstproc* process-only agent.
- Any other process becomes associated with a PPD Directory via an explicit sccSignOn.

The security of these items depends on the underlying storage medium:

- BBRAM contents are zeroized upon tamper. BBRAM is directly suitable for sensitive data.
- FLASH contents may be visible to an adversary who physically opens the Coprocessor. Consequently, sensitive data should be first encrypted before being stored in FLASH, with the encryption key being stored in BBRAM. (The sccSavePPD service includes options to transparently encrypt with DES or TDES.)

Where failure might otherwise leave PPD in an unknown state, the services ensure *atomicity* of access, with one exception —CP/Q++ also provides *one* service that allows non-atomic read/write access to BBRAM storage, for applications that require improved performance.

---

6 There is an error in Schneier's *Applied Cryptography*, second edition, Table 12.13; the `X'F0'` should be `X'F1'`.

These storage services are provided by the PPD Manager for the Internal User:

- sccQueryPPDSpace This service reports the amount of free space that CP/Q++ has left for PPD storage in FLASH and/or BBRAM. (This value may change with subsequent PPD requests, and with internal PPD activities, such as garbage collection.)
- sccCreate4UpdatePPD This service enables the caller to allocate a BBRAM region for a a PPD item with a specified name, and optional data with which to populate this space. (If the data is not explicitly provided, the region will be populated with zeros.)

  If an item with this name already exists in the PPD directory associated with the caller's process agent, then that item is deleted and this new item is created.

  This operation is atomic.
- sccSavePPD This service enables the caller to save a PPD item with a specified name in a specified medium (FLASH or BBRAM). If an item with this name already exists in the PPD directory associated with the caller's process agent, then the item is replaced; otherwise, the item is created.

  This operation is atomic.
- sccUpdatePPD This service enables the caller to update the portion of the BBRAM region associated with a preexisting PPD item (in the PPD directory associated with the caller's process agent).

  This operation is *not* atomic. (If the Internal User desired atomicity of BBRAM update, he should use the sccSavePPD service above.) However, failures of this non-atomic operation lead to a mingling of old data, new data, and indeterminate bytes. In particular, data from *another* PPD Item is not garbage-collected into space here.
- sccGetPPDDir This service enables the caller to obtain a list of names of PPD items currently in the PPD directory associated with the caller's process agent.
- sccGetPPDLen This service enables the caller to obtain the length of a particular PPD item in the PPD directory associated with the caller's process agent.
- sccGetPPD This service enables the caller to retrieve a PPD item with a specified name from the PPD directory associated with the caller's process agent.
- sccDeletePPD This service enables the caller to delete a particular PPD item from the PPD directory associated with the caller's process agent.

  This operation is atomic.
- sccDeleteAllPPD This service enables the caller to delete *all* PPD items from the PPD directory associated with the caller's process agent.

  This operation is atomic (for the entire set).
- sccSetClock This service enables the Internal User to set the hardware time-of-day clock to an arbitrary value.

Configuration Services: The SCC Manager provides several miscellaneous services to the Internal User:

- sccGetConfig This service provides the caller with generic, non-sensitive status information about the device's current configuration.
- sccClearILatch The device provides an external input that can be connected to an external sensor, such as a cover switch on the host cabinet. Triggering of this input sets an internal “intrusion” latch This service enables the caller to clear that latch.

  Note that this “intrusion latch” only records whether an external signal has occurred. This latch is provided solely as a courtesy for certain users, and *does not trigger zeroization*. Rather, the physical

security and zeroization of the device, as documented in the FIPS 140 validation of the Coprocessor hardware, relies on internal sensors independent of this signal.

- sccClearLowBatt The device has an internal sensor that monitors battery voltage and sets a latch when the battery drops low enough to indicate an urgent need to be replaced, but high enough so the device and its tamper-response circuitry is still functional. (Should the battery drop too low, the tamper-response circuitry will activate and zeroize the Coprocessor.) This service enables the caller to clear this latch.

Outbound Authentication Services: The OA Manager provides these key management and ciphering services to the Internal User. Note that the Internal User does not have access to the key values, only to the use of the keys. This enables Miniboot to ensure appropriate destruction of the private keys upon a tamper occurrence and during the routine loading of code into the segments. Through a certificate chain, OA public keys can be validated back to the published IBM 4758 PCI Cryptographic Coprocessor root public key.

The Internal User can call these services:

- sccOAGetDir Count or list all key certificates.
- sccOAGetCert Retrieve a certificate.
- sccOAGenerate Generate an keypair and obtain a certificate with the generated public key.

  The sccOAGenerate user service can generate RSA and DSA keys. It creates the private (retained) and public (exportable) sections of the key object. The DSA key generation service follows the method described in FIPS-186. The RSA key generation service provides a call option to generate ANSI X9.31-compliant RSA key pairs.
- sccOADelete Delete a certificate and the corresponding keypair. Note that the private key may have previously been deleted because of actions by Miniboot.

  The sccOADelete user service does not erase the key storage area. Rather the call erases the key directory information which is required to access the memory containing the key. The memory region containing the “deleted” key is contained in BBRAM (zeroized on tamper) or flash memory (encrypted, unavailable after BBRAM has been erased).

  Note that in the lifecycle of the typical persistent OA key (object), the private key is erased before the sccOADelete call. In such cases, public parts of keys are retained in the OA object, and the subject of the sccOADelete call does not contain any secret information.
- sccOAPrivOp Perform an operation using one of the keys of the keypair.
- sccOAVerify Verify the signature in one certificate using the public key from another certificate.
- sccOAStatus Obtain information about the contents of the Coprocessor.

Serial Port Services: The Serial Port Manager provides the Internal User access to the serial port electronics. The serial port is not employed for any purpose by other portions of CP/Q++. The security implications of the serial port are strictly a function of the segment-3 application. If a segment-3 application does not open the serial port through use of the ASYNopen() service, the serial port is ignored by CP/Q++.

These services are available to the Internal User:

- ASYNopen Open the serial port.
- ASYNioctl Change communication parameters.
- ASYNread Read data from the serial line.
- ASYNwrite Write data to the serial line.

- ASYNdrain Wait for serial-port operation to complete.
- ASYNflush Purge the serial-port buffer.
- ASYNclose Close the serial port.

External-User Services

The CP/Q++ system offers three classes of services to the External User.

Coprocessor Communication Request Services: Provides the mechanism to communicate with the Internal User program within the Security Module.

- sccRequest An External User can enqueue a request header for a specific agent, 'G'.

  If G is currently an active request agent, then the request becomes open and its header is placed on the request queue to which G is linked. Otherwise, the service fails.

Device Driver Services: Typically, the host-side device driver employs this service class transparently to the External User.

- reset This service causes the entire device to undergo a hardware and software reset. As a side-effect, this closes all open requests.
- ABORT_REQUEST This service causes an abnormal termination of a currently open request. (The Coprocessor-side Internal User which was participating in this request will notice the abort via an error code the next time it asks for a service pertaining to that request.)

Status Services: This service class provides status information for the External User.

- sccGetConfig, issued externally, returns an sccAdapterInfo_t data structure like the internal version.
- sccQueryAgent permits the External User to determine whether or not a specified agent is currently signed on.

Authentication

Officer 1, Officer 2, and Officer 3 are authenticated by Miniboot; see the Miniboot documentation for details (e.g. in the Security Policy for the Coprocessor Security Module hardware).

The Internal Users are authenticated (by identity) by the officer (Officer 2 or Officer 3) who authorizes the program load for segment-3.

The CP/Q++ module does not authenticate the External User. The host API is the External User. Segment-3 applications authenticate External Users as required.

Self Test

Before the invocation of CP/Q++, the Security Module performs various checks of the hardware including the random number generator. When control is passed to CP/Q++, these additional self-tests are invoked:

Random number generator statistical tests The random number statistical tests are designed to perform the Monobit, Poker, and Runs tests as specified in TE11.16.01 and TE11.16.02 found in the *Derived Test Requirements for FIPS 140-1* document.

Continuous random number testing The random bit-stream is monitored for 64-bit repeating patterns. If three successive blocks are equal, operation is terminated. The test monitors the output of the function returning the random value, regardless of the actual source of randomness (pure hardware, software with hardware bits if available, or hardware-seeded software).

SHA known answer test CP/Q++ runs a SHA-1 known-answer test during its initialization.

RSA and DSA key consistency CP/Q++ checks the consistency of all generated public and private RSA and DSS keys.

DES and TDES known answer test CP/Q++ runs a DES and TDES known-answer tests during its initialization.

Persistent data (SRDI) checks CP/Q++ runs checks during initialization to determine that BBRAM and Flash storage are available and that the TDES keys used to protect sensitive data are not corrupted. Segment-3 application programs are responsible for checking the integrity of data they store using the PPD Manager services.

Security Rules

CP/Q++ is not intended to protect one application-level process from another one.

The segment-3 application program shall comply with these rules:

- The Internal User will access the CP/Q++ services via the official library calls, rather than trying to send messages directly to the appropriate managers.
- When the Internal User supplies request-ids as a parameter to services, he shall have earlier obtained these ids from a request header. (However, the request header may have been obtained by a call from a different task from the one calling the request service.)
- It may be possible in some circumstances for a malicious task to forge a request header and add it to a request queue, or to forge a service response message for some other task who has called that service asynchronously. The segment-3 program shall not do that.
- If the segment-3 application employs the serial port through the use of the ASYNopen() and related services, the application program is responsible for information that it exchanges over the serial port. The security implications of such exchanges must be considered in the validation of such application programs.

Cryptographic Module Configuration for FIPS 140-1 Compliance

For a Security Module with CP/Q++ to be considered fully FIPS 140-1 compliant:

1. The hardware must be loaded with:
   - a FIPS 140 validated Miniboot in segment-1.
   - a FIPS 140 validated CP/Q++ in segment-2. The segment ‘owner identifier’ must be 2. Do not use identifier 6 in a production system.
   - a FIPS 140 validated application in segment-3.
2. The External User must observe any restrictions documented for FIPS-140 compliant use of the segment-3 application.

Security Relevant Data Items

Through the facilities of the PPD Manager, CP/Q++ manages the storage of SRDI items.

Items

The SRDI items consist of:

- The PPD items that the Internal User stores in BBRAM
- The PPD items that the Internal User stores in FLASH that the manager encrypts using the PPD TDES Key
- The OA key-pairs and related certificates managed by the OA Manager
- The “PPD TDES Key” used to protect SRDIs stored in FLASH.

Generally, PPD items stored in BBRAM or flash memory result from service calls by the application running in segment-3. With the exception of the flash memory encryption key (the “PPD TDES Key,” see below), and OA Manager certificates and keys (also see below), CP/Q++ does not itself cause the generation, storage, retrieval, or deletion of PPD objects.

A particular PPD object that never resides in flash is the PPD TDES key. This double-length TDES key is retained in BBRAM and used to encipher and decipher flash contents. Multiple copies of the key are retained in BBRAM, and are synchronized if a discrepancy is detected. The PPD TDES Key never leaves the security enclosure or is exposed to any user in segment-3.

Associated with each item (“data block”) saved in nonvolatile memory is a name chosen by the application that owns the data. The name is assigned when the item is first saved or when space to hold the item is allocated. Subsequent requests to read, write, update or delete the item refer to it by name. The PPD Manager holds the name index in flash memory.

To be secure, segment-3 applications must request the PPD manager to encrypt SRDI items retained in flash memory. The PPD Manager performs en/decryption transparently to the requestor. (Of course, an application could itself encrypt items for storage.)

OA key objects contain RSA or DSA keys and a corresponding certificate. The objects are partitioned so that only the public-key and certificate information can be exported. The private-key information is never exported. The number and type of OA key objects depends on the segment-3 application requesting the services of the OA Manager.

The PPD Manager deletes items by removing the item-name from the index rendering the item not retrievable. Subsequent garbage collection will recover the unused storage space. Detection of a tamper event will cause BBRAM to zeroize including the PPD TDES key required to decrypt secure items in flash memory.

Modes of Access

The PPD TDES Key is created and used by the PPD Manager, and stored redundantly as SRDIs in BBRAM by the manager. This key is not accessible to any internal or external user service.

PPD items may be accessed in the following modes:

- Update-in-place BBRAM items may be *allocated*
- Items may be *read*
- Items may be *atomically written*
- Update-in-place BBRAM items may be *non-atomically written*
- Items may be *deleted*
- Items may be *listed*
- Items may be *sized*
- Items may be *zeroized*

The OA Manager employs the PPD Manager for the persistent storage of OA-related information. Internal-User access to these items is mediated by the OA Manager.

- Key pairs and the related public-key certificate may be *allocated*
- Items may be *deleted*
- Items may be *listed*
- Items may be *zeroized*
- Keys can be *used*
- Certificates can be *read* (private keys cannot be read by the Internal User).

Roles vs. Services vs. SRDI vs. Modes of Access

The CP/Q++ SRDIs consist of:

- The PPD TDES key used to encrypt items in flash storage; no services directly access this SRDI.
- Data items managed by the PPD Manager including application supplied items and OA keys and certificates stored for the OA Manager.
- OA keys and certificates stored and deleted by Miniboot.

Access to the PPD-managed SRDI data items is discussed in terms of:

- Officer services
- External User services
- Internal User services.

Figure 1-6 on page 1-17 shows how and by whom SRDIs are accessed. The FIPS 140 security policy covering the Security Module hardware and Miniboot defines roles for four *Officers*. Officer 0 (not listed in Figure 1-6) is responsible for the release-to-production and supervision of hardware manufacturing, POST-0 and Miniboot-0 content, and the introduction of an initial Officer 1 approved POST-1 and Miniboot-1 into segment-1.

Officer Services: The Officers, 1, 2, and 3, digitally sign commands which, when used, affect the software content of a Security Module. Their signed commands are distributed to those with physical control of IBM 4758s. The use of certain commands affects the SRDIs within a specific Security Module as shown in the upper portion of Figure 1-6. These services were certified as part of the hardware and Miniboot FIPS 140 validation.

The following actions will *zeroize* all of the PPD SRDIs:

- Coprocessor hardware detecting a tamper event
- Miniboot detecting a hardware or software malfunction rendering a segment-3 application unsafe to execute.
- Loading an initial installation of a segment-3 program.
- Changing the execution environment of segment-3 for which Officer 2 or Officer 3 did not indicate trust when they signed the command.

External User Services: The external user services have no impact on SRDIs. An external user must employ a segment-3 program coded with internal user services to affect SRDIs.

Note: An “external user” controls the use of code-loading files signed by the Officers.

Internal User Services: The lower part of Figure 1-6 on page 1-17 presents how the Internal User services access SRDIs. Unlisted internal user services do not interact with SRDIs.

Figure 1-6. Roles, Services, Modes of Access

Service	Role					OA SRDI						Appl.		SRDI Access
	Officer			User		OpSys		Epoch		Config.		SRDI		
	1	2	3	Ext	Int	Pvt	Cert	Pvt	Cert	Pvt	Cert	Key	Item	
Reload segment-1 (“Remburn-1”) See Note 1.	√			(√)		DC	C			D				Miniboot atomically updates, creates
Establish owner 2	√			(√)										(No CP/Q++ SRDI actions)
Load segment-2 (“Emburn-2”) See Note 1.	√			(√)		DC	DC	D	D	D	D	D	D	Miniboot atomically destroys, updates, creates
Reload segment-2 (“Remburn-2”) See Note 1.		√		(√)		DC	C			D				Miniboot atomically updates, creates
Surrender segment-2		√		(√)		D	D	D	D	D	D	D	D	Miniboot atomically destroys
Establish owner 3		√		(√)										(No CP/Q++ SRDI actions)
Load segment-3 (“Emburn-3”)	√			(√)				D	D	D	D	DC	D	Miniboot atomically destroys, updates, creates
Reload segment-3 (“Remburn-3”)			√	(√)						D				Miniboot atomically updates, creates
Surrender segment-3			√	(√)				D	D	D	D	D	D	Miniboot atomically destroys
sccQueryPPDSpace					√									N/A
sccSavePPD					√								UC	Item gets atomically written to specified value
sccCreate4UpdatePPD					√								C	Item gets allocated and atomically written to zeros
sccUpdatePPD					√								U	Item gets non-atomically updated
sccGetPPD					√								R	Item gets read
sccGetPPDDir					√								R	All items get listed
sccGetPPDLen					√								R	Item gets sized
sccDeletePPD					√								D	Item gets deleted
sccDeleteAllPPD					√								D	All items get deleted
sccOAGetDir					√		R		R		R			All OA items get counted or listed
sccOAGetCert					√		R		R		R			Certificate is read
sccOAGenerate					√			C	C	C	C			Keypair and certificate are atomically written
sccOADelete					√			D	D	D	D			Keypair and certificate are atomically deleted
sccOAPrivOp					√	R	R	R	R	R	R			Key is read by the OA Manager, but not returned to the Internal User
sccOAVerify					√	R	R	R	R	R	R			Keys and a certificate are read by the OA Manager, but not returned to the Internal User

User Ext External user
User Int Internal user
OpSys Operating system OA keys
Epoch Epoch OA keys
Config. Configuration OA keys
Pvt Private key
Cert Certificate
Seg-3 Segment-3

(√) Someone local to the Coprocessor must initiate use of the code-loading command signed by a Security Officer. These are commands acted upon by Miniboot and can have an impact on SRDIs employed by CP/Q++ and segment-3 applications.
D Destroy if any
DC Destroy if any, create new
C Create new, retain any current
UC Update or create
R Read
Note 1: If Officer 3 has signed segment-3 code such that changes to segment-1 or -2 code are untrusted, or untrusted if not countersigned, these actions cause an effect similar to ‘Surrender segment-3’.

IBM

Revised 2003.08.05

PDF File