Chapter 2

Security Policy

2.1 Scope of Document

This document presents the security policy for FIPS 140-1 validation of the CP/Q++ software layer for Model 1-class devices in the IBM 4758 secure coprocessor family.

This “delta validation” builds on earlier work:

- the **Level 4** FIPS 140-1 validation of the IBM 4758 Model 1 coprocessor, and
- the **Level 3** FIPS 140-1 validation of the IBM 4758 Model 13 coprocessor.

2.2 Applicable Documents

The following documents may be useful.

An overview of the design goals and internal structure of the CP/Q++ software layer:

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

The main programming reference for the services discussed in this document:

- *IBM 4758 PCI Cryptographic Coprocessor Custom Software Interface Reference.* October 21, 1999 version.

(Note that follow-on versions of this manual exist for follow-on "Model 2" hardware, which has additional features that do not apply here.)

The main reference for the CP/Q kernel that this module is built on:

- *IBM 4758 PCI Cryptographic Coprocessor CP/Q Operating System Application Programming Reference.* Version 1 Driver 16.

An overview of different ways a 4758-family device may be configured:

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

A recent paper containing a concise overview of the FIFO structures supporting I/O and DES in the 4758 module:

- Lindemann, Smith. *Improving DES Hardware Throughput for Short Operations.* Research Report, IBM T.J. Watson Research Center. Draft, May 1, 2000.

2.3 Cryptographic Module Overview

2.3.1 Background

The traditional notion of a cryptographic module is a black box that performs cryptographic operations.

With our IBM 4758 product family, we have been extending this concept to high-performance *secure coprocessors*: devices that can perform high-performance cryptography *as well as other sensitive computation* beyond the reach of adversaries who may have physical access to the device.

We built our device as a generic secure coprocessor platform, in order to enable external developers (as well as IBM) to build and deploy secure coprocessor applications. Consequently, it consists of three different components:

- an application software layer ("Layer 3"), and
- a system software layer ("Layer 2"),
- both guarded by the foundational physical security package and "Miniboot" security control software.

What the module 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 OEM application development, we developed the *CP/Q++* package for Layer 2. CP/Q++ runs at supervisor-privilege within the card, and offers features that simplify the development process for user-level Layer 3 applications. These features include:

- a programming environment,
- communication with the outside world,
- secure data storage,
- cryptography,
- and (when appropriate) debugging tools.

Validation Our previous work validated the foundational platform: the hardware, guarded by Miniboot. (We note that our Model 1 work was the world's first FIPS 140-1 Level 4 validation.)

The purpose of this present validation is to enable a vendor who wishes to validate a 4758 application program written over CP/Q++ to focus on the internal behavior of their application program, rather than on internal details of CP/Q++.

In short, our goal here is to do *once* what each such OEM would otherwise need to do separately for each validation.

The Delta Module For this present validation, the cryptographic module consists of a Model 1 or Model 13 IBM 4758 device, configured with the deployment version of CP/Q++ in Layer 2.

Note that this present module can operate in validated mode *only when* the application loaded on top of it has also been validated.

2.3.2 Software Structure

Our CP/Q++ layer executes at supervisor privilege, and provides a programming environment and specialized hardware services to the application program in Layer 3.

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*:

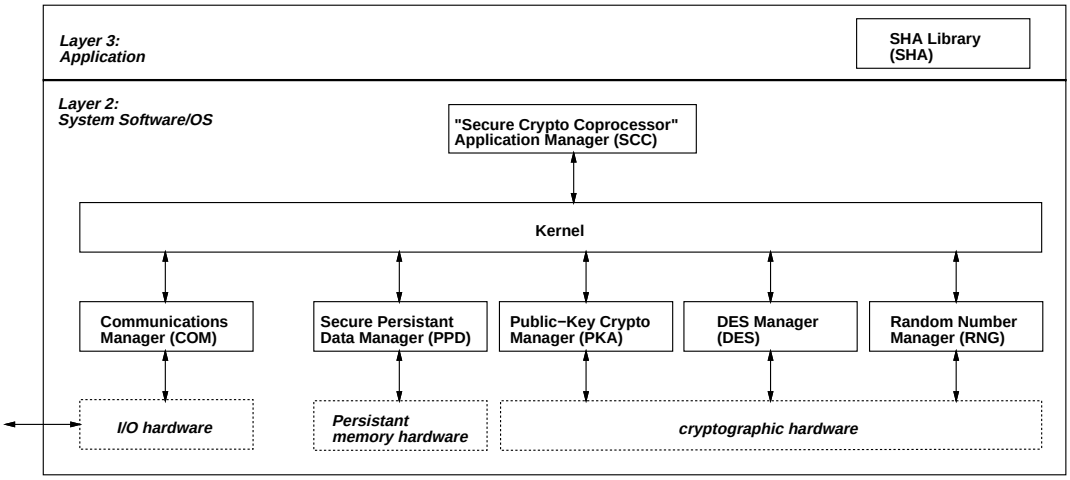

Figure 2.1 The application support architecture within Layer 2.

- the *SCC Manager* handles names and associations;
- the *COM Manager* handles communications;
- the *DES Manager* provides DES services;
- the *PKA Manager* provides public key services and modular math services;
- the *PPD Manager* provides secure data storage services; and
- the *RNG Manager* provides random number services.

Each of these managers is implemented as an independent CP/Q *process*.

Additionally:

- the *SHA Library* provides (via compile-linked library) SHA services. (This is subset of the `scclib`, which also provides wrappers for applicatino code to use to call the above services.)

Figure 2.1 illustrates this architecture.

Additional details of this software structure can be found in our “Application Support Architecture” paper (Section 2.2).

2.3.3 Included Algorithms

This module includes DES, SHA-1, and DSS, as well as RSA encipher/decipher and modular math primitives.

2.4 Cryptographic Module Security Level

This module is intended to be Level 3. See Table 2.1.

However, in order for this Layer 2 software to run at that level, any program loaded into Layer 3 must also have been validated at that level.

(Note that only a few of the formal model tasks separate us from a Level 4 in the “Software Security” category, and hence from an overall Level 4 when built on the Model 1 platform.)

Security Requirements Section	Satisfies Levels
Cryptographic Module	1,2,3,4
Module Interfaces	3,4
Roles and Services	3,4
Finite State Machine	1,2,3,4
Physical Security	4 (if Model 1); 3 (if Model 13)
Software Security	3
Operating System Security	N/A
Key Management	3,4
Cryptographic Algorithms	3,4
EMI/EMC	3,4
Self Test	1,2,3,4

Table 2.1 Module Security Level Specification. The CP/Q++ module is overall Level 3 for either the Model 1 or Model 13 platforms.

2.5 Roles

As noted earlier, this module is built upon an IBM 4758 Model 1 or Model 13 secure processor platform, which itself is a FIPS validated module. This platform's roles consist of Officer 0 through Officer 3, and a space of unauthenticated users.

This module builds on this platform by partitioning the set of users into two additional roles:

- The *external user* accesses module services from the host.
- The *internal user*, who accesses module services from inside the application Layer 3. These entities obtain module services by formulating their requests into a program, which Officer 2 or Officer 3 then authorizes for loading into Layer 3 in that module.

The CP/Q++ software under validation permits zero or one internal users.

As with our platform validation, Officer 0 relates to the "Cryptographic Officer 0" role.

2.6 System Objects

We introduce some additional system objects.

Application We define an *application* to be the set of computational resources inside the card belonging to some external Officer 3.

Processes The system can have one or more *processes*. A process is the 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 Layer 3 designates one process within each application 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 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.

2.7 Services

2.7.1 Officer Services

This module is built upon a validated IBM 4758 platform, whose services are used to configure and maintain this module. Table 2.2 summarizes the IBM 4758 security policy in this context.

When the 4758 platform is configured as this delta-module, the 4758 officers have these responsibilities:

- Officer 0 and Officer 1 must ensure that Layer 1 contains the proper Miniboot software.
- Officer 1 and Officer 2 must ensure that Layer 2 contains CP/Q++.
- Officer 2 is responsible for initial loads of Program 3, and SRDI-clearing reloads.
- Officer 3 is responsible for SRDI-preserving reloads of Layer 3 ("Ordinary Burn 3").

Authentication of Internal User. Since, in this delta-module, Layer 3 embodies the internal user service requests, Officer 2 and Officer 3 take on some additional responsibilities. When one of these officers invoke the base platform "Layer 3 load" services in the context of this delta-module, 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 for this Layer 3 has indeed been validated at the appropriate FIPS level;
- and, if this is an "ordinary burn" that preserves SRDI, verify that the internal user whose service requests are embodied in the new Layer 3 is the same entity whose service requests were embodied in the current Layer 3.

2.7.2 Internal User Services

Within this Module, an Internal User accesses the "++" services of CP/Q++.

Table 2.3 summarizes these services.

When accessing one of these services, the internal user implicitly specifies the task and process from which the service is being called.

The discussion of these services tries to follow the same sequence as the *IBM 4758 PCI Cryptographic Coprocessor Custom Software Interface Reference*.

Signing On The internal user may register its existence:

- `sccSignOn` For this service, the caller 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 pre-existing 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.

Requests 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 R 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-card*.

  (Note that the product documentation uses the terms "input" and "output" for these buffers, but *from the perspective of the host;* from the module'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 R 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 one 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.

- `sccGetBufferData` This service permits the internal user to read a "to-card" buffer that is part of a specific open request R.

- `sccPutBufferData` This service permits the internal user to write a "to-host" buffer that is part of a specific open request R.

- `sccEndRequest` This service permits the internal user to close a specific open request R.

DES Services DES services are provided by the DES Manager.

- `sccDES8bytes` This service permits the internal user to perform DES on one 8-byte block.

- `sccDES` This service permits the internal user to perform DES on an arbitrary length of data.

- `sccDES3Key` This service permits the internal user to perform three successive DES operations on one 8-byte block.

- `sccEDE3_3DES` This service permits the internal user to perform inner-CBC 3DES on an arbitrary length of data.

- `sccTransformCDMFKey` This service transforms an arbitrary DES key into a CDMF key (whose effective length is only 40 bits).

Hashing Services In the 4758 Model 1, hashing services are provided by library code which developers link in at build-time.

- `sccSHA1` performs SHA-1 hashing for the internal user.

(In Model 2, these services are provided by hardware, via an API provided by CP/Q++ for Model 2.)

Public Key Services Public Key services are provided by the PKA Manager.

- `sccRSAKeyGenerate` This service generates an RSA keypair for the internal user. The X931 option was designed to be ANSI X9.31 compliant.
- `sccRSA` This service performs basic RSA operations for the internal user. (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; hoewever, an application program can use them to build an ANSI X9.31-compliant RSA implementation.)
- `sccComputeBlindingValues` This service generates blinding values, for the internal user to use in `sccRSA` to ensure that operation time leaks no private information. (This service is unnecessary for our Model-2 hardware, where RSA hardware itself resists such timing attacks.)
- `sccDSAKeyGenerate` This service generates a DSA keypair for the internal user.
- `sccDSA` This service performs a DSA operation for the internal user.
- `sccModMath` This service provides basic modular math services (which, for example, the application programmer might use to implement elliptic curve cryptosystems).

Random Number Generation Services Random number services are provided by the RNG Manager.

- `sccGetRandomNumber` This service provides the internal user with 64-bits of random data from the internal hardware random number generator.

Persistent Memory Services Secure Persistent Memory services are provided by the PPD Manager.

These services permit the internal user to create, read, and write data items in the device's nonvolatile storage: FLASH and BBRAM.

Names for these items are chosen by the caller who creates them, and are unique for all callers 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 vanishes upon tamper, and is directly suitable for sensitive data.
- FLASH contents may be visible to an adversary who physically opens the device (since the device may not have the time or power to erase FLASH upon detection of tamper). 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, our services ensure *atomicity* of access, with one exception—we also provide *one* service that allows non-atomic read/write access to BBRAM storage, for applications that require improved performance.

These are the PPD services:

- `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.)
- `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.
- `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 proecss agent, then that item is deleted and this new item is created.

  This operation is atomic.
- `sccUpdatePPD` This service enables the caller to update the portion of the BBRAM region associated with a pre-existing 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.
- `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.
- `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.
- `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).

Configuration Services CP/Q++ provides a generic status service available to the internal user:

- `sccGetConfig` This service provides the caller with generic, non-sensitive status information about the device's current configuration.
- `sccSetClock` This service enables the internal user to set the hardware time-of-day clock to an arbitrary value.
- `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 earlier validation, 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 fire and zeroize the device.)

  This service enables the caller to clear this latch.

2.7.3 External User Services

The CP/Q++ system offers three classes of services to the external user.

Request Services One class are standard calls available to host-side applications.

- `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 The second class of services are typically used by the host-side device driver, 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 card-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 The third class of services provides status information.

- `sccGetConfig`, issued externally, returns an `sccAdapterInfo_t`, like the internal version.
- `sccQueryAgent` permits the external user to determine whether or not a specified agent is currently signed on.

2.8 Authentication

Officer 0, Officer 1, Officer 2, and Officer 3 are authenticated by Miniboot; see the Miniboot documentation for details.

The internal users are authenticated (by identity) by the officer (Officer 2 or Officer 3) who authorizes the program load for Program 3.

The external user has no authentication (within this module).

2.9 Security Rules

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

The Layer 3 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 Layer 3 program will not do that.

2.10 Module Configuration for FIPS 140-1 Compliance

The Layer 3 program is FIPS-validated, and executed only in a mode compliant with its validation.

The officers use Miniboot in a FIPS-compliant mode to configure the card.

2.11 Security Relevant Data Items

2.11.1 Items

For this delta-module, the SRDI consists of:

- The PPD items that the internal user stores in BBRAM

(Modules that add an OEM application to this module will likely have more complex SRDI scenarios.)

2.11.2 Modes of Access

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*

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

Officer Services As documented in the previous validation, the following actions will *zeroize* all the PPD items:

- device hardware detecting a tamper event
- Miniboot detecting a hardware or software malfunction rendering Layer 3 unsafe to execute.
- A code-loading officer doing an initial installation of the Layer 3 program.
- A code-loading officer undertaking a change to the execution environment of Layer 3 for which the the Layer 3 officer did not indicate trust.

Internal User Services Table 2.3 presents how the internal user services access SRDI.

External User Services Table 2.4 presents how the external user services access SRDI.

Service	Role				
	Officer 0	Officer 1	Officer 2	Officer 3	External User
Using the device: Query: SKA Cert Query: Status Query: Signed Health Query: Certlist Continue to MB1 Continue to CP/Q++	YES				
Factory Manufacturing: IBM Burn IBM Initialize	YES	(No access to pre-initialized device)			
Factory Repair: Revive Emergency Burn 1 Field Certify Refresh SKA	YES				
Re-Certify		YES			
Update Miniboot 1: Ordinary Burn 1		YES			
Install CP/Q++: Establish Officer 2 Emergency Burn 2		YES			
Update CP/Q++: Ordinary Burn 2 Surrender Officer 2			YES		
Install Program 3: Establish Officer 3 Emergency Burn 3			YES		
Update Program 3: Ordinary Burn 3 Surrender Officer 3				YES	

Table 2.2 4758/Miniboot Officer Security Policy, when configured as this delta-module

Service	SRDI Access
sccSignOn	N/A
sccGetNextHeader	N/A
sccGetBufferData	N/A
sccPutBufferData	N/A
sccEndRequest	N/A
sccDES8bytes	N/A
sccDES	N/A
sccDES3Key	N/A
sccEDE3_3DES	N/A
sccTransformCDMFKey	N/A
sccSHA1	N/A
sccRSAKeyGenerate	N/A
sccModMath	N/A
sccRSA	N/A
sccComputeBlindingValues	N/A
sccDSAKeyGenerate	N/A
sccDSA	N/A
sccGetRandomNumber	N/A
sccQueryPPDSpace	N/A
sccSavePPD	Item gets *atomically written* to specified value.
sccCreate4UpdatePPD	Item gets *allocated* and *atomically written* to zeros.
sccUpdatePPD	Item gets *non-atomically updated*
sccGetPPD	Item gets *read*
sccGetPPDDir	All items get *listed*
sccGetPPDLen	Item gets *sized*
sccDeletePPD	Item gets *deleted*
sccDeleteAllPPD	All items get *deleted*
sccGetConfig	N/A
sccSetClock	N/A
sccClearILatch	N/A
sccClearLowBatt	N/A

Table 2.3 Internal User service/SRDI access policy.

Service	SRDI Access
sccRequest	N/A
ABORT_REQUEST	N/A
reset	N/A
sccGetConfig	N/A
sccQueryAgent	N/A

Table 2.4 External User service/SRDI access policy.