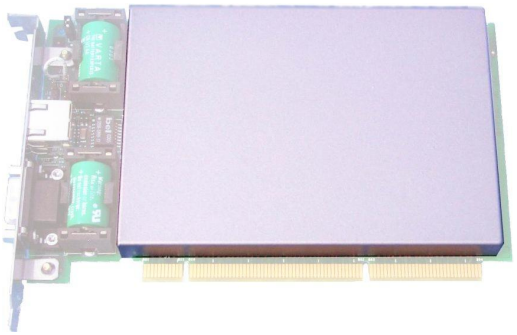
IBM eServer Cryptographic Coprocessor Security Module
Model 4764–001
Firmware (MiniBoot) version 1.16

Security Policy

IBM
Advanced Cryptographic Hardware Development

March 31, 2005



Policy information: `$Revision: 1.40 $`

1 Scope of Document

This document describes services that the IBM eServer Cryptographic Coprocessor (4764–001, "*the module*") with Miniboot software resident in ROM and flash, provides to a population of security officers, users, and the security policy governing access to those services.

The document is built on the foundations of the previously FIPS-validated IBM 4758 Model 002 (validation certificates 116), reflecting the implementation differences between the 4758 and the 4764–001.

Background of Family The module is a programmable secure coprocessor. It consists of:

- base *hardware*;
- *embedded firmware* that is not visible to the outside;
- *Miniboot* software, which controls the security and configuration of the device;
- higher *system software* and *application* layers

  Note that higher layers of software and application (Layers 2 and 3) are not included in the current validation.

The combination of the hardware and Miniboot comprise the security foundation of the module. What a particular module ends up doing is controlled by higher software layers. However, what goes into these layers, and under what circumstances their secrets are preserved or destroyed, is controlled by Miniboot layers.

The validation of this basic platform establishes that, no matter what is loaded into Layer 2 and Layer 3, this platform is secure.

- Miniboot correctly configures and identifies what's in these layers, in accordance with our policy, each time Miniboot runs.
- If an entity uses an outbound (Layer 1) private key which Miniboot validated to belong to an untampered card in a specified application configuration, then either:
  - that entity is that application configuration on that untampered card,
  - or that application configuration on that card gave away its key.

The original Model 1 was introduced in August 1997. In November 1998, the foundational hardware and software received the world's first FIPS 140-1 Level 4 validation (validation certificate 35). Subsequently, the Model 13 was introduced, as a variant of the same "Model 1" device; the Model 13 received a FIPS 140-1 Level 3 validation (validation certificate 81).

In 2000, IBM introduced two additional members of this family: the Model 2, and the Model 23. These devices consist of the follow-on "Model 2" device, with differing levels of physical security. Table 1 summarizes these variations; base firmware has been validated at Level 4 and 3, respectively (validation certificates 116 and 117).

The IBM eServer Cryptographic Coprocessor, introduced in 2003, is functionally very similar to the Model 2 4758 with enhanced infrastructure capabilities (in terms of performance, enhanced capabilities of its PCI-X interface, and RAS features). In other terms, the 4764–001 firmware is functionally equivalent to a Model 2 4758.

This Security Policy corresponds to the functionality of Segments 0 and 1 of the follow-on coprocessor of the IBM 4758/4764 family, the IBM eServer Cryptographic Coprocessor (specifically model 4764–001, with Level 4 overall security).

Follow-on Hardware The module hardware offers significant performance improvements over the 4758 hardware:

- The module CPU is now a 266 MHz PowerPC.

Model	*Base Hardware*	*Software*	*Physical Sec.*	*Overall*
4758 Model 1	Original PCI	Original Miniboot	Level 4	Level 4
4758 Model 13	Original PCI	Original Miniboot	Level 3	Level 3
4758 Model 2	Follow-on, PCI	Follow-on Miniboot	Level 4	Level 4
4758 Model 23	Follow-on, PCI	Follow-on Miniboot	Level 3	Level 3
4764–001	Enhanced, PCI-X	Follow-on Miniboot compatible	Level 4	Level 4

Table 1 Summary of 4758/4764 product family

- Internal battery-backed SRAM offers 64 KB ECC-protected storage (with 63 KB usable)
- Hardware support of the *Advanced Encryption Standard* (AES) algorithm (128 to 256 bit key sizes)
- Vastly improved modular math engine (enhancements to performance, features, and reliability)
- Improved TDES engine (single-DES is still supported for legacy applications)
- Hardware support for SHA-1 and MD5 hashing

Outbound Authentication Miniboot completes the original security architecture by including full *outbound authentication* support, enabling applications *at runtime* to authenticate themselves, their software configuration and the fact they are executing on untampered hardware.

With outbound authentication, secure coprocessor applications become first-class cryptographic entities empowered to participate in a full range of protocols: from signing messages, to receiving encrypted messages, to exchanging keys with programs running on the other side of the Internet.

4764 This document describes the policy for module with Level 4 physical security.

The terms "*4764*" and "*4764–001*" are used interchangeably in this document, unless otherwise noted. Differences between 4764 models are specifically highlighted, where applicable.

> Note that the 4764–001 designation, in IBM terminology, is a *machine type* and *model number*, together generally referred to as model. The same module may be assigned different *feature codes* in specific configurations, especially if embedded in another subsystem (such as I/O boards in zSeries mainframes). Feature codes containing the same card configuration may also be different in different server platforms.
>
> Independent of the actual feature code in the end configuration, the machine type of the secure module, 4764–001, does not change.
>
> As described in this document, the module does not need to trust its PCI-X host, therefore it is prudent and reasonable to use the card-specific 4764–001 machine type to identify the module. In certain cases, references may still be made to the behavior or PCI-X properties of the host system, irrespective of the actual platform.

2 Applicable Documents

- the FIPS 140-2 standard, the *Derived Test Requirements*, and on-line implementation guidelines
- DES: FIPS PUB 46, FIPS PUB 74, and FIPS PUB 81
- AES: FIPS PUB 197
- SHA-1: FIPS PUB 180-2
- DSS: FIPS PUB 186-2
- MD5: RFC 1321, “The MD5 Message-Digest Algorithm”
- Pseudorandom Number Generation: Appendix 3 of FIPS PUB 186-2.
- *Digital Signature Scheme Giving Message Recovery*: ISO/IEC 9796
- the TDES standard, ANSI X9.52, *Triple Data Encryption Algorithm Modes Of Operation*
- the ANSI X9.31 standard (referenced in the context of RSA signatures and key generation)

This document is based on the security policy of the IBM 4758–002:
IBM 4758 Model 2 Security Policy,
`http://csrc.nist.gov/cryptval/140-1/140sp/140sp116.pdf` (accessed 2004.09.02).

3 Secure Coprocessor Overview

A multi-chip embedded product, the module is intended to be a high-end *secure coprocessor:* a device—with a general-purpose computation environment and high-performance crypto support—that executes software and retains secrets, despite most foreseeable physical or logical attack. Customers can use this secure platform as a foundation for their own secure applications, which may range from crypto APIs to digital media distribution.

Miniboot The foundational Miniboot code helps achieve this security goal by permitting software (including updates to Miniboot itself)

- to load and execute safely,
- while allowing participants to authenticate that they are interacting with a specific untampered device in a specific software configuration.

Authenticating the Configuration Verifying that one is interacting with an untampered device operating the correct software is necessary for both classes of applications:

- **Standalone devices, such as cryptographic accelerators.** Research results show that if a user cannot verify that their crypto box is *both* untampered, and operating the intended software, then their entire cryptographic operation is threatened. For example, the Young and Yung attack shows how an adversary can replace the key generation algorithm with one that appears to behave completely correctly and "randomly"—except the adversary can learn all the keys.
- **Distributed applications.** Many e-commerce scenarios require that one party be able to trust computation that occurs at a remote site, which is under the physical control of a party who may benefit from tampering with this computation. See Fig. 1.

The module provides full outbound authentication for all layers of software. OA features are integral to Segments 1 and 2; Segment 3 entities (applications) may access OA services through an exposed Segment 2 interface.

Maximum Flexibility, Minimal Trust We provide these security properties while also accommodating these constraints:

- no trusted couriers or on-site security officers are needed
- IBM maintains no database of device secrets
- IBM does need to see application software
- rewriteable software can fail, or behave with malice, without compromising the integrity of lower layers
- IBM (or other software developers) have no "backdoor access" to customer's on-card secrets

Secure Platform Our goal is to produce a secure platform on which developers (including IBM) can build secure applications.

Our module, for validation, consists of the IBM eServer Cryptographic Coprocessor hardware, along with the foundational Miniboot software.

By obtaining FIPS validation for our *hardware* and *bootstrap/configuration control software* (Layer 0 and Layer 1, in Fig. 3), we make it easy for developers to build and deploy secure, FIPS-validatable applications—since they simply have

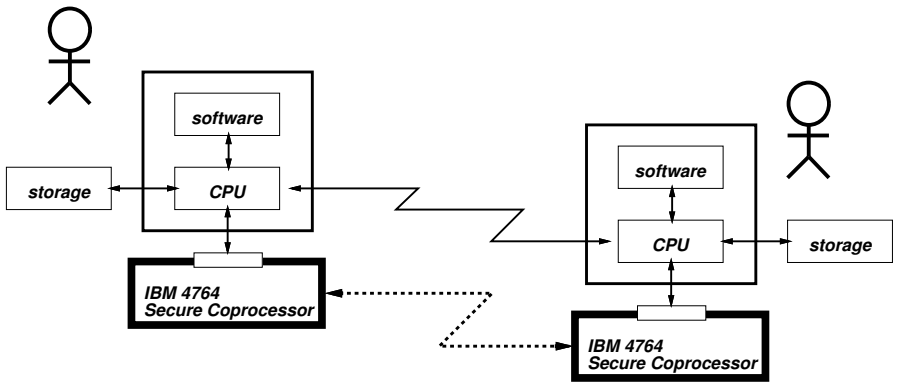


Figure 1 Our goal is to enable users, who have never met, to buy our hardware, download software from their chosen security officers, then interact securely—each able to verify that they are talking to the *real thing*, doing the *right thing*.

to prepare validation documentation for their additional software, and have it evaluated for secure operation with this module.

Validating this platform at Level 4 customers the flexibility to design to any FIPS level of any code built on top of this module.

More Information For more details on the security architecture of the IBM 4758/4764 family of devices, see:

- S. W. Smith, S. H. Weingart. "Building a High-Performance,Programmable Secure Coprocessor." *Computer Networks, Special Issue on Network Security.* 31: 831-860. April 1999.

(The document is available in softcopy on the IBM Security web site.)

3.1 Architecture and Resources

The module incorporates state-of-the-art hardware security and cryptography technology, including:

- modular exponentiation hardware
- AES hardware
- TDES hardware, optionally usable to perform DES for compatibility with legacy applications
- SHA-1 hardware
- MD5 hardware
- protective,tamper-respondent matrix
- tamper detection and response circuitry
- hardware-noise random number generation; software postprocessing complies with FIPS 186-2, Appendix 3.3

See Fig. 2.

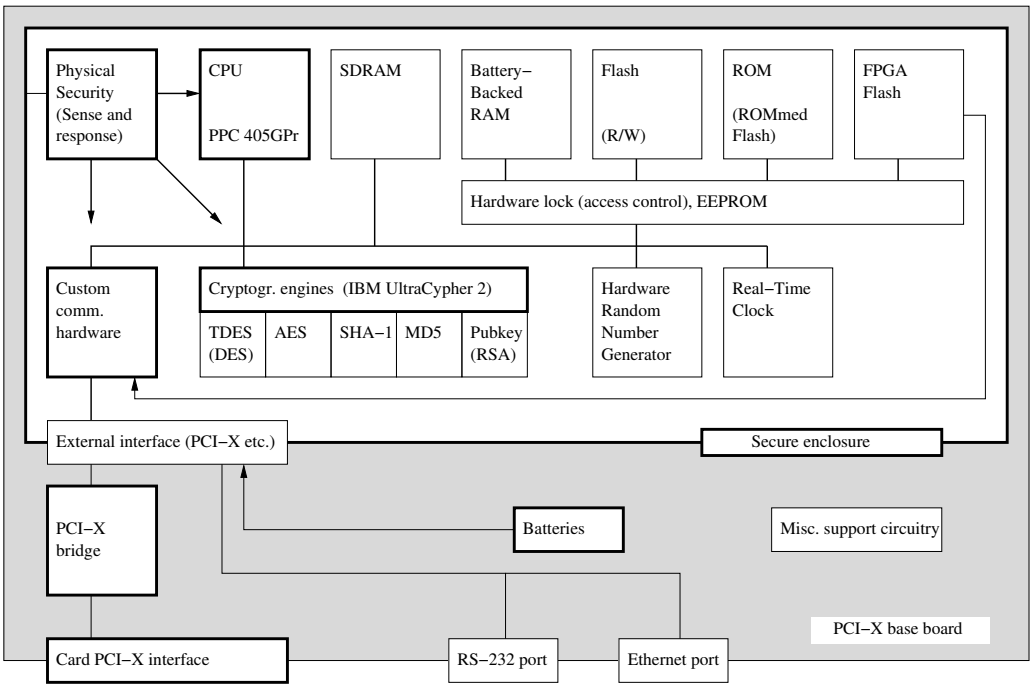


Figure 2 Module hardware architecture.

Physical Security Our device is Level 4-tamper-protected for life, from the moment it leaves the factory vault. When the internal tamper circuitry that is *always* active detects physical penetrations, it near-instantly *zeroizes* internal secrets by explicitly "crowbarring" memory devices (BBRAM), shorting positive supply voltage with ground. In addition to cutting power and discharging, special-purpose wiping code purges data buffers in the communications FPGA, removing memory contents actively at hardware speeds.

Non-zeroized memory devices either discharge and lose contents in a few milliseconds if module power is removed (SDRAM) or lose their encryption key when BBRAM is zeroized (flash).

The protection circuitry also detects and responds to other environmental attacks, including temperature, voltage, and radiation.

The module monitors removal from its PCI-X slot. If the module is ever removed from the PCI-X slot hosting it, the *external warning* (*"intrusion latch"* tamper bit) is set to indicate the removal. This bit (event) does not cause zeroization, but software may choose to respond to it and zeroize module secrets, if necessary.

Table 7 summarizes the effects of tamper types and the recommended application actions.

The module has a dedicated jumper wire to destroy secrets if a security-conscious user does not wish secrets to leave the site when the module is serviced or replaced. The wire is externally available, and may be severed at all times.

Software Architecture The internal software architecture is divided into four layers.

The foundational two layers—submitted for this validation—control the security and configuration of the device. These layers come shipped with the device.

- Layer 0: Permanent POST0 (Power-on Self Test) and Miniboot 0 (security bootstrap).
- Layer 1: Rewritable POST1 and Miniboot 1

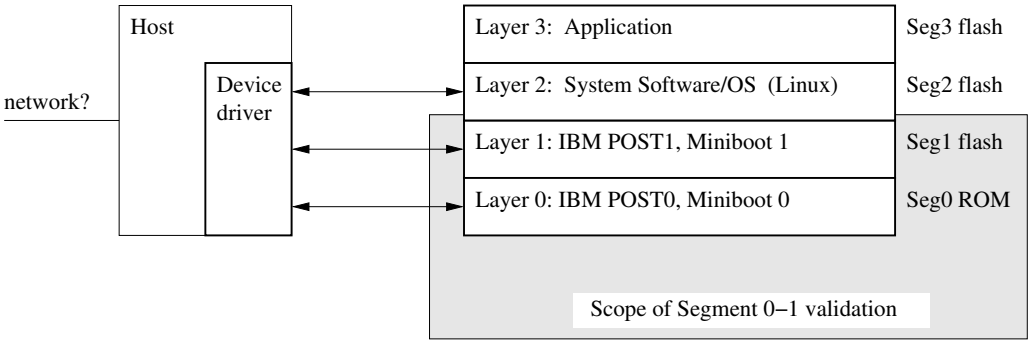


Figure 3 Module software architecture.

POST routines perform initial and higher-level testing of card infrastructure. If both POST passes are successful, card hardware is guaranteed to be functional for basic services. In addition to POST, both Miniboot 0 and 1 perform detailed, targeted tests of card hardware (cryptographic, transport, and other infrastructure) before relying on their services.

The upper two layers customize the operation of each individual device. *Note that the following layers are not included in the current FIPS validation.*

- Layer 2: System Software. Supervisor-level code.
- Layer 3: Application code.

These two layers are added in the field. The foundational Miniboot software ensures that installation, maintenance, and update of these layers can proceed safely in a hostile environment.

See Fig. 3.

Memory The internal non-volatile memory components consist of *flash*, battery-backed static RAM (*BBRAM*), and *EEPROM.* The memory resources are organized according to this layer structure.

- *flash* is organized into four *segments*, one for each layer. Each segment contains the program for that layer. Layer 0 is boot-block ROM. Layer 1 has two copies, to provide full *atomicity*[1] for Miniboot 1 updates.
- *BBRAM* is organized in to four sections, one for each layer. Each section contains the secrets for that layer.
- *EEPROM* contains some special status fields.

Hardware Locks Write-access to flash, read/write access to *BBRAM*, and read/write access to the *EEPROM* are guarded by the separate *Hardware Lock Microcontroller (HLM).*

- The HLM makes many access control decisions based on the value of its internal *ratchet.* Hardware reset clears this value to zero; the HLM will advance the ratchet when requested by the main CPU, but the only way to decrease the current value is a hardware reset—which also forces the CPU to begin executing from known ROM in known state.
- The HLM also implements, in internal EEPROM, the *factory sticky bit* ("module has been initialized" bit). Once this bit is turned off (indicating the device is about to venture from the secure factory into the world), the HLM will never let it be turned on again.

[1] By "atomicity," we mean that a change happens *entirely*, or *not at all*—despite failures and interruptions. Since Miniboot 1 supports in-field firmware repairs, it's critical that a working copy of Miniboot 1 itself always be present. Our approach eliminates the window of vulnerability created by the underlying *flash* memory technology, which requires first erasing a region, then rewriting it.

(The hardware lock is a critical part of ensuring that the Miniboot security software works despite potentially arbitrary software in Layers 2 and 3.)

3.2 Included Algorithms

The module includes these NIST-approved algorithms:

- AES
- TDES
- DES (for compatibility with legacy applications)
- DSS
- SHA-1 (on byte-granular input)
- software DRNG, compliant with FIPS PUB 186-2, Appendix 3.3[2]

Symmetric algorithms support every combination of encryption/decryption and ECB/CBC modes for all possible key sizes (56 bits for DES; 112 or 168 bits for TDES; 128, 192, or 256 bits for AES). Each available algorithm is extensively tested during startup and during operations (see Table 3).

The module also includes these *non-approved* algorithms:

- MD5 (on byte-granular input)
- RSA (for signing/signature verification)
- ISO9796 padding for public-key signatures
- hardware random number generation

[2] limited to with fixed-size seeding

Security Requirements Section	Level
Cryptographic Module Specification	4
Module Ports and Interfaces	4
Roles, Services, and Authentication	4
Finite State Model	4
Physical Security	4
Software Security	4
Operational Environment	N/A
Cryptographic Key Management	4
EMI/EMC	4
Self-Tests	4
Design Assurance	4
Mitigation of Other Attacks	N/A

Table 2 Module Security Level Specification.

4 Cryptographic Module Security Level

This module is intended to be Level 4. See Table 2.

5 Ports and Interfaces

The module communicates with its host through a PCI-X bridge chip, hosted on a PCI-X main board. Three flexcable connectors connect the secure module to the PCI-X board; these connectors carry the following signals:

- PCI-X bus data/addresses (the module is a PCI-X master)
- PCI-X control signals
- Power, 3.3 V from PCI-X bus
- Power, from batteries mounted on the PCI-X main board
- RS-232 signals
- Ethernet connector signals
- External warning control

  A physical security feature, the external warning bit alerts the module if it is removed from its PCI-X host (page 28).

In the configuration submitted for FIPS validation, the RS-232 port is used for status output during self-tests. It does not serve as an input.

Apart from self-tests, Segment 0 and 1 code does not drive or use the Ethernet connector, which is mounted on the PCI-X main board.

6 Self-tests

The 4764 executes the following self-tests upon every startup:

Configuration integrity test verifies firmware flash memory modules and code integrity. The initial and continuous checks are basically identical, verifying memory checksums when required. Initial checks simply verify integrity once before data is used for the first time.

Non-modifiable PowerPC code, POST0 and Miniboot0, are checked for integrity through embedded checksums. In case of checksum mismatch, the code halts itself (POST0) or is not even permitted to execute (Miniboot0, inhibited by POST0). This code is executed only at startup.

Flash failures are detected and corrected where possible in Segment 1, reverting to the unaffected image if possible (two copies of Segment 1 code are stored in flash). Segment 2 and 3 code corruptions are detected but are not correctable, since there is only one copy of each. The same check applies to Segment 2 and 3 secrets. Checksums are checked upon each write operation on a continuous basis.

OS and application segments may implement error checking and recovery for their own persistent data in flash. The IBM Segment 2 image implements such a flash filesystem, permitting graceful degradation in case of a flash failure (consistent with well-known failure patterns of flash memory). Such checks are outside the scope of the Segment 0–1 FIPS validation, but are mentioned here for completeness.

Functional integrity of hardware components is tested through a selected set of known answer tests, covering all programmable components. The programmable devices verify their own code integrity, external tests verify proper connectivity.

CPU integrity is verified as part of POST0, before execution continues to Miniboot. These checks verify fundamental functionality, such as proper execution control, load/store operations, register functions, integrity of basic logical and arithmetic operations, etc.

Once the CPU tests pass, CPU failures are monitored using other error-checking mechanisms (such as parity checks of the PCI bus etc.)

FPGA integrity (communications firmware) is checked by the FPGA itself, through a checksum embedded in the image, upon loading. If the test fails, the FPGA does not activate, and the card remains inaccessible.

After initialization, FPGA interfaces and internals are covered through parity checks internally, and external end-to-end checks at higher logical levels.

Crypto ASIC integrity is verified by comprehensive known-answer tests at startup, covering all possible control modes. These tests implicitly cover FPGA transport as well, since tests are performed using both available internal interfaces.

During regular operations, the crypto ASIC covers all traffic through combinations of redundant implementations, CRCs, and parity checks, in a way specific to each crypto engine. Any failure is indicated as a hardware failure, to the module CPU and the host.

Tamper code integrity is checked through a checksum embedded in the executable image. This initial test is performed on the internally stored code executed by the controller, and only failure is explicitly indicated.

HLM controller integrity is verified through its embedded checksum.

Modular math engine self-tests cover all possible control modes, and different sizes of modular arithmetic. The modular math primitives' testing covers only modular arithmetic, up to full exponentiation, but not algorithm-level (i.e., RSA or DSA protocols).

A separate, fully specified known-answer test (KAT) is performed on the DSA implementation, including a KAT through a predefined "random sequence". The RSA implementation is tested through dedicated KATs, in addition to tests of underlying primitives.

Both DSA and RSA tests are performed continuously on operations, checking answer consistency through performing the operation in the reverse direction, in addition to initial KATs.

Symmetric crypto engines are tested by KATs. All algorithms are subject to KATs in all available modes of operation, and key sizes, both encryption and decryption. Hash functions are covered by several KATs (Table 3).

Deterministic random number generator (postprocessing software) is covered by a KAT: seeded with a known value, the "random" output of the generator is compared against the expected value, halting the module in case of a mismatch. As described before, the DRNG is compliant with FIPS PUB 186-2, Appendix 3.3., and shares code with the DSA implementation.

In addition to end-to-end DRNG software coverage, POST sanity checks verify that the hardware source is functional (through statistical tests).

Interactive communications tests verify that the card PCI-X bus is functioning properly.

As part of automatic self-tests, *critical functions tests* cover the module CPU cache control logic (data and instruction), processor registers, and instruction set; PCI-X bus transport integrity (including communication mailboxes), and RAM module integrity.

Apart from interactive communication tests, self-tests run without further user intervention, if code execution is advanced to Segment 1. Non-interactive Segment 0 tests execute before PCI-X communications are tested.

In addition to startup tests, the module executes conditional data tests on in the following modules:

Pairwise consistency test on RSA and DSA operations

Continuous integrity checks on modular math arithmetic (including RSA and DSA operations), implemented in hardware

Cross-checks between redundant, independent implementations of TDES and DES algorithms.

Bi-directional consistency checks on AES encryption and decryption (results are ran through the reverse operation, verifying that the original input is restored properly).

Parity checks on all other operations performed in the symmetric crypto engine (including hashing), including partially redundant data flow and control logic.

Continuous test on the utilized random number generator, both hardware and software.

Algorithm	*Key size (bits)*	*Mode*	*Operation*
Symmetric algorithms			
AES	128	ECB	encryption, decryption
		CBC	encryption, decryption
	192	ECB	encryption, decryption
		CBC	encryption, decryption
	256	ECB	encryption, decryption
		CBC	encryption, decryption
TDES	112 (128)	ECB	encryption, decryption
		CBC	encryption, decryption
	168 (192)	ECB	encryption, decryption
		CBC	encryption, decryption
DES	56 (64)	ECB	encryption, decryption
		CBC	encryption, decryption
Public-key algorithms			
DSA	1024	N/A	signing, signature verification
RSA	512	N/A	signing, signature verification
Modular math	1024	N/A	(comprehensive test of primitives, up to exponentiation)
	2048	N/A	(comprehensive test of primitives, up to exponentiation)
Hash algorithms			
SHA-1	N/A	N/A	(hashing)
MD5	N/A	N/A	(hashing)
Random-number generator			
FIPS 186–2, App. 3.3	N/A	N/A	(generate known “random stream” from a fixed seed)

Table 3 Algorithm known-answer tests

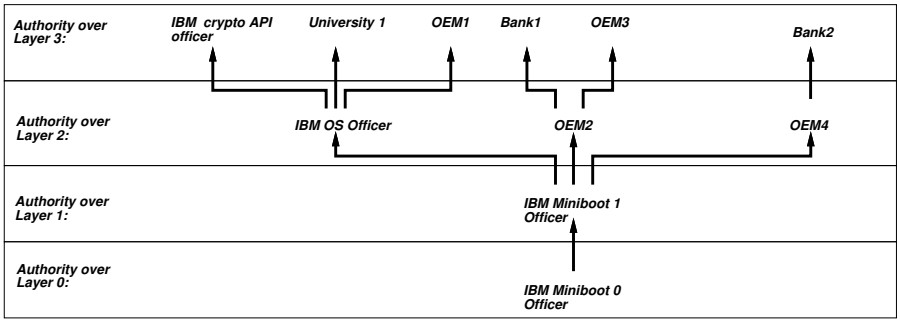


Figure 4 Although each device has at most one officer in charge of each layer. The space of *all* officers over *all* devices is organized into a tree. This diagram shows an example hierarchy.

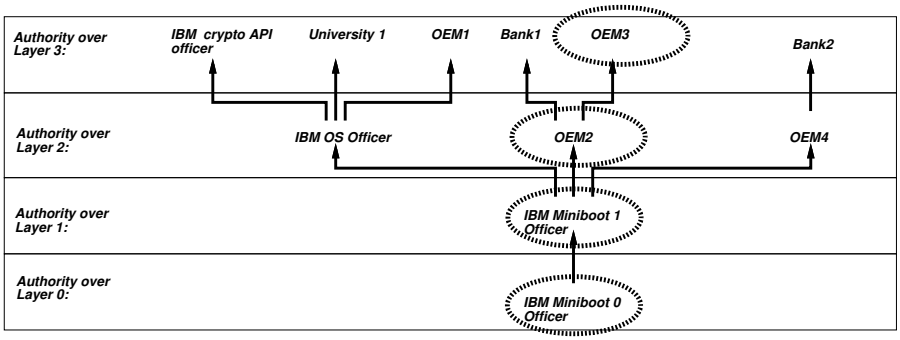


Figure 5 Within this example owner hierarchy, one family of devices might have a Layer 2 controlled by "OEM2" and a Layer 3 controlled by "OEM 3."

7 Roles and Services

7.1 Roles

Our module has roles for *Officer 0, Officer 1, Officer 2, Officer 3* and a generic *user*.

Each layer in each card either has an external officer who is in charge of it ("owns" it)—or is "unowned."

This entity does not have to be co-located with the card—in fact, it usually isn't. (Further, any one officer may be in charge of layers in many cards.)

We enforce a tree structure on officers:

- All cards will have IBM_Officer_0 as their *Officer 0.*
- All cards will have IBM_Officer_1 as their *Officer 1.*
- If layer n is unowned in a card, then no layer $m > n$ can be owned.
- One owns exactly one layer n (but perhaps in many cards); one's parent owner $(n-1)$ must be the same in all such cards.

Fig. 4 through Fig. 6 sketch examples of this structure.

A card's *Officer 2* is identified by a two-byte OwnerID chosen by its *Officer 1.* A card's *Officer 3* is identified (among all other officers sharing the same *Officer 2* parent) by a two-byte OwnerID chosen by its *Officer 2.* (Both OwnerIDs together identify an *Officer 3* among *all Officer 3*s.)

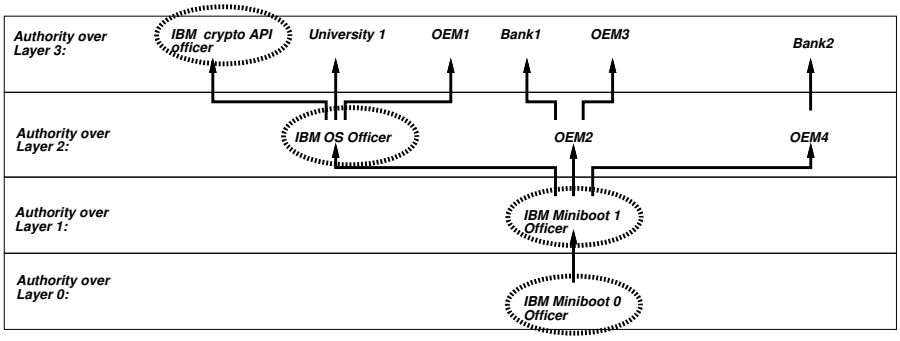


Figure 6 Within this example owner hierarchy, another family of devices might have the IBM OS/Control Program in Layer 2 and the IBM crypto API in Layer 3.

We additionally have a notion of *User*: someone who happens to be communicating with the card wherever it is installed. (See also Section 9).

Specific application programs may define other classes of principals.

Table 4 summarizes what commands are allowed for what roles.

Each role must authenticate separately for each service request, as part of that request. Per our design goals, Officer n (for $n > 0$) can do this remotely.

Fig. 7 illustrates how the commands change initialization of the device; Fig. 8 illustrates how the command change the configuration of Segment 2 and Segment 3.

7.2 Operations

Our module provides the following types of services:

- Miniboot *queries*
- Miniboot *commands*

Miniboot queries and commands must be presented to the module from its host, when the appropriate half of Miniboot is executing.

As the name implies, Miniboot runs at boot time. Hardware reset forces the module CPU to begin executing from a fixed address in Segment 0, which contains POST0 and Miniboot 0 (MB0). If POST0 fails, the device halts. If POST0 is successful, then Miniboot 0 executes. It listens and responds to zero or more queries, followed by exactly one command.

If the command is a *Continue* and Segment 1 is deemed safe, execution proceeds to Segment 1, which contains POST1 and Miniboot 1 (MB1). If POST1 fails, the device halts. If POST1 is successful, then Miniboot 1 executes. It listens and responds to zero or more queries, followed by exactly one command. If the command is a *Continue* and Segment 2 is deemed safe, execution proceeds to Segment 2.

Halt In many situations, Miniboot will halt, by sending out an explanatory code, and entering a halt/spin state. In particular, Miniboot will halt upon:

- rejection of any command
- successful completion of any command other than “Continue”
- detection of any error (self-test or command functional error)

		Services	Roles				
			Officer 0 (IBM)	Officer 1 (IBM)	Officer 2	Officer 3	User
Queries		Query Status	No access restrictions				
		Query Signed Health					
Commands	Run	Continue to Segment 1	No access restrictions				
		Continue to Segment 2					
		(IBM Initialize)	Perform without restrictions, exactly once, in factory				
	Officers	Establish Officer 2		yes			
		Establish Officer 3			yes		
		Surrender Officer 2			yes		
		Surrender Officer 3				yes	
	Code Management	IBM Burn (Segment 1)	Perform without restrictions, while still in factory				
		Ordinary Burn (Segment) 1		yes			
		Ordinary Burn (Segment) 2			yes		
		Emergency Burn (Segment) 2		yes			
		Ordinary Burn (Segment) 3				yes	
		Emergency Burn (Segment) 3			yes		

Table 4 Miniboot command/query policy.

- detection of any other condition requiring alteration of configuration

This was a design decision: always halting makes it easier to be sure that precondition checks and clean-up are applied in a known order. POST (infrastructure) failures are treated similarly to Miniboot, halting the module after outputting a failure status.

Reset To resume operation, the user must cause another hardware reset. On a hardware level, the device can be reset by:

- power-cycling the device
- triggering the designated control bit in the Bus Master Control/Status Register accessible from the PCI-X host (it forces a module reset through the external PCI-X bridge chip).

On a software level, the IBM-supplied host-side device drivers will transparently reset the device (via the "Add-on Reset" signal) when appropriate:

- When the user "closes" the device after opening it for Miniboot
- When the user "opens" the device for Miniboot, but the device driver detects the device is halted.
- When the user opens the device for ordinary operation, but the host-side driver determines that the device is not already open. (In this case, the IBM-supplied host-side device drivers will transparently reset the device and also execute MB0 Continue and MB1 Continue, to try to advance to the Program 2 code.)

Receipts Upon successful public-key commands, Miniboot 1 provides a signed receipt (to prove to a remote officer that the command actually took place, on an untampered card). Miniboot 1 also signs its query responses.

7.3 Inbound Authentication

Miniboot authenticates each command request individually.

For $N \geq 1$, Miniboot authenticates a command from Officer N by verifying that the *public-key signature* on the command came from the entity that is Officer N for that card, and was acting in that capacity when the signature was produced. This approach enables the officers to be located somewhere other than the devices they control.

In a module configured in FIPS mode, signatures are made with 1024-bit DSA keys. *Forging 1024-bit DSA signatures on segment contents is assumed to be infeasible (NIST, "Digital Signature Standard (DSS)", FIPS 186–2).*

Miniboot authenticates Officer 0 commands (used for emergency repairs when the device is returned to the IBM factory vault) using secret-key authentication based on TDES keys. Use of any of these commands destroys any other officer secrets that may remain in the device. (Note that these commands are not available outside the secure manufacturing facilities, but are sometimes mentioned for completeness.)

The module has a dedicated jumper wire to destroy secrets if a security-conscious user does not wish secrets to leave the site when the module is serviced or repaired. Removing the jumper wire disconnects the battery path and zeroizes the module through a hard voltage tamper (p. 28).

7.4 Outbound Authentication

At the last stage of manufacturing, Miniboot on a card generates its first keypair. IBM certifies the public key to belong to that untampered card with that version of Miniboot. This certificate attests that the entity which knows the private key

matching that public key is that untampered card, with that Miniboot software. The certification takes place in the secure manufacturing vault.

Each time Miniboot 1 replaces itself, it generates a keypair for its successor and certifies the new public key with its current private key. This certificate establishes that if one trusted the current installation of Miniboot, then one can trust the identity of the next one.

Each time the application configuration changes, Miniboot 1 also generates and certifies a keypair for Layer 2. (Miniboot also zeroizes the old Layer 2 private key, if one exists.) This certification binds that keypair to that application configuration on that card. (Our intention is that Layer 2 will in turn use this keypair to provide outbound authentication services to the application.)

This binding, coupled with the trust chain for Miniboot's own keypair, permits parties to make accurate trust judgments about the entity wielding a private key certified this way.

7.5 CSP

The value of a secure coprocessor lies in its ability to be a trusted platform: "the real thing,doing the right thing."

Since it is Miniboot and the hardware—the module, as submitted for this validation—that provides this property, the CSP for Miniboot consist of the various authentication and configuration elements. These fall into two groups.

For Layer 0, which is incapable of asymmetric cryptography, secrets (and CSPs) are related to symmetric keys. Three "keys" are used to authenticate in Segment 0 operations: "P0" and "P1" are symmetric (TDES) keys used to authenticate Miniboot 0 and Miniboot 1. "NZ" is a non-zeroizable key (secret) that could be used during card recovery (if that is ever performed, which is not the case as of this writing). These symmetric keys are used for authentication. Use of these keys is only possible in secure manufacturing facilities, so they are mentioned here only as reference. *All Segment 0 secrets are internally generated, not exported or imported through the secure boundary, and are not accessed by field-applicable services.*

The Layer 0 state includes the TDES secrets used for secret-key authentication of Officer 0. (Note, however, that if Officer 0 uses these secrets to potentially affect the configuration of a higher layer, that layer's secrets are atomically destroyed. Such factory actions, again, are outside the scope of this document.)

For Layer 1 through Layer 3, the CSP consists of:

- the identity of the officer over this layer
- the state that this program has accumulated in non-volatile BBRAM
- miscellaneous status fields
- Segment 1 has a device private key pair, generated during card initialization.
- Segment 2 code may offer public-key services similar to those of Segment 1. If such code is present, Segment 2 may contain public-key pairs similar to the device key pair (see below)

For each of these segments, segment state is uniquely determined by the set of the above parameters. Officer public keys are imported to the module. Status fields and segment state are managed by Miniboot, some of them are returned in the Segment 1 Query.

Officer identity is represented through a public key stored on behalf of the respective officer, as part of CCPs (card configuration parameters). An officer must be able to demonstrate possession of the corresponding private key by signing commands. Officer identities may be queried through the Segment 1 status query, which returns the public key of the officer owning that segment. Officer identities are imported in the "Establish Owner" command of the corresponding segment.

Layer 1 state/CSPs also include the *device private key* that provides the foundation of that untampered device's outbound authentication ability. This key is also referred to as the *Segment 1* or *Layer 1 private key* in support documentation. The Layer 1 device key signs Segment 1 queries.

If present, Segment 2/3 state includes Layer 2 private key(s) noted above. These keys share the properties of Segment 1 keypairs: they are internally generated, are not exportable, but may be queried since they have exportable public key certificates. These keys may be generated and used through an internal OA interface that mimics the public-key services of Miniboot 1, if the Segment 2 code supports such interfaces. The default IBM code provides these interfaces; it supports *key generation*, *certificate export*, *signing*, and *verification* for a suitable Segment 3 application. These Segment 2/3 capabilities are outside the scope of this FIPS validation, but are mentioned here for completeness, since they are derived from Segment 1 key objects.

The contents of code are also critical to module security. Segment contents are updated in sync with state and officer identity. Table 5 summarizes administrator-level actions performed by Segment 0 and 1.

Segment 0 information is only indirectly accessed by field-accessible services. Segment 1–3 secrets, unless described otherwise in the table, correspond to segment ownership (i.e., to officer public keys). Actions in the table describe whether a given item is *created (C)*, *read (R)*, *written (W)*, or *destroyed (D)*. Standalone writes imply cooperation of the segment owner; *write and destroy (W/D)* operations override the segment owner through of actions of the underlying segments' owner.

Notes applicable to the table are the following:

1. The (Segment 3) *User* is entirely controlled by the Segment 3 officer, and actions of Officer 3 apply to the User as well. The User is incapable of performing Segment 0–1-level actions herself.
2. Certain Segment 0 actions, which are never performed in the field or Officers 1 to 3, are included in the table for completeness.
3. These services affect the corresponding officer's *identity*, in the form of their *public keys*, and the give segments' state.
4. These services affect the corresponding segment's *content*.
5. If Segment 3's secrets (persistent objects) are labeled to distrust Segment 1–2 configuration changes, these changes imply a Surrender Segment 3 operation as well.
6. Segments 2, 3, and the User do not exist yet when this factory command is executed.
7. Factory initialization of Segment 1 data includes generation of the device (Layer 1) public key, including its certificate from the factory CA.
8. Segment 1 Queries are always signed by the Segment 1 device key.
9. Segment 2 public-key items, which depend on Segment 2 support, are outside the scope of this validation and are not represented in the table.
10. All secrets are destroyed by a tamper response event, which does not have a corresponding Miniboot command.

The available functions affect the following CSPs and CCPs:

Query Status Read status, including layer owner identities and card infrastructure configuration

Query Signed Health Read status, including owner identities and public keys

Continue to Segment 1 read/check Segment 1 code state

Continue to Segment 2 read/check Segment 2 code state

IBM Initialize Generate device (Layer 1) keypair; write new certificate; clear Layer 2 and 3 parameters and BBRAM

Establish Officer 2 Write Layer 2 owner ID

Establish Officer 3 Write Layer 3 owner ID

Service	*Role*					***CSP** (CCP) access*
	Officer				*User*	
	0	1	2	3	Note 1 (p. 19)	
Queries						
Query Status		R				(Segment 1 and card infrastructure state)
Query Signed Health		R	R	R	(R)	(returns Segment 1–3 state, including ownership; signed with Segment 1 key, Note 8)
Execution control						
Continue to Segment 1		R				(Check: does Segment 1 state permit execution?)
Continue to Segment 2			R			(Check: does Segment 2 state permit execution?)
Officers						
(IBM Initialize)	C	C	N/A	N/A	N/A	**create: Segment 0 CSPs** Note 2, Note 6 (create: device key pair)
Establish Officer 2		W				(Segment 2 owner id) Note 3
Establish Officer 3			W			(Segment 3 owner id) Note 3
Surrender Officer 2			D			(Segment 2 owner id) Note 3
Surrender Officer 3				D	(D)	(Segment 3 owner id) Note 3
Segment contents						
(IBM Burn)	R/W	C	N/A	N/A	N/A	**Read/write Segment 0 CSPs** Note 2, Note 6 (initialize: Segment 1 code)
Ordinary Burn (Segment) 1		W				(write: Segment 1 code) Note 4, Note 5
Ordinary Burn (Segment) 2			W			(write: Segment 2 code) Note 4, Note 5
Emergency Burn (Segment) 2		W/D				(write: Segment 2 code) Note 4, Note 5
Ordinary Burn (Segment) 3				W	(W)	(write: Segment 3 code) Note 4
Emergency Burn (Segment) 3			W/D			(write: Segment 3 code) Note 4

Table 5 Roles, services, and CSP access

Surrender Officer 2 Clear Layer 2 and 3 parameters and BBRAM

Surrender Officer 3 Clear Layer 3 parameters and BBRAM

IBM Burn Load Layer 1 (owner) public key; clear Layer 2 and 3 parameters and BBRAM

Ordinary Burn 1 Load Layer 1 (owner) public key; optionally clear Layer 2 and 3 parameters and BBRAM, as defined by Segment 2/3 persistent object definitions

Ordinary Burn 2 optionally clear Layer 3 parameters and BBRAM; write Segment 2 code

Emergency Burn 2 clear Layer 2 BBRAM and Layer 3 parameters; write Segment 2 code

Ordinary Burn 3 write Segment 3 code

Emergency Burn 3 write Segment 3 code; clear Layer 3 BBRAM

7.6 Queries and Commands

Table 4 summarizes the queries and commands that Miniboot offers.

Miniboot 0 Queries Miniboot 0 provides one "field" query:

- *Query: Status*. This query returns general status information about the card software versions, card identification.

Miniboot 0 Commands Miniboot 0 provides these commands:

- *IBM Burn*. Install a new Program 1 and public key for *Officer 1*, while still in the factory but after it's no longer convenient to change the flash chips.

  Note that the IBM Burn command is not accessible in the lifetime of the module once it left the factory. It is included for reference only.

- *Continue.* Continue execution to Segment 1, if possible.

In an end-user environment, Miniboot 0 can issue only "Continue" commands to advance execution to Segment 1.

Miniboot 1 Queries Miniboot 1 provides these queries:

- *Query: Get Health.* The requester selects and sends a nonce. The card returns a signed response containing general health information:
  - the same data as Miniboot 0's *Status*
  - identifying information about code and owners in reliable segments
  - the nonce (so the requester can know this response is fresh)
- *Query: Certlist.* The card returns a signed response containing the certificate chain taking the card's current public key back to the IBM Factory CA (Certificate Authority).

Miniboot 1 Commands Miniboot 1 provides these commands:

- *IBM Initialize.* While still in the factory: generate a device keypair and SKA secrets, have these certified by the Factory CA, and turn the "module is initialized" (factory "sticky bit") off forever. (This command is rejected if sticky bit is already off.)

  Note that obviously the IBM Initialize command is not accessible in the lifetime of the module once it left the factory. It is included for reference only.

- *Establish Owner* n, for $n > 1$. Give an UNOWNED layer n to someone.

  Ownership may be established only if the given segment is not claimed by any owner (is "UNOWNED" in status queries). The owners' supplied public key is registered to the segment ownership, segment state is upgraded to "OWNED BUT UNRELIABLE", indicating that the current code may not be executed, even if ownership is properly initialized. Miniboot will not execute code from such a segment.

  Once the new owner loads code to the segment, its state is upgraded to "RUNNABLE", indicating that the segment owner identity is known and the segment code has been written after verifying its signature. Once these two conditions are met, execution may pass to this segment.

- *Surrender Owner* n, for $n > 1$. Give up ownership of Layer n.

  A prerequisite of surrendering ownership is that the segment is owned (since ownership of it is required to sign the command). The segment ownership indication is removed (segment reverts to "UNOWNED"), segment contents are flagged as *not runnable*. Just as with a segment before its contents are written first, Miniboot will not pass execution to such segments.

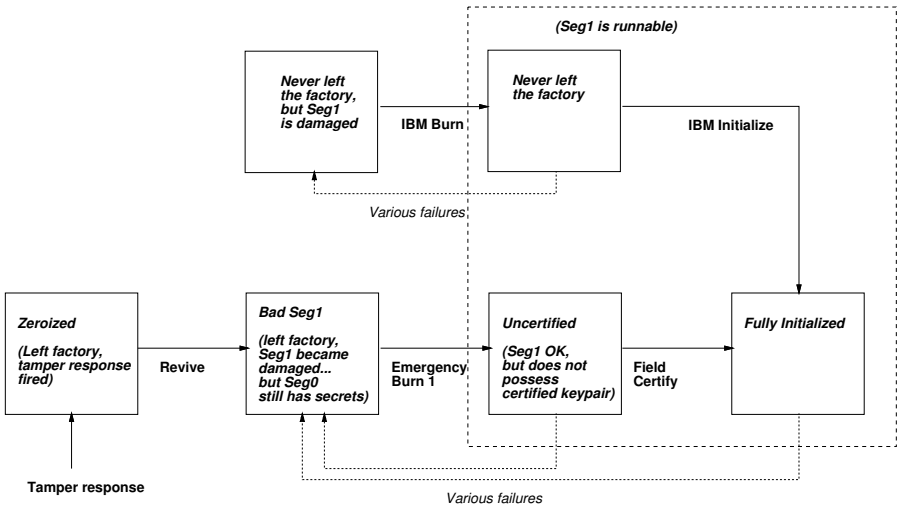


Figure 7 A sketch of the configurations and main flows for device initialization and Segment 1.

- *Ordinary Burn* n. Ordinary update of Program n and public key for *Officer n*, in an untampered card. (In preliminary documentation, this was called "Remote Burn." The older term may still persist in a few places.)
- *Emergency Burn* n for $n > 1$. Install Program n and public key for *Officer n*, in an untampered card—but without using current contents of Segment n.
- *Continue.* Continue execution to Segment 2, if possible.

7.7 Overall Security Goals

The overall goal of this policy is to ensure that the following properties hold:

- **Safe Execution.** Miniboot will not execute or pass control to code that depends on hardware that has failed.
- **Access to Secrets.** Program n should have neither read nor write access to the secrets belonging to Program $k < n$.
- **Safe Zeroization.** In case of attack or failure, the device will destroy the secrets belonging to Program n before an adversary can access the memory where those secrets are stored.

  Besides hardware tamper, such attacks may include (for $k < n$) loading of a Program k that Officer n does not trust, or fraudulent behavior by some Officer k.
- **Control of Software.** Should layer n later change in any way *other* than demotion due to failure, some *current* Officer k (for $k < n$) is responsible for that action (using his current authentication keys).
- **Outbound Authentication.** On-board applications can authenticate themselves to anyone. Suppose Alice knows a Layer 2 private key certified back, through Miniboot on an untampered card, to IBM. if Bob trusts the entities named in this certification chain, then Bob can conclude that Alice is the application entity named in that last certificate.

Module security supports the Officer and User roles as described in Table 6. The security assumption is that the identity-authenticating mechanisms are as secure as the underlying cryptographic functions. Officer 0 is represented by a TDES key. Officers 1 through 3 are identified through digital signatures. In FIPS mode, authentication mode must use the DSA (as defined in FIPS 186-2) as the digital signature algorithm.

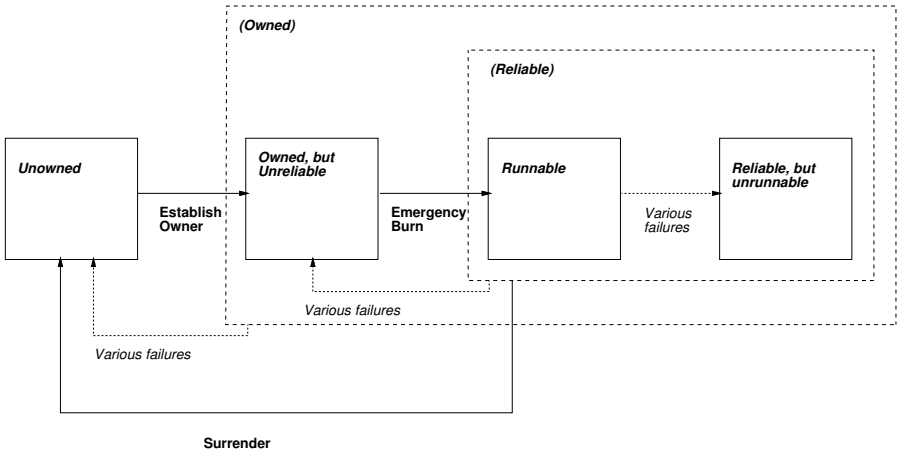


Figure 8 A sketch of the configurations and main flows for Segment n, for $n > 1$. Note that, in addition to the transitions shown, one can also "Burn" or "Emergency Burn" from any of the *reliable* states into *Runnable*.

Role	*Type of Authentication*	*Authentication Data*	*Strength of mechanism*
Officer 0	Identity	TDES key	TDES
Officer 1	Identity	Digital signature	(as of dig. signature)
Officer 2	Identity	Digital signature	(as of dig. signature)
Officer 3	Identity	Digital signature	(as of dig. signature)
User	*(Authenticated by Officer 3)*		

Table 6 Officer Roles and Authentication Mechanisms

In addition to officers, a *Generic User* role is supported by the module. The actual external user would probably interface with a Level 2 or 3 application, but that detail is outside the scope of this security policy. The User role is incapable of altering Segment 0–2 CSPs and CCPs, since it is related to applications loaded to Segment 3.

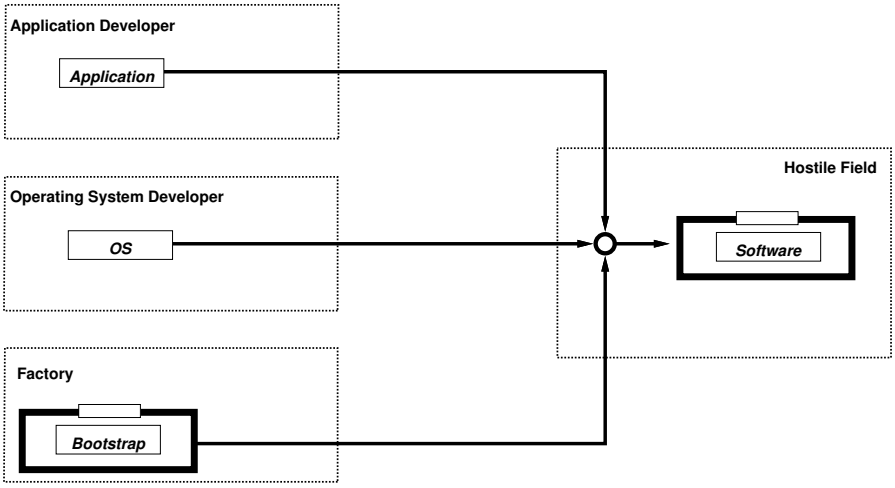


Figure 9 The module supports three layers of rewriteable software, from potentially mutually suspicious developers, configurable in a hostile field location, with neither trusted courier nor on-site security officer.

8 Security Rules

The module shall maintain the state of an officer's program only while the module continuously maintains an environment for that program that is verifiably trusted by that officer.

The module shall not require officers to trust each other (or trust the hardware manufacturer) any more than is necessary.

The module shall support public-key authentication, wherever possible.

The module shall permit officers to retain their data across uploads, where possible and reasonable.

The module shall enable all three rewriteable software layers to be installed and maintained in a hostile field, without the use of trusted couriers or on-site security officers. See Fig. 9.

9 FIPS-Related Definitions

This FIPS validation addresses *only* the hardware, and Layer 0 and Layer 1 of the software—the generic device, as shipped.

For the purposes of this FIPS 140-2 validation, Officer 0 relates to the "Cryptographic Officer 0" role. (This officer operates only in the secure factory, and most of its operations are therefore out of the scope of this policy.)

Layer 2 and 3 are excluded for this FIPS validation effort. In certain cases, Segment 2 and 3 contents are explicitly mentioned in the context of Segment 0 and 1, where absolutely necessary.

The "User" role can access a subset of Segment 3 capabilities, and can't directly influence the configuration of the module. User actions are therefore discussed in the context of Segment 3 officer actions. In most environments, external users would interact with interfaces provided by the User role, through applications loaded under the Segment 3 officer's control.

10 Module Configuration for FIPS 140-2 Compliance

10.1 Miniboot

To be a FIPS-compliant module, the device must be loaded with Version 1.16 of Miniboot 1. With production POST 1, v2.9, the validated Miniboot version resides in the card if its Segment 1 hash is `2a4e 5289 49b2 66e7 5a6e 27b9`.

The above Miniboot version must be loaded to a card with a security module with *part number 16R0911*.

The host driver should provide a method to query the Segment 1 hash; this detail is platform-dependent.

To then operate in a FIPS-compliant mode, each officer must choose DSS as the digital signature method for their public keys. Host drivers interacting with Miniboot 1 should retain module configuration and make it available to user applications on request.

For reference, the segment signature type is part of the `seg_ids` field of the Segment 1 query return structure (`mbid_t`) as documented in the host API.

10.2 Layers 2 and 3

The Miniboot software currently submitted for validation only controls the configuration of the device. Miniboot responds to queries and

- *either* responds to a configuration-changing command (then halts),
- *or* proceeds to invoke the program in Layer 2 (if it's there)—and goes away forever (until the next boot)

Because Miniboot advances the trust ratchet *before* passing control to Layer 2, the CSP[3] that Miniboot depends on (in protected flash and the Miniboot region of lockable BBRAM) cannot be compromised by Layer 2 or Layer 3.

As noted earlier, no matter what is loaded into Layer 2 and Layer 3, our validation establishes:

- that Miniboot will still run securely the next time the device is reset;
- that if an entity uses a private key which Miniboot certified to belong to an untampered card in a specified application configuration, either that entity is that application configuration on that card, or that application configuration on that card gave away its key.

In order to actually do something, the device must be loaded with Layer 2 (and, depending on the design of that program, Layer 3 as well).

Hence, to operate after bootstrap as a FIPS-compliant module, layers 2 and 3 must also be FIPS validated. The level of validation of the module in operation, as a whole, will be limited by the level of validation of these layers.

However, if both Layer 2 *and* Layer 3 are FIPS-validated, and neither permits ANY uncertified code to run in the device, then the operating system/Common Criteria requirements of FIPS 140-2 will not apply to the OS/control program residing in Layer 2.

10.3 Usage of non-Approved algorithms or modes of operation

The UltraCypher 2 ASIC used in the module provides hardware acceleration for non-Approved security algorithms (i.e., MD5 hashing), and software in Layer 2 or 3 may support other non-Approved algorithms. Even if not utilized by Segment 0 and 1 code as part of this validation, Segment 2 and 3 code mode may use MD5 facilities. The Segment 2 and

[3] a term used in the FIPS 140-2 Derived Test Requirements.

3 application is required to unambiguously indicate when it implements non-approved algorithms or modes of operation. This Segment 2/3 requirement is outside the validation requirements of Segments 0 and 1, but it is mentioned here for completeness.

As part of non-approved algorithms, the "fastpath", a host interface providing modular exponentiation support to a PCI-X host without involving the module CPU, is not enabled in FIPS-compliant mode. Segment 0 and 1 under validation does not enable fastpath facilities; the indication requirement for Segments 2 and 3 is applicable.[4]

10.4 Determining Mode of Operation

The "Signed Health Query" to Miniboot 1 will return the identities and revisions of each layer's programs, and the signature-algorithm chosen by each officer. In addition to segment identification and revision number, each program layer is identified through its contents' SHA-1 hash (see p. 26, "Miniboot", for the Segment 1 hash being validated). If Segment 1–3 contents correspond to validated configurations, and their signature algorithm is DSS, Segment 0 and 1 are in their FIPS-compliant mode.

For reference, the segment signature type is part of the `seg_ids` field of the Segment 1 query return structure (`mbid_t`) as documented in the host API. The procedure to access the Signed Health Query results is platform-dependent. On IBM server platforms, host drivers provide administrative functions to display segment configurations, which are queried during card startup.

Host drivers are assumed to store the results of the query and make them available to higher-level (application) users to find out what configuration the module is in. User applications/administrators should be able to unambiguously verify the configuration of module segments. Host driver implementation, while outside the scope of this specification, should provide a convenient way of querying card configuration, since most likely user application won't interface to Miniboot 1 directly.

[4] The fastpath status indicator bit is bit "HMM_AE" of the *HCSR* command status register, as described in the host API.

11 Module Officer/User Guidance

Primarily providing advice for security officers and users, this section also includes operational recommendations that may be useful during operating the module. These operating recommendations are relevant also to system administrators, who may not be directly involved with officer/user actions.

Since the module is shipped in an initialized state, and it may not be repaired in the field, administrator-level recommendations only cover regular operations.

11.1 Physical Security Inspection/Testing Recommendations

The module physical security mechanisms are mainly automatic, but application software (both module and host) may react differently to different tamper types, based on the requirements and assumptions of the card application. Intrusions, which may destroy card secrets through an internal, independent action, is observable both as an officer/user and system administration event.

System administrators may notice tamper detection through unusual module startup, such as a card failing to initialize. The details of such administrator-level logging are platform-dependent. It is recommended to investigate the tamper event type reported by the module, possibly cross-checking the tamper event with other logs.

The module may not be recovered after a tamper event. This includes internally stored secrets, which are destroyed after a hard tamper.

Hard tamper events are caused by very high over voltage, temperature out of reasonable operational range, radiation, or a physical tamper (penetration of the tamper-detection matrix). Reaction is instantaneous. Module memory-type devices (BBRAM, communication FIFOs) are zeroized. Module secrets, for practical purposes, are immediately destroyed. The module becomes permanently inoperative: the boot sequence does not successfully terminate without secrets in BBRAM.

Hard tamper events may only be detected after the fact by the host application. As the module is held in reset after a hard tamper, module secrets are lost. There is no further action possible on such a card, as it is held in reset by the internal circuitry, until the batteries are removed. Removing and restoring batteries does not restore functionality, as the module does not boot without its secrets.

Hard tamper events (practically, the type of tamper) are latched in PCI-X registers.

Soft tamper events are caused by moderate over voltage or temperature moderately out of operational specifications. Reaction is instantaneous. The module is held under reset while the soft tamper conditions persist, but secrets are not destroyed.

Soft tamper events may be detected after the fact by the host application. The module recovers from a reset following a soft tamper. Soft tamper events (type of tamper) are latched in PCI-X registers. has been *removed* module has been *removed* from the PCI-X slot housing it, even if no tamper condition has been detected. (Note that this is not a physical intrusion event, just a logical one.) The corresponding tamper bit is immediately set, but it may only be inspected when the module is powered up for the next time. Module secrets are not destroyed by Miniboot.

Based on the nature of the host/module application, application code may elect to zeroize module secrets if it is restarted with the external warning latch indicating previous removal from the host system.

The external warning latch state is persistent, and may be cleared through software means (by card code).

Low battery warning signals a condition where the batteries have been drained to a level that may endanger safe operations if the module is not powered externally. (A field kit is available for battery replacement.) This bit does not indicate an intrusion event.

The low battery warning is not latched; it monitors battery voltage continuously.

Physical Sec. Mechanism	*Severity/Effect*	*Recommended Frequency of Inspection/Test*	*Test Guidance*
Hard tamper	Zeroization	N/A (automatic)	N/A
Soft tamper	Module reset	N/A (automatic)	N/A
External warning	Warning	module start	appl. discretion
Low battery	Warning	as frequent as feasible	(replace ASAP)

Table 7 Physical Security: tamper types and recommended actions

11.2 Module initialization and delivery

The module is initialized at the factory. Internal controls guarantee that each one may be initialized only once.

Once a module has been delivered, and put into production, its configuration should be logged, to verify that it is fully operational and is loaded by an approved code level. (The application-specific details of this verification are available outside this policy.)

11.3 Miscellaneous

Note that the module is more sensitive to environmental conditions than most general-purpose devices. Environmental requirements, specified in a platform-dependent manner, are safely within the range encountered a well-managed and reliable enterprise computing environment.

Security officers and users should verify the identity and configuration of the module before utilizing its services. If the card identity (device key or serial number) does not match security officer/user expectations, applications should investigate the discrepancy and react in a prudent fashion. Module code configuration, returned by a Segment 1 query for all segments, may be verified by all officers and users.

(See also verification of module FIPS mode.)

Contents

12 Glossary

CCPs are *card configuration parameters*, configuration state descriptors that are critical, nonsecret properties of a module. Such critical information includes segment code and segment ownership (i.e., officer public keys). These parameters are not secret, but their consistency is critical for proper card operations, therefore they must be enumerated when describing module configuration.

Device keypair is a device-specific public-key keypair generated and retained by Segment 1. It is non-exportable, traceable back to the IBM factory CA through a certificate chain, and may be used by external parties to verify the identity of a module.

HLM *Hardware Lock Microcontroller*, a dedicated microcontroller assisting the module CPU with access control and management of persistent storage.

KAT Known Answer Test

Miniboot software component of module firmware.

Miniboot functionality, together with POST, roughly corresponds to those of a system BIOS in PCs (with obvious additions to cover cryptographic functionality and module-specific hardware).

POST Power-On Self-Test, infrastructure tests resident in ROM and flash.

RAS Abbreviation of *Reliability, Availability, Serviceability*

Policy information: `$Revision: 1.40 $`, last archived: `$Date: 2005/04/04 14:37:25 $`, check-in by user *tvi (Tamas Visegrady)*.