IBM 4765 Cryptographic Coprocessor Security Module

Firmware identifier `e1ced7a0`

Security Policy

Advanced Cryptographic Hardware Development
IBM Poughkeepsie and IBM Research, Zürich

December 13, 2010

Policy revision: 135, last archived: 2010.12.13. 19:07:08.

Contents

1 Scope of Document

This document describes services that the security module of IBM 4765 Cryptographic Coprocessors ("*the module*", also known as "the module of *the 4765*") with Miniboot software resident in ROM and code flash, provides to a population of security officers, users, and the security policy governing access to those services.

Firmware identifier refers to unambiguously identifiable leading characters of Segment 1 (firmware) hash, a unique value describing firmware configuration. The actual value, a SHA-256 hash of the segment image, is returned by card configuration queries. This policy applies to the following firmware identifier: `e1ced7a0`, when loaded to cards with supported hardware part number. Please see Section 7 (p. 24) for the validated combination of hardware and firmware.

Describing the module, this document is built on the foundations of previously validated IBM 4764 and 4758 families–see page 28. *We use "module" to refer to the security unit, which features the security boundary as its external surface (Fig. 4). If we mention properties of the entire card assembly, such as the PCIe board the module is mounted on, the distinction is unambiguously noted.*

Background The 4765 is a programmable secure coprocessor. Its module consists of:

- base *hardware*, a general-purpose environment with security-relevant additions, such as cryptographic accelerator hardware and tamper-protection circuitry;
- hardware-based partitioning functionality, creating four different, hierarchical layers ("Segments"), primarily separating infrastructure (*Segments 0/1*) from OS/applications (*Segments 2/3*).
- *embedded firmware* not observable to the outside; executed by the internal security processor, the SSP. Embedded firmware also contains *power-on selftests* (POST), segmented along with the rest of the code, Segments 0 to 2.
- *Miniboot* software, which controls security and configuration of the device, and provides externally visible services
- higher *system software* and *application* layers.

  OS/applications are executed on "the" processor ("module CPU"), physically separated from the SSP.

Note that this policy covers services of trusted, lower layers of internal firmware (Layers 0 and 1, and a stub of Layer 2). Higher layers, OS and applications (2 and 3) are not included in the current validation. *Our security foundations do not require a cooperative or trustworthy OS/application for consistent and secure Miniboot operation.*

The cryptographic boundary is the enclosure of the self-contained module of 4765 cards (hardware part number 45D6048). The module is labeled unambiguously with model and part numbers of the host PCIe card, and that of the module itself (Fig. 4). The correspondence between end-user product, module, and security policy is self-explanatory.

We allow Miniboot to distrust and influence OS/application behavior. Internal, non-infrastructure code is executed on a different processor; access control of Miniboot secrets is enforced by infrastructure and is not influenced by OS/application code. Inter-processor interaction is limited to the following:

1. The SSP may reset the module CPU at any time. Most Miniboot *commands* do this, *queries* generally do not.
2. The SSP may pass data to the module CPU, shared through regions that are read-only for the module CPU.

   Data sharing is generally unidirectional, and is controlled entirely by the SSP. We describe exceptions where applicable.
3. The SSP receives and acts on module CPU status output. The only security-relevant instance is the SSP waiting for successful module CPU startup testing, as performed by POST 2. In this case, the code executed by the module CPU logically belongs to firmware, i.e., it is part of privileged code, out of OS/application control.

The combination of hardware and Miniboot provide security foundations of module. What a particular instance ends up doing is controlled by higher software layers. However, what goes into these layers, and how their secrets are preserved or destroyed, is controlled by Miniboot. Miniboot also provides "outbound authentication" (OA), module-internal signing services to securely authenticate module entities. Applications can build on OA to establish trust in other entities, proving that their instance runs within a specific module, including unique identification of the particular module.

Validation of this basic platform establishes that, no matter what is loaded into Layer 2 and Layer 3, our platform is secure:

- Miniboot always correctly configures and identifies what's in these layers
- Only Miniboot is allowed to update internal executable content, both privileged and OS/application code. This makes our system immune to infrastructure-level compromise by hostile OS/application code.

- If an entity uses outbound authentication, which Miniboot validates to belong to Miniboot of an 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.

For OS/application code in Segment 2/3, the internal infrastructure is not modifiable. Therefore, for a compound security validation combining Segment 0/1 and 2/3 segments, Miniboot will be applicable as described in this document; for OS/application code, security requirements of non-modifiable environments apply.

Follow-on Hardware Current hardware offers improvements over previous hardware generations:

- Separation of firmware security management (dedicated "service processor") and OS/application execution. *OS and applications execute on a processor without write access to code flash*.

  Since functionality is now physically separated between trusted and untrusted code, the security infrastructure is simpler than in previous card generations.
- *The module CPU is a redundant embedded PowerPC (405Gr).* Replication, while itself software-transparent, allows simplification of certain self-tests, as module CPU failures are detected through redundant computation.
- CPU-type device are integrated into an FPGA, reducing physical size of a potential tamper target.
- Integrated tamper detection, response, and *actively erased BBRAM* regions replace previous discrete tamper circuitry. Core secrets reside within the "high-speed erase BBRAM" (HSEB).
- Significantly increased memory sizes, both persistent and transient memory.

  In addition to the HSEB, traditional BBRAM sizes also increased. Slower flash chips have been replaced by BBRAM, without impacting applications using persistent memory. Traditional BBRAM is crowbarred and discharged upon tamper events, as was in previous 47xx families.
- Hardware support for additional algorithms, such as SHA-256 and HMAC
- Hardware supports larger modular math calculations than previous card families
- Directly-connected USB port, with limited device/type support

Other, less significant hardware improvements are described as appropriate.

> Note that the 4765 designation, in IBM terminology, is a *machine type/model number* of an entire card assembly, or simply *model* in casual use. Modules may be assigned different *feature codes* in configurations, especially if embedded in another subsystem, such as I/O boards in mainframes. Feature codes containing the same cards may also be different in different server platforms. *Since modules only trust entities within their secure enclosures, and their security officers, the actual host platform and further packaging does not affect our security foundations.*
>
> Independent of the actual final feature code, the card machine type 4765 does not change.
>
> Since the module does not (need to) trust its PCIe host, therefore it is prudent and reasonable to use the card-specific designation to identify modules. In certain cases, references may still be made to the behavior or PCIe properties of the host system, irrespective of the actual platform. Such distinction is necessary, for example, where discussing connectivity tests, which require host interaction.

2 Secure coprocessor overview

A multi-chip embedded product, the 4765 is intended to be a high-end *secure coprocessor:* a device with a general-purpose computation environment and high-performance crypto support, which executes software and retains secrets, despite foreseeable physical or logical attacks. Customers can use this secure platform as a foundation for their own secure applications, such as high-assurance digital signature generation or financial transaction processing.

Miniboot Base Miniboot code helps achieve security goals by permitting software:

- to load and execute safely, in a controlled manner,
- allow entities to authenticate interaction with a specific untampered device in a specific software configuration,
- maintain a consistent host-visible module state, or force the module to stop if in an unexpected state.
- update code, if authorized, including updates to portions of Miniboot itself.

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.** If a user cannot verify that their crypto provider is both untampered, and operating the intended software, then their entire cryptographic operation may be compromised.
- **Distributed applications.** Many scenarios require one party to be able to trust computation 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 outbound authentication: internal, non-exportable private keys can sign things output from the module. OA!features are integral to Segments 1 and 2; Segment 3 entities—applications—access OA services through an exposed Segment 2 interface. While offering similar services, OA signatures distinguish Miniboot and Segment 2 certificates, therefore other parties may unambiguously identify Miniboot and OS/application originated authentication.

Maximum flexibility, minimal trust We obey previously described security rules while accommodating constraints:

- no trusted couriers or on-site security officers are needed to operate modules
- IBM maintains no database of device secrets
- IBM does need to see application software
- rewritable OS or applications can fail, or behave with malice, without compromising the integrity of lower layers
- IBM or other infrastructure developers have no "backdoor access" to customer's on-card secrets. Obviously, one can not provide guarantees about third-party code within the internal OS or applications, such as custom extensions. Assurance of such code is outside the scope of this document.

Secure Platform Our goal is to produce secure infrastructure on which developers—including IBM—can build secure applications. Our module, for validation, consists of all hardware within the secure boundary, along with the foundational Miniboot software.

By obtaining FIPS 140–2 validation for our *hardware* and *bootstrap/configuration control software* (Layer 0 and Layer 1, plus POST 2, see Fig. 5), we make it easy for developers to build and deploy secure applications. Obtaining FIPS 140–2 validation for such applications would require additional documentation and a separate validation for software built for our module's environment, having it evaluated for secure operation specifically within our module. *Software evaluated in this environment would inherit the physical protection afforded by our Level 4 enclosure.*

Validating this platform at Level 4 customers the flexibility to design to any FIPS 140–2 Security Level of any code built on top of the 4765 infrastructure.

More information For details and history of the security architecture of the IBM 47xx families 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.

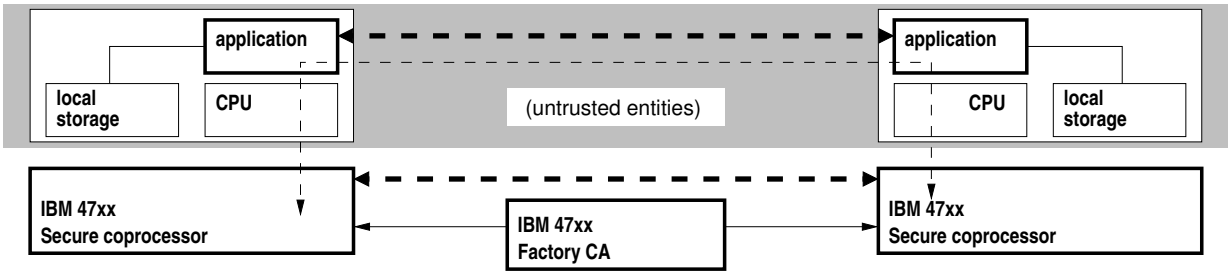

Figure 1: We enable users, who have never met, to use our hardware, download software from their chosen security officers, then interact securely—each able to verify that they are talking to the proper counterparty

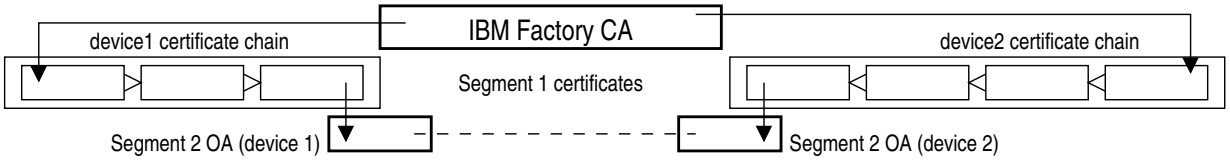

Figure 2: Cards' mutual authentication through the Factory CA, certificate chains of Miniboot1 keys, and Segment 2 OA

Architecture and resources The module incorporates state-of-the-art hardware security (Fig. 3), including:

- protective,tamper-respondent matrix to monitor for intrusion and adverse physical conditions
- tamper detection and response circuitry, with active wiping of secrets at microelectronics latencies
- modular math engines
- AES, TDES hardware, the latter usable for DES for legacy compatibility
- SHA-256, SHA-1, and SHA-224 engines, as standalone digest or in HMAC
- hardware-based random number generation for seeding; RNG complies with FIPS 186-2, Appendix 3.1

Physical security Our device is Level 4-tamper-protected for life, from the moment it leaves the factory. When internal tamper circuitry—which is *always* active—detects physical or logical attacks, it near-instantly *zeroizes* internal secrets by "actively erasing" memory devices (BBRAM), through a chip-integrated feature. In addition to active wiping of the core secrets, special-purpose wiping code purges data buffers in the communications FPGA. *The most critical regions, those within the HSEB, are actively erased within microseconds; external chips (which, by then, are effectively also lost due to loss of their HSEB encryption keys) are discharged slower, at normal RAM discharge rates*.

Non-zeroized memory devices either discharge and lose contents in milliseconds' range if power is removed (SDRAM) or lose their encryption key when HSEB is zeroized.

The only chip that stores persistent secrets in the clear, *the HSEB, is actively protected against memory imprinting* by periodic bit inversion. Bit inversion process flips BBRAM sections' polarities, preventing recovery of long-term BBRAM secrets through transistor-level resident damage. This bit inversion is a chip-internal feature, not influenced by other components, and therefore is immune to potentially untrustworthy code in Segments 2/3.

Protection circuitry also detects and responds to other environmental attacks, including extreme temperature or voltage ranges. Clock reconstruction and conditioning isolates clock signals within the secure enclosure from the external PCIe interface. *Temperature and voltage protection is integrated, reacting when either external analog sensors, or on-chip sensors indicate non-operational ranges.* The combination of external components and substrate-internal ones protects core secrets even against localized attacks—such as attempts to cool down only parts of the module.

The module monitors removal from its PCIe slot. If it is removed from the PCIe slot hosting it, the *external warning—"intrusion latch"* tamper bit, an unfortunate historical name—is set to indicate the removal. This event does not cause zeroization, but software may choose to respond to it and intentionally zeroize module secrets.

Table 9 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 accessible. In addition to hardware-initiated tamper, a card-specific signed Miniboot 1 command ("Software tamper") may also trigger a tamper response.

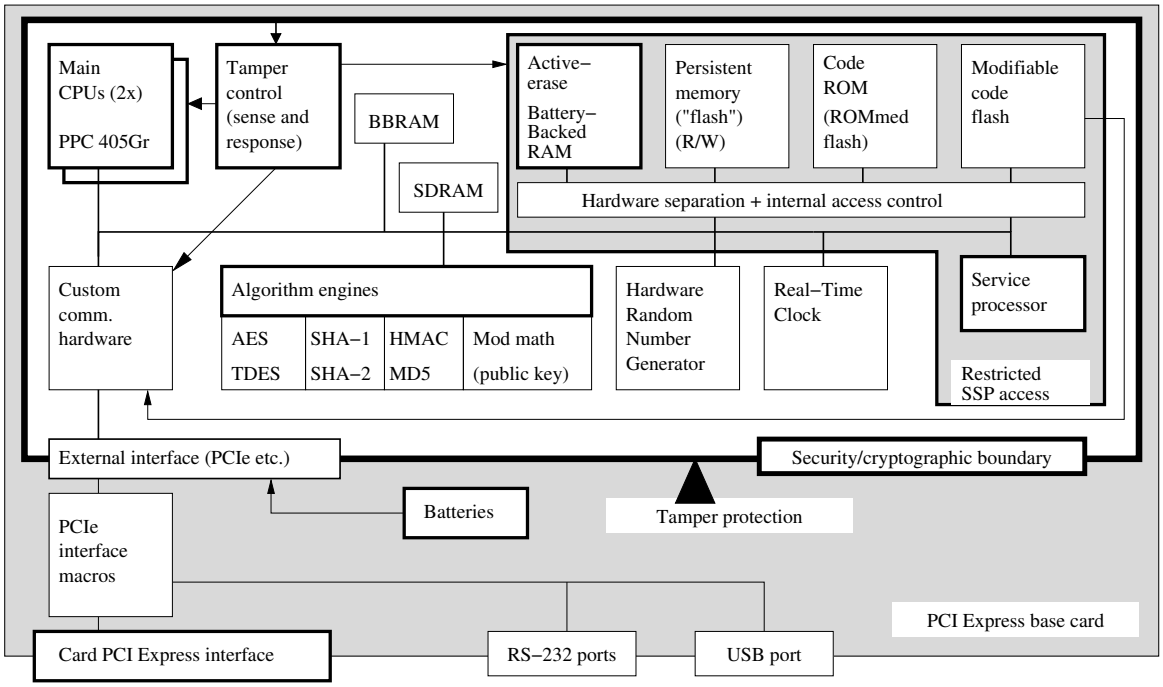

Figure 3: Module hardware architecture, with directly connected components on the hosting PCIe board

Figure 4: The module as mounted on its PCIe board

Software Architecture Internal software is divided into four layers. The foundational two layers, and a stub in the third layer—submitted for this validation—control the security and configuration of the device. These layers come shipped with the device.

- Layer 0: Permanent POST 0 (Power-on Self Test) and Miniboot 0 (security bootstrap). This code is in ROMmed persistent memory, serving as a trusted bootstrap step for the entire module.
- Layer 1: Rewritable POST 1 and Miniboot 1, responsible for most of user-visible infrastructure functionality.
- POST 2, while executed by the module CPU, is logically controlled as part of Layer 1. It performs infrastructure selftests specific to the module CPU, on behalf of Miniboot 1.

POST routines perform initial and higher-level testing of card infrastructure. If both SSP POST passes are successful, and POST 2 reports success of the module CPU tests, 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.

Note that *POST 2*, an executable region logically controlled by Miniboot, is considered part of Layer 1, even if it is executed by the module CPU (Fig. 3,5). No module CPU-controlled code may interfere with POST 2 execution, and the separation is therefore justified. Specifically, POST 2 gets control before higher-level applications—immediately after module CPU reset, before any OS. POST 2 does not get access to secrets, operating on fixed data and tests, and it must be approved by the Layer 1 Security officer to load (being part of Segment 1 firmware updates).

The upper two layers customize the operation of each individual device. *Note that the following layers are not included in the current FIPS 140–2 validation.*

- Layer 2: System software. Supervisor-level code, excluding the startup stub (POST 2).
- 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 untrusted environments. See Fig. 5 for the distribution of code layers.

Post-tamper firmware A noticeable difference between previous 47xx revisions and the current generation is termination of Miniboot services upon tamper. Previous generations allowed Miniboot 0 to survive and *revive cards—but not secrets*—after tamper. Revival is no longer supported, simplifying security foundations: we no longer need to retain secrets previously used by revival Since we removed Segment 0 secrets completely, no Miniboot 0 field commands need to remain. Removal of post-tamper Miniboot functionality also allows us to *destroy every card-resident secret during tamper-response*. We actually disable production firmware in a tampered card: this "afterlife", without any secrets, is not security-relevant.

In addition to field-visible code, cards contain an inactive Segment 1, an extended POST version, which the card reverts to after a tamper event. This image is activated when tamper response destroys card secrets, and it allows failure analysis on cards in case of user interest. Earlier 47xx variants required disassembly of tampered cards to achieve the same, which destroyed traces of tamper history.

Code activated after tamper response is intentionally incompatible with production POST/Miniboot images, including different host interaction. More detailed investigation, such as *the lack of OA capability of cards after tamper, makes it impossible for a tampered card to impersonate a working one*. The apparent interface change, and incompatibility with normal production host drivers, makes this distinction obvious even to non-OA-aware applications.

Since post-tamper code is dormant, and only gets activated when card secrets have been zeroized, it is not relevant for our regular operations. We only mention it for completeness.

Memory Non-volatile memory components consist of battery-backed static RAM (BBRAM). Certain BBRAM regions are designated as "flash" for historical reasons, while they correspond to a previous persistent-memory subsystem, which has been merged into logically segmented BBRAM. Memory resources are organized according to this layer structure.

Persistent code flash is organized into four segments (layers). Layer 0 is boot-block ROM. Modifiable layers have two copies, providing *atomic updates*. Since Miniboot 1 supports in-field firmware repairs, it's critical that a working copy of Miniboot 1 itself always be present.

As described above, persistent, OS/application-modifiable data "flash" resides within a BBRAM region, and it is used for persistent data. It is encrypted and handled as any other persistent storage would, just as previously actual flash was. Applications on the module CPU use this memory transparently, oblivious to the OS-supplied encryption.

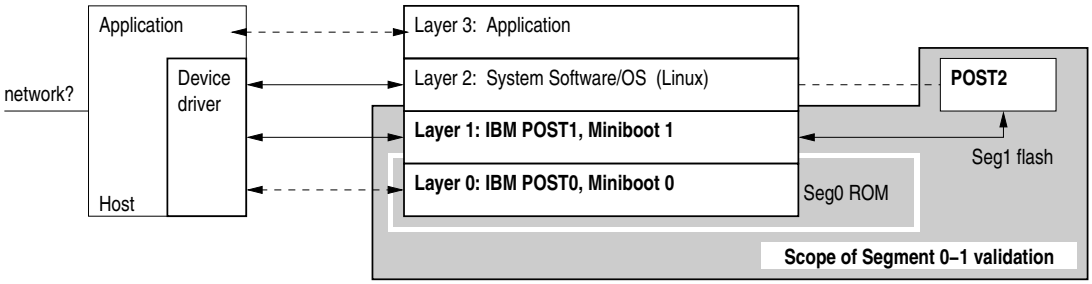

Figure 5: Module software architecture.

Hardware Locks Since Miniboot is restricted to a certain part of the hosting FPGA, and its resources are unavailable to the module CPU, our hardware segmentation is a considerably simplified version of previous "hardware lock management" (HLM). In our simplified model, Miniboot may write all code—in flash—and may push information to the module CPU, while the latter is restricted to read-only access to designated shared regions (Fig. 3, 9). We continue to refer to the remaining functionality as "HLM", even if has been merged into base infrastructure.

As access to memory regions is unconditionally restricted by hardware, much of previous complexity has been removed. The current generation, in fact, does not require the granular racketing present in the predecessor 47xx variants: *we effectively treat any module CPU code as untrusted* from a Miniboot perspective (POST 2 is trusted to return a pass/fail result).

- The "HLM" also contains, under hardware enforcement, the *"factory sticky bit"* ("initialized" indicator). Once this hardware-protected bit is activated, it will never revert to the non-initialized state.

  The initialization bit prevents the module from reinitialization, once it has completed the process. It is set upon device—Segment 1—keypair generation, and allows us to reinitialize cards if they fail intermittently during manufacturing. Activating this bit disables Miniboot services limited to factory use, for the rest of module lifetime.

- The "sticky bit" implementation resides within dedicated hardware. This hardware-enforcement—with Miniboot assistance—guarantees that the card follows its expected lifecycle. Unexpected state of this state model triggers an involuntary card tamper, assuming some fundamental hardware failure.

Hardware separation of the SSP and module CPU is a critical part of ensuring that the Miniboot security software works despite potentially arbitrary software in Layers 2 and 3.

Included Algorithms

The module includes and uses the following FIPS approved algorithms:

- SHA-256, restricted to byte-granular input (SHA Cert.#1188)
- SHA-1, SHA-224, restricted to byte-granular input (SHA Cert.#1188) (both latent, not used or exported as a service by Miniboot)
- RSA signatures, ANSI x9.31-compliant, with SHA–256 hashes (RSA Cert.#621)
- software RNG, compliant with FIPS 186–2, (General purpose), Appendix 3.1[1] (Cert.#722)
- DSA, compliant with FIPS 186–2 (Cert.#419). This functionality is not available as a service without an application loaded.
- HMAC/SHA-256, HMAC/SHA-1, HMAC/SHA-224 (not currently used by POST or Miniboot) (Cert.#754)
- AES, ECB and CBC modes (128, 192, and 256) (Cert.#1294). This functionality is only used internally within Miniboot, not available as an external service.

In addition to the above, Miniboot also has internal access to TDES (112 and 168-bit keys) in hardware (Cert.#911), but does not use TDES in any protocol. TDES services are not available as a service without applications loaded. Note that Miniboot has access to TDES, but field services do not actually encrypt data.

[1] limited to with fixed-size seeding, with SHA-1-sized (160 bit) input

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 1: Module Security Level specification.

SHA-256 is used in as an integrity check of certain internal structures. Firmware integrity is protected by this hash, *protecting against random corruption*[2]. While none of this SHA-256 calculation is externally observable, use of the algorithm for EDC generation is mentioned here for completeness.

The module includes FIPS allowed internal seed generation, feeding an Approved RNG.

The module also includes the following *non-FIPS-approved* algorithms:

- MD5, on byte-granular input (latent, not used by Miniboot)
- DES (latent, not used by Miniboot)
- DES MAC (Miniboot uses DES MAC as integrity check for certain internal structures, not available as a service)

Higher-level software layers may provide and other algorithms, out of scope of this document.

3 Cryptographic module Security Level

This module is intended to provide Security Level 4 protection. See Table 1.

The *operational environment qualifies as "limited"*, being access-controlled separately, out of OS/application control. While persistent storage may be modified by certain authenticated Miniboot 1 commands, operators can not generate the necessary signatures. *For everyone except the Segment 1 security officer, the card environment is non-modifiable.*

Segments 2 and 3, i.e., software layers, are prevented from modifying code storage. Due to these restrictions, FIPS 140 requirements on the operational environment do not apply to the module if they do not allow execution of host-supplied code (which Miniboot itself can not check against, and therefore does not enforce).

[2] these critical structures are not externally observable, but must be protected against failure of storage medium

Signal	Notes	Type
PCI Express signals: 4-lane (x4) external		
PCIe data/addresses	bidirectional	Data, input; Data, output
PCIe control	bidirectional; PCIe v1.1 compliant "single function" device	Control input Status output
Auxiliary signals, tunneled over shared flexcables		
RS-232 ports	*only used as output by current IBM firmware*	Status output
USB port	bidirectional; may tunnel other signals (such as Ethernet-over-USB) *not used by current IBM firmware*	N/A (with current firmware)
PCIe power	3.3 V	Power
Battery power	variable, nominal 3.0 V	Power
External warning	host connectivity test, latching removal from host bus monitored within module	Control input (from sensor) Status output (to host)

Table 2: Connector and signal types

4 Ports and interfaces

The module communicates with its host through a PCIe connector hosted on a PCIe main board. Flexcable connectors connect the module to the PCIe board; these connectors carry signals summarized in Table 2.

In the configuration submitted for FIPS 140–2 validation, RS-232 ports are used for status output during self-tests. They do not serve as inputs. Production firmware does not exercise the USB port. *In typical mainframe use, modules are encapsulated within closed metal "books"—board enclosures—and these ports not externally accessible.* Custom firmware in Segment 2 or 3, such as our development toolkit, may utilize these ports, which is outside scope of the current validation.

5 Self-tests

The module 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 SSP code, POST 0 and Miniboot 0, are checked for integrity through embedded 32-bit checksums. In case of checksum mismatch, the code halts itself (POST 0) or is not even permitted to execute (Miniboot 0, inhibited by POST 0). This code is executed only at startup.

Flash (code) failures are detected and corrected where possible in Segment 1, reverting to the unaffected image if possible (redundant copies of Segment code are stored in flash). Segment 2 and 3 code corruptions are detected in a similar fashion. The same integrity check applies to Segment 2 and 3 secrets. Checksums are checked upon each write operation on a continuous basis; a 64-bit DES MAC is used as segment checksum. *The segment integrity check includes all executable segments, not just lower Layers.*

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. Such checks are outside the scope of the this FIPS validation, but are mentioned here for completeness.

Functional integrity of hardware components is tested through 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 POST 0, before execution continues to Miniboot 0. These checks verify fundamental functionality, such as proper execution control, load/store operations, register functions, integrity of basic logical and arithmetic operations, etc. Once SSP tests pass, processor failures are monitored using other error-checking mechanisms, such as parity checks of the PCIe bus or memory ECC.

Substantially similar functionality is performed by the module CPU during startup. POST 2 verifies the integrity of things "on the module CPU side", allowing Miniboot 1 to launch further module CPU code if it has been tested. POST 2 is, by construction, the first code the module CPU executes when released from reset.

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.

During FPGA code updates, the new FPGA code is digitally signed as part of "Segment 1". The FPGA programming file is modified only after if this signature has been verified. The bitfile internal checksum is used in addition to initial integrity checking, but it only extends the assurance provided by the digital signature.

Crypto ASIC integrity is verified by known-answer tests (KATs) at startup, covering all control modes. These tests implicitly cover internal FPGA transport as well.

During regular operations, the crypto ASIC covers all traffic through combinations of redundant implementations, CRCs, and parity checks, in engine-specific ways. Failures are reported as specific hardware failures.

Modular math engine self-tests cover all control modes, and different sizes of modular arithmetic. Testing covers only modular arithmetic, up to full exponentiation and padding, but not full protocols (such as digital signatures).

A separate, fully specified KAT test is performed on the DSA implementation, including a test through a predefined "random" sequence. Note that DSA code, in our current firmware setup, is tested but not used by Miniboot.

The RSA implementation is tested through separate 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).

KATs cover some of the algorithms present in hardware but never used by Miniboot itself (Table 4).

Deterministic random number generator (postprocessing) is covered by a KAT: seeded with a known value, output of the generator is compared against the expected stream. The RNG is compliant with FIPS 186–2, Appendix 3.1 (general purpose), and shares code with the DSA implementation.

At runtime POST sanity-checks the hardware source, through basic statistical tests.

Interactive communications tests verify that the card PCIe bus is functioning properly. Testing covers both mailboxes (FIFO-organized registers) and data/addressing through the PCIe connectors.

Communication channels are tested through real and simulated host–module handshakes. These tests verify host–module connections reliability, transporting fixed patterns across communication channels. Similar testing covers mailboxes and transport FIFOs, allowing the module to discover most connectivity problems early.

Externally visible channels are untrusted; the connectivity test is a preemptive measure against channel data corruption.

As part of automatic self-tests, *critical functions tests* cover module CPU cache control logic (data/instruction), processor registers, and instruction set; PCIe bus transport integrity (including communication mailboxes), and memory 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 PCIe communications are tested.

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

Two PowerPC cores execute in transparent "lockstep", monitoring for mismatches between the synchronized cores, with redundancy protecting against failures.

Theoretically, the two processors could be released from lockstep, but this capability is currently not exploited.

Pairwise consistency test on RSA and DSA operations

Continuous integrity checks on modular math arithmetic (including RSA) implemented in hardware. Data mismatch during modular math operations triggers a hardware error.

Cross-checks between redundant, independent TDES engines (optionally useable as single-DES, with similar redundancy)

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 SHA-1 and SHA-256 hashes), partially redundant data flow and control logic.

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

As shown in Table 4, current IBM firmware uses RSA officer keys exclusively, and generates only RSA signing keys within the module. The only hash function used is SHA-256. Internal object encryption uses AES. Hardware-resident TDES, HMAC, SHA-1, SHA-224, DSA, or MD5 implementations are not used by Miniboot itself, only–possibly–by Segment 2/3 code, outside the scope of this document.

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	168 (192)	ECB	encryption, decryption
		CBC	encryption, decryption
Asymmetric algorithms			
DSA	1024	N/A	signing, signature verification
RSA	2048,4096	N/A	signing, signature verification
Modular math	1024,2048	N/A	(comprehensive test of primitives, up to exponentiation)
Hash algorithms			
SHA-256	N/A	N/A	(hashing)
SHA-1	N/A	N/A	(hashing)
MD5	N/A	N/A	(hashing)
Deterministic random-number generator (RNG)			
FIPS 186–2, App. 3.1	N/A	N/A	generate known "random stream", signatures from fixed seed

Table 3: Algorithm known-answer tests

Used by Miniboot	Latent *unused by current IBM firmware*
Approved	
RSA SHA-256 AES RNG	DSA TDES SHA-1 SHA-224 HMAC with SHA-256, SHA-1, SHA-224
Allowed	
NDRNG (seeding approved RNG)	-
Non-approved	
DES MAC as internal error-detection code	MD5
-	DES

Table 4: Algorithm utilization summary

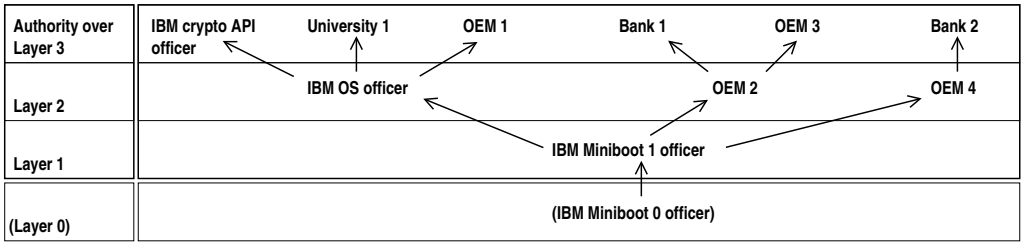

Figure 6: Each device has at most one officer in charge of each layer. The space of *all* officers over *all* devices is organized into a tree. This diagram shows an example hierarchy.

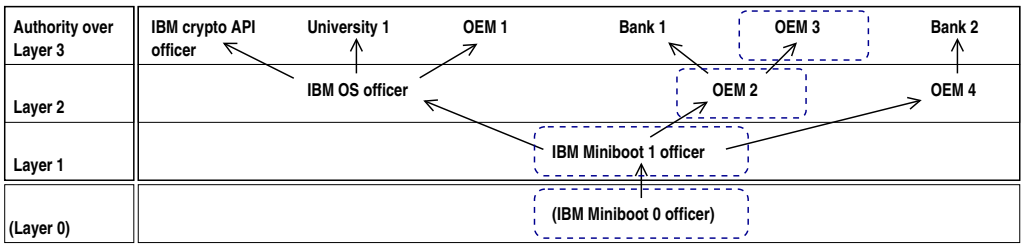

Figure 7: Within this example owner hierarchy, one family of devices might have a Layer 2 controlled by "OEM2" and a Layer 3 controlled by "OEM3"

6 Roles and Services

6.1 Roles

Our module has roles for *Officer 0*, *Officer 1*, *Officer 2*, *Officer 3* and a generic *user*. *Officer 0* is only active within the factory, and is not relevant to the discussion of field-accessible services.

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

The controlling entity does not have to be co-located with the card—in fact, it usually is not. *We represent officers as their public keys, equating presence of a digital signature with officer intent.* We enforce a tree structure on officers:

- All cards will have IBM as their *Officer 0*.
- All cards will have IBM as their *Officer 1*.
- If layer n is unowned in a card, then no layer $m > n$ can be owned.

Fig. 6 through Fig. 8 sketch examples of segment ownership structures.

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.)

We additionally have a notion of User: someone who interacts with the card wherever it is installed. (See also Section 7.1).

Applications may define other classes of principals. Application-level additions are outside the scope of this document.

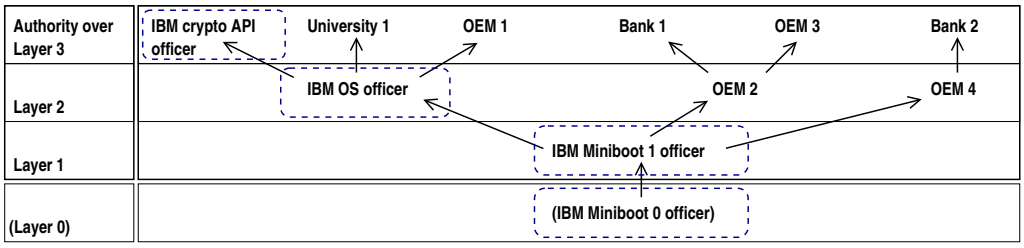

Figure 8: Within this example owner hierarchy, another family of devices might have the IBM OS/Control Program in Layer 2 and an IBM crypto API, such as PKCS#11, in Layer 3.

		Services ∗ marks both MCPU−resetting and non−intrusive variants	Roles Officer 0 (IBM)	Officer 1 (IBM)	Officer 2	Officer 3	User
Queries		Query Status ∗	Unauthenticated				
		Query Signed Health ∗					
		Query certificate list					
		Algorithm test					
Commands	Run	Continue to Segment 1					
		Continue to Segment 2					
	Init	(IBM Initialize)	Perform without restrictions while within factory FACTORY USE ONLY				
		(IBM Burn) (Segment 1)					
	Kill	Software tamper	yes				
	Officers	Establish Officer 2	yes				
		Establish Officer 3		yes			
		Surrender Officer 2		yes			
		Surrender Officer 3				yes	
	Code management	Ordinary Burn (Segment) 1	yes				
		Ordinary Burn (Segment) 2		yes			
		Emergency Burn (Segment) 2	yes (cross−signatures)				
		Emergency Burn (Segment) 3		yes (cross−signatures)			
		Ordinary Burn (Segment) 3 (concurrent update)				yes	

Table 5: Miniboot command/query policy.

Table 5 summarizes what commands are allowed for what roles. Note that the table includes certain factory-only commands, which may only be performed by the factory CA.

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. *Officer authentication is equivalent to signature generation/verification.*

Fig. 10 illustrates how the commands change initialization of the device; Fig. 11 illustrates how the commands change the configuration of Segment 2 and Segment 3.

6.2 Operations

Our module provides Miniboot *queries* and *commands*. 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 SSP to begin executing from a fixed address in Segment 0, which contains POST 0 and Miniboot 0. If POST 0 fails, the device halts. If POST 0 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 POST 1 and Miniboot 1 (MB1). If POST 1 fails, the device halts. If POST 1 is successful, then Miniboot 1 executes. It listens and responds to zero or more queries, followed by exactly one command; at the same time, if the command has reset the module CPU, POST 2 tests the module CPU infrastructure. If the command is a *Continue* and Segment 2 is deemed safe (POST 2 terminates), 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, either self-test or functional one
- detection of any other condition requiring alteration of configuration

Halting upon command completion is a design decision: always halting makes it easier to be sure that precondition checks and clean-up are applied in a known order. POST—in general, 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, such as controlling the power status of its PCIe slot
- triggering the designated control bit in the Bus Master Control/Status Register accessible from the PCIe host (it forces a module reset through the external PCIe bridge chip).

Internal firmware design acknowledges such interruptions; internal code must accommodate that the module may be reset *at any particular point in time* (such as performing atomic transactions, for example). Host drivers generally drive a state machine in sync with the modules they drive, and will transparently issue resets whenever appropriate.

On a software level, 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 driver determines that the device is not already open. In this case, the default IBM-supplied host drivers will transparently reset the device and also execute Miniboot 0 Continue and Miniboot 1 Continue, to try to advance to Program 2 code.

Receipts Upon successful command completion, Miniboot 1 returns a signed receipt, proving to a remote officer that the command actually took place, on an untampered card. *Protocols include nonces to prevent replay.*

6.3 Inbound Authentication

Miniboot authenticates each command request individually.

For $0 < N$, 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 RSA-based (ANSI x9.31 padding, SHA-256 hash, 4096-bit RSA). *Forging 4096-bit RSA signatures on segment contents is assumed to be infeasible.*

After module initialization, Segment 0 commands are no longer available, therefore we do not effectively have Officer 0 authentication. Segment 0 field operations are restricted to queries, not requiring authentication. Typically, host drivers separate userspace—non-OS code—from such access, which is outside the scope of this policy.

6.4 Outbound Authentication

At the last stage of manufacturing, Miniboot on a card generates its first keypair. IBM, through a Factory CA, 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 successor keypair and signs the new public key with its current private key. These transition certificates establish transitive trust in the sequence of Miniboot 1 keys.

If application configuration changes, Miniboot 1 also generates and certifies a keypair for Layer 2, replacing the previous one. This certification binds the keypair to a specific Layer 3 configuration. The binding between Segment 1 and 2 keypairs, 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 (Fig. 2).

6.5 Keys (secrets) and critical configuration parameters

CSPs are internally generated by Miniboot, and are not exported outside the secure boundary. Officer identity, and other CCPs, are imported/generated by Miniboot; some public data is shared with the module CPU, see Fig. 9. Certain CSPs are generated but not used by Miniboot, and they are owned by Segment 2/3 code, outside the scope of this policy.

Segment 0 state does not include secrets, as the module does not include authenticated Officer 0 commands. Note that this is a change from previous members of the 47xx family, where Miniboot 0 was involved with card recovery. We no longer provide a capability to restore ("revive") a module to functional—without secrets, obviously, but allowing factory re-initialization—which was a dormant feature previously (was never exercised, per company policy).

Segment 1 has a *device private key pair* ("DKP1"), generated during card initialization, rolled over with subsequent Segment 1 updates (see Table 7). This private key serves as the card trust root, and it is the most valuable card secret. It is stored encrypted with a Miniboot-owned AES key ("MBK"), which itself is actively erased in case of tamper.

The card also stores a public certificate chain, starting from its first device keypair (issued by the IBM factory CA). Querying this certificate list allows one to establish trust recursively in any current card-resident key (Fig. 2).

Segment 2/3 code offers public-key services (OA) similar to those of Segment 1, through a keypair referred to as *OA key(pair)* ("OAKP"). When generated, the OA keypair is stored encrypted into MCPU-visible storage, together with its Segment 2 flash-encryption key ("S2FK"). These keys are not used by Miniboot itself; MCPU code "owns" them. Once generated, the OA keypair is issued a certificate by the current device keypair, pushed to the module CPU, then ignored by Miniboot. Segment 2/3 code may use these keys, which is outside the scope of this document.

Generating keys in Miniboot on behalf of the module CPU allows Miniboot to issue an `DKP1/OAKP` certificate, and to erase persistent Segment 2/3 secrets even without a cooperating Segment 2/3 entity. If the S2FK is replaced, all persistent data encrypted by previous Segment 2/3 code is indirectly lost.

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

- identity of the officer controlling the layer
- code residing in this layer, i.e, the *segment hash* identifying layer code contents. This code is identified through a hash, but itself is not a secret.
- state that this program has accumulated in persistent memory

Compound 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. 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, returning registered public keys. Officer 2/3 identities are imported in the "Establish Owner" command of the corresponding segment and removed by the "Surrender Owner" commands. (Segment 1 does not have equivalents, as Miniboot 1 is always retained as an active entity.)

If present, Segment 2/3 state includes Layer 2 private key(s) noted above. These keys resemble Segment 1 keypairs: they are internally generated, are not exportable by Miniboot, but may be their exportable public-key certificates may be queried. These keys may be generated and used through an internal OA interface that mimics the public-key services of Miniboot 1, in the Segment 2. The default IBM code provides these interfaces; it supports *key generation*, *certificate (list) export*, and *signing*, 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.

Control of code is obviously also critical to module security. Segment contents are updated in sync with state and officer identity. Table 6 summarizes administrator-level actions performed by Segment 0 and 1. CSP/CCP actions show the following actions: parameters are *created* (**C**) or *imported* (**I**), they may be subsequently *read* (**R**), may *verify* signatures (**V**), or may themselves *sign* (**S**). Finally, parameters may be *destroyed* (**D**). In case of certain transactions, the sequence is indicated, such as when a device keypair gets updated: C/S/D/S shows a new key is *created*; the previous one *signs* it, the previous private key gets *destroyed*, and the new key *signs* the final response.

The following notes apply to Table 6:

1. The *User* is entirely controlled by the Segment 3 officer, and actions of Officer 3 apply to the User as well. The User entity is incapable of influencing infrastructure. In a typical application, the module-resident portion of a CSP—such as PKCS#11—would belong to the User role.

   Note that such User-level entities may further divide their responsibilities to API Security Officer and User; these distinctions are outside the scope of Miniboot policy.

2. Certain Miniboot actions, which are never performed in the field, are included for completeness. Control operations *before the module is released* are relevant, even if they may not be invoked during regular operations.

3. If Segment 3's secrets (persistent objects) are labeled to distrust Segment 1–2 configuration changes, certain changes destroy Segment 3 persistent data.

4. All secrets are destroyed by hardware-managed tamper response. Tamper does not need/use Miniboot functionality.

5. Commands' responses are signed by the device keypair.

6. Use of non-Officer public keys is outside Miniboot control, and not represented.

The available functions affect following CSPs and CCPs (see the more detailed listing under 6.6):

Query Status Read infrastructure status, including layer owners. Reset the module CPU (OS/application).

Query Status/Noreset ("Query Firmware") Read infrastructure status, including layer owners. Do not reset module CPU.

Query Signed Health ("Get Health") Read status, including owner identities and public keys. Resets module CPU.

Query Signed Health/Noreset Read status, including owner identities and public keys. Do not reset module CPU.

Algorithm test Hashes host-supplied data as an, interactive communications/infrastructure selftest. Does not access CSPs.

Continue to Segment 1 advance into Segment 1 code if status permits

Continue to Segment 2 advance into Segment 2 code if possible. POST 2 selftest must have completed successfully.

IBM Burn Load Layer 1 (owner) public key and initial code; clear all internal persistent storage.

This command starts the card lifecycle, after manufacturing tests have passed. Hardware controls prohibit it, once the card has completed initialization. We mention it here only for completeness.

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

Note that this command is only available in the factory, and mentioned here only for completeness.

Establish Officer 2 register new Officer 2 (i.e., public key)

Establish Officer 3 register new Officer 3 (i.e., public key)

Surrender Officer 2 Clear Layer 2 and 3 parameters, public keys, and persistent data

Surrender Officer 3 Clear Layer 3 parameters, public key, and persistent data

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

Ordinary Burn 2 optionally clear Layer 3 parameters and persistent data; write Segment 2 code (over previous active one)

Emergency Burn 2 clear Layer 2 and 3 persistent data; write Segment 2 code

Ordinary Burn 3 write Segment 3 code (over previous active one)

Emergency Burn 3 write Segment 3 code; clear Layer 3 persistent data

Software-induced tamper destroy *all* card-resident secrets, rendering the card unusable.

Note that this command must be targeted to particular cards, requires IBM cooperation to create (instances are unique), and is therefore not expected to be used during the lifetime of a typical deployment.

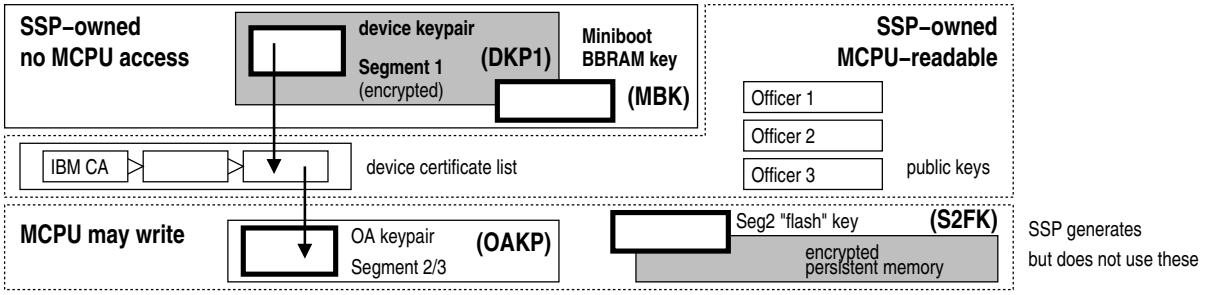

Figure 9: Layout of secrets and security-relevant parameters (overview)

Service	*Keys and CCPs*								*Notes*
	Private key		Public key				**Secret key**		
	DKP1	**OAKP**	DKP1	Off1	Off2	Off3	**MBK**	**S2FK**	
Queries									
Query Status				R	R	R			includes *Query Firmware*
"Get Health" (Query Signed Health)	S		R	R	R	R			includes non-resetting variant
Query Certlist	S		R						current+past Seg 1 certificate/s
Algorithm test									
Execution control									
Continue to Seg1									only infrastructure checks
Continue to Seg2									only infrastructure checks
Factory initialization (not field-accessible)									
(IBM Burn)	D	D	D	**I**	-	-	-	-	no secrets exist yet install factory Officer1
(IBM Initialize)	C	C	C				C	C	populate Segment 1 infrastructure
Officer administration									
Establish Officer 2	S			**V**	I				new Officer 2
Establish Officer 3	S				**V**	I			new Officer 3
Surrender Officer 2	S				**V**/D				remove Officer 2
Surrender Officer 3	S					**V**/D			remove Officer 3
Segment content administration									
Ordinary Burn (Seg) 1	C/S/D/S	D/C	D/C	**V**/**(I)**/S			D/C	D/C	Image may include new Officer 1 key Device keypair transitions New device key signs response
Emergency Burn (Seg) 2	S			**V**	V				combined signature (Officer 1+2)
Ordinary Burn (Seg) 2	S				**V**				
Emergency Burn (Seg) 3	S	D/C			**V**	V			combined signature (Officer 2+3) OA starts next "epoch"
Ordinary Burn (Seg) 3	S	D/C				**V**			OA starts next "epoch"
Card destruction									
Software tamper	D	D		**V**			D	D	public keys survive

Table 6: Roles, services, and CSP access

Key name		*Type*	*Size*	*Storage*	*MAC*	*Notes*
Private keys						
Device keypair	DKP1	RSA	4096 bits	encrypted (MBK)	SHA-256	Encrypting symmetric key (MBK) destroyed by tamper response
OA keypair	OAKP	RSA	4096 bits	encrypted (S2FK)	SHA-256	Encrypting symmetric key (S2FK) destroyed by tamper response generated but not used by Miniboot
Secret keys						
Miniboot BBRAM key	MBK	AES	256 bits	in clear (HSEB)	SHA-256	wiped upon tamper from active-erased BBRAM
Segment 2/3 flash key	S2FK	AES	256 bits	in clear (MCPU-readable BBRAM)	SHA-256	erased upon tamper through forced BBRAM discharge generated but not used by Miniboot
Seed/s						
RNG seed		N/A	160 bits	in clear (DRAM) transient seed blocks	N/A	from internal hardware source exhausted seed blocks are wiped upon use discharged upon tamper
Public key/s (relevant CCPs)						
Certificate list including DKP1 public key		see device keypair		in clear	DES MAC (as checksum)	Miniboot 1 query reports entire past history of device keypairs
Officer public keys		controlled by officers		in clear (Miniboot BBRAM)	DES MAC (as checksum)	copy is visible to the module CPU

Table 7: Keys and public-key CCPs managed by Miniboot

6.6 Queries and Commands

Table 5 summarizes queries and commands that Miniboot offers. Note that "Officer 0"—IBM manufacturing—does not interact with cards, once they have left the factory. The "User", understood as the application-level entity controlling card-resident applications, interacts with the card under controlled circumstances, and can not influence infrastructure state. (We disregard infrastructure-level or physical control, such as the ability to remove card power, for these purposes.)

Lacking direct Officer 0 and User interaction, they are present in the table without indicated actions. This is not a mistake.

Miniboot 0 Queries Miniboot 0 provides two versions of a "field" query:

- *Query: Status* This query returns general status information about the card software versions, card identification. The module CPU is reset while performing the query.
- *Query: Status/Noreset* Return status information without resetting an module CPU. Added to be able to serve queries from modern host code, which is aware of the coexistence of the SSP and the module CPU.

Legacy applications, if unaware of enhancements since the 4764 family, will continue executing, since they issue a resetting query variant (by construction: the single processor was reset while Miniboot was being queried). New applications may start to issue "Noreset" variants of queries, without impacting execution on the module CPU.

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. This command allows manufacturing to initialize cards without assuming anything in Segment 1 or above.

  Note that the IBM Burn command is not accessible once it left the factory. It is included for reference only.
- *Continue*. Transfer execution to Segment 1, if possible. Prohibited if integrity check on Segment 1 failed.

In an end-user environment, Miniboot 0 can issue only the "Continue" command 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 the Status query of Miniboot 0
  - identifying information about code and owners in reliable segments
  - host-supplied nonce to indicate freshness

  The Get Health query resets the module CPU. This behavior is equivalent to that of the 4764 and previous generations.
- *Query: Get Health/Noreset*. Equivalent to the Get Health query, without resetting the module CPU, without impacting running Seg3 applications. An alias for this functionality is "Query: Firmware".
- *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).
- *Algorithm test*. Hash user-supplied data. Using a module to calculate unkeyed hashes is overkill; this service is used by host drivers as an overall connectivity and functionality test. It passes data through all relevant interfaces, from host to hardware engines and back, and returns the SHA-256 hash of host data.

Miniboot 1 Commands Miniboot 1 provides these commands:

- *IBM Initialize*. While still in the factory: generate a device keypair, have it certified by the Factory CA, and set the "module is initialized" (factory "sticky bit") active. This command is rejected if the card has been initialized.

  The IBM Initialize command is not accessible in the field. It is included for reference only.

- *Establish Owner* n, for $n > 1$. Give an UNOWNED layer n to someone. Register a new public key as the officer controlling Segment n.

  Ownership may be established only if the segment is not claimed by any owner (i.e., is "UNOWNED"). The new owner's public key is registered, segment state is upgraded to "OWNED BUT UNRELIABLE". This state prevents execution, but allows subsequent code loading. Miniboot does not execute code from such a segment.

  Once the new owner loads code to the segment, its state is upgraded to "RUNNABLE", indicating that 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, even if their previous contents are not removed.

- *Ordinary Burn* n. Update Program n. The Segment 1 command replaces Segment 1 code, and may include a subsequent public key for *Officer 1*. Since the command *replaces* an active segment with another image, it only works for "reliable" segments—those with officers and firmware loaded. (Segment 1 is always reliable, if Miniboot runs.)

  Documentation may refer to this command as "Remote Burn." The command is signed by the originating officer.

- *Emergency Burn* n for $n > 1$. Install Program n for *Officer n*—without using current contents of Segment n.

  This version, while overwriting Segment n, is controlled by Officer $n - 1$. It is used, for example, to recover the card from lost officer keys, or other situations where the a layer wishes to override the officer controlling the lower trust level. It is also used during initial firmware loading, where obviously no previous image is present in flash.

  Persistent objects may be flagged to survive an Emergency Burn, or be destroyed during the process (i.e., allow an officer to restrict its secrets to that particular card configuration). This distinction is enforced by Miniboot 1, and OS/application officers may not influence it.

- *Continue* Continue execution to Segment 2, if possible. Possible only after POST 2 has reported that module CPU infrastructure tests passed.

- *Software-induced tamper* Destroy card secrets, triggering a regular tamper response. This command must be targeted to specific cards, therefore its payload requires IBM cooperation to produce (as it must be signed by our factory CA). Most users will not need to utilize this command, but it may be useful in restricted environments where timely module destruction is desired, but may not be performed (such as due to lack of physical access).

  Since our modules do not allow reinitialization, a successfully issued software-induced tamper is the last command ever executed by a particular module.

6.7 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.

Generally, modules react immediately to hardware faults, remaining consistent in the presence of transient failures.

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.

Control of Software. Should layer n later change in any way *other* than demotion due to failure, some *current* Officer k (for $k \leq n$) is responsible for that action, using his current authentication key.

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 entities named in this certification chain, then Bob can conclude that Alice is the entity named in that last certificate (Fig. 2).

Trustworthy state. 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.

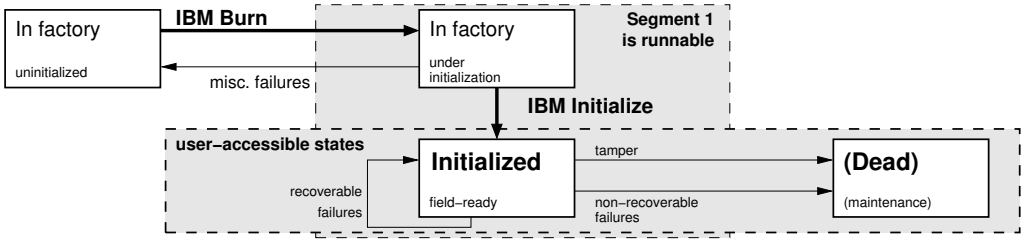

Figure 10: Configurations and main flows of device initialization, specifically Segment 1

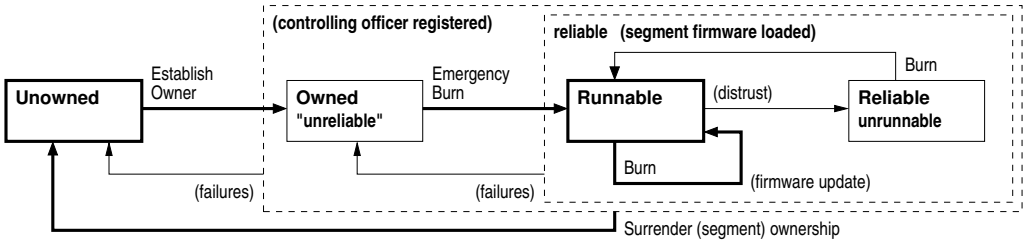

Figure 11: Configurations and main flows for Segment n, for $n > 1$. One can "Burn" or "Emergency Burn" from any of the *reliable* states into *Runnable*.

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

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

End-to-end secure code control. The module shall enable all rewritable software layers to be installed and maintained in a hostile field, without the use of trusted couriers or on-site security officers. See Fig. 12.

Module security supports the Officer and User roles as described in Table 8. We assume identity-authenticating mechanisms are as secure as the underlying cryptographic functions. Officer 0 does not operate after manufacturing, and has no identifying keys. Officers 1 through 3 are identified through digital signatures, which may be user-controlled. IBM firmware is generally signed with 4096-bit RSA keys, including images currently used. *The cryptographic strength of 4096-bit RSA keys currently used in production, the cryptographic strength is approximately 128 bits, significantly over the minimum* 10^{-6} *unauthorized success probability.*

In addition to officers, a *Generic User* role is supported by the module. The actual external user would probably interface with a Segment 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.

6.8 End of life

Security assumptions span the entire module lifetime. Tamper circuitry is activated before module initialization, and it is never deactivated. *Tamper response erases all BBRAM-based keys, resulting in a card beyond recovery at microelectronic speeds.* Other than actively-erased BBRAM-resident keys, other RAM-type devices are active-discharged. Since the core secret, the Miniboot key (MBK) encrypting the device keypair, is actively erased, remaining RAM structures are encrypted with an actively-erased key and therefore become useless faster than their (passive) discharge completes.

Following prudent security practice, sensitive intermediate values from Miniboot are purged before being released. We minimize the time window where sensitive values exist in RAM.

Role	*Type of authentication*	*Authentication data*	*Cryptographic strength*
Officer 0	N/A	N/A	N/A
Officer 1	Identity	digital signature	≈128 bits (4096-bit RSA, SHA-256)
Officer 2	Identity	digital signature	≈128 bits (4096-bit RSA, SHA-256)
Officer 3	Identity	digital signature	≈128 bits (4096-bit RSA, SHA-256)
User	(Authenticated by Officer 3)		

Table 8: Officer Roles and Authentication Mechanisms

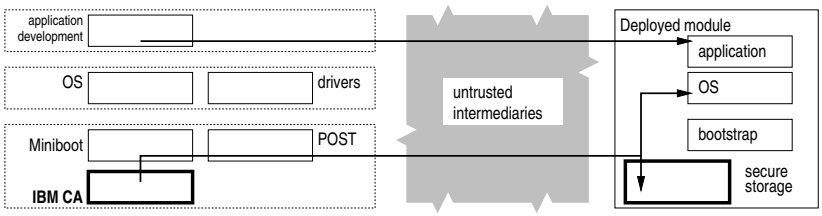

Figure 12: Firmware supports three layers of rewritable software, from potentially mutually suspicious developers, configurable in a hostile field location, with neither trusted courier nor on-site security officer.

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 wire disconnects the battery path and zeroizes the module (p. 26).

In addition to hardware-level destruction, one may trigger an Officer 1 software service to purge card-resident CSPs. This command is available to force a *software tamper,* destroying card contents actively. By design, these commands are signed to target particular cards, and therefore require IBM cooperation to create. The primary use is in environments where tightly controlled destruction is desired, without physical access. Most regular use does not require use of this service.

Note that since *our modules may not be reinitialized*, once they have triggered a tamper response, both hardware and software-induced tamper events are final.

7 Module Configuration for FIPS 140–2 Compliance

7.1 FIPS 140-related definitions

This FIPS 140 validation addresses only hardware, and Layer 0 and Layer 1 of the software—the generic device firmware, as shipped.

For the purposes of this FIPS 140–2 validation, "Officer 0" is the logical owner of Miniboot 0. Since this officer operates only in the secure factory, most of its operations are therefore out of the scope of this policy, and are only indirectly referenced.

Layer 1 includes all infrastructure, including low-level programs, and configuration. The segment image may include bitfiles for multiple hardware components–i.e., FPGA/SSP–as a single logical unit.

Layers 2 and 3—OS/application code—are excluded from this FIPS 140 validation effort. In certain cases, Segment 2 and 3 contents are explicitly mentioned in the context of Segment 0 and 1, where necessary. POST 2, logically controlled by Segment 1, is included in the validation set, since it is access-controlled by Miniboot 1, even if executed on the module CPU. Excluding module CPU code from the current validation allows other OS/application code to get separately validated on the same infrastructure, not tying card usage, for example, to the default IBM-supplied OS.

The "User" role can access a subset of Segment 3 capabilities, and can't directly influence module configuration. 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 Segment 3 officer's control.

7.2 Hardware and firmware identifiers

Firmware configuration is primarily characterized by Miniboot revision; FPGA, POST 1, and POST 2 versions are also included in "module firmware configuration". Configurations are officially tracked through Segment hashes and not individual Miniboot, FPGA, or POST revisions. We use leading digits of the Segment 1 hash as firmware identifier ("FWID") to briefly characterize Segment 1 firmware contents. Host administrator utilities generally report the entire hash.

This FIPS 140–2 validation is applicable to modules with the following combination of hardware and firmware:

Hardware	FWID	Segment 1 hash (SHA-256)
45D6048	**e1ced7a0**	`e1ced7a0 e835fb37 52ca2d21 b6a834f4 e582c2f3 1199c402 4618b38f ab1f87cb`

To be in FIPS mode, a module must have been loaded with the above validated Segment 1 revision. Security officers must verify hardware part number during installation. *Tools report the PN of the entire card assembly, not just that of the module.* These numbers are recognizably close, by construction: as an example, card 45D6047 containing module 45D6048. *Module hardware PN is displayed separately on packaging for easier tracking.*

Host drivers SHOULD provide a method to query the Segment 1 hash, issuing the Miniboot 1 *Get Health* query; this detail is platform-dependent. Platform-specific documentation describes the necessary commands or tools; the development kit also describes how the same structures may be obtained by host software.

Note that segment hashes do not correspond to the hash of the actual image file: it is calculated over a subset of the signed commands. Therefore, utilities such as `sha256sum` may not be used to identify contents of an image file directly.

7.3 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 then halts.

Since the module CPU has only read-only access to code memory, and no access to SSP-sensitive data, CSPs and CCPs that Miniboot depends on 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.

  Note that the identity of an OA-issued certificate may be unambiguously tracked, as certificates are issued by card-unique keys, which never leave the card they are resident in.

In order to actually do something, the device must be loaded with Layer 2 (and, most likely, Layer 3 as well).

Hence, to operate after bootstrap as a FIPS 140–2 compliant module, layers 2 and 3 must also be validated. The level of validation of the module in operation, as a whole, will be limited by the level of validation of these layers.

If both Layer 2 and *Layer 3 are FIPS-validated, and neither permits uncertified code to run in the device, then the OS/Common Criteria requirements of FIPS 140–2 do not apply to the OS/application residing in Layer 2/3.* This is justified since OS/applications can not modify binaries—the module CPU has read-only access to code flash—and they can not keep the SSP from resetting the module CPU (therefore, reloading it to a known, good state, initialized from SSP-verified flash).

7.4 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. Segment 2 and 3 code is therefore 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 PCIe host without involving the module CPU, is not enabled in the current FIPS-compliant mode. Segment 0 and 1 under validation does not enable fastpath facilities; the indication requirement for Segments 2 and 3 is applicable[3].

While both the SHA-224 and HMAC variants supported by hardware are all Approved, POST or Miniboot do not currently use SHA-224 or HMAC. Segment 2 and 3 must run their own KATs on SHA-224 and HMAC engines before using them; this is done by the current module CPU driver before first use. (A POST-based HMAC KAT will be included in the first firmware update, but not the currently shipping Segment 1.) Not accessing HMAC engines or SHA-224, delegating this testing to Segment 2 does not impact Miniboot, but we mention it for completeness.

7.5 Determining Mode of Operation

Miniboot uses only approved algorithms and modes of operation. *If the module is functional, and the validated firmware variant is loaded to the validated hardware platform, the module is in FIPS mode.* The "Signed Health Query" (Miniboot 1), in addition to segment ownership and revision number, returns code layers' contents' SHA-256 hashes. Please see p. 24, "Miniboot", for the Segment 1 hash being validated.

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 *Get Health* is platform-dependent. On IBM server platforms, drivers generally provide functions to display segment configurations.

Host drivers are assumed to store query results and make them available to higher-level—application—users. 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 directly.

[3] See host API documentation for platform-specific queries returning fastpath state.

Physical security mechanism	*Severity/Effect*	*Recommended frequency of inspection*	*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 A.S.A.P.

Table 9: Physical security: tamper types and recommended actions

8 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.

8.1 Physical Security Inspection/Testing Recommendations

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

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.

Secrets within a module may not be recovered after a tamper event.

Hard tamper events are caused by very high overvoltage, temperature—or its rate of change—out of reasonable operational range, or physical tamper (penetration of the tamper-detection matrix). *Module memory-type devices—BBRAM, communication FIFOs—are actively zeroized. Module secrets, for practical purposes, are immediately destroyed: BBRAM is actively cleared at microelectronic speeds (sub-milliseconds).* The module becomes permanently inoperative: Miniboot startup does not successfully terminate without secrets in BBRAM.

Hard tamper events may only be detected after the fact by the host application. The module is held in reset after a hard tamper, no further action possible on such a card, as it is held in reset by the internal circuitry until battery removal. 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 PCIe registers. Host code may interrogate and log the reason for the tamper event.

Soft tamper events are caused by moderate overvoltage or temperature moderately out of operational range. Reaction is instantaneous. The module is held under reset while the soft tamper conditions persist. 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 PCIe registers.

External warning indicates that the module has been *removed* from the PCIe slot housing it, but not a hard tamper. (Note that this is not a physical intrusion event, just a logical one.) The corresponding tamper bit is immediately set. Secrets are not destroyed.

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 warning latch state is persistent, and may be cleared through software means.

The original, historical name of external warning state, "intrusion latch", persists in some documentation. We changed it, partially since it sounds threatening, but describes a normal event.

Low battery warning signals when batteries have been drained too low, endangering 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. Alerts trigger at 2.8V.

8.2 Module initialization and delivery

The module is initialized at the factory. Internal controls guarantee that each one may be initialized only once, therefore there are no field initialization requirements, other than platform-specific ones for installation of PCIe cards.

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

8.3 Miscellaneous

Note that the module is very sensitive to environmental conditions. 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 module configuration 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, is public.

Model	*Hardware* (external bus type)	*Physical security*	*Overall*	*Certificate of module*
4758 family				
4758 Model 1	PCI	Level 4	Level 4	Nr. 35
4758 Model 13	PCI	Level 3	Level 3	Nr. 81
4758 Model 2	PCI	Level 4	Level 4	Nr. 116
4758 Model 23	PCI	Level 3	Level 3	Nr. 117
4764 variants				
4764–001	PCI-X	Level 4	Level 4	Nr. 524
4764–001 (updates)	PCI-X	Level 4	Level 4	Nr. 661
4765				
4765	PCI Express (x4)	Level 4	Level 4	—

Table 10: Overview of 47xx product families

9 Predecessors: the 4758 and 4764 families

The first member of the 4758 card family was introduced in 1997. In 1998, the foundational hardware and software received the world's first FIPS 140-1 Security Level 4 validation. Subsequently, the Security Level 3 Model 13 was introduced.

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. These models introduced *outbound authentication*, the capability of a card to authenticate itself to external parties.

4764 variants The 4764, introduced in 2003, is functionally very similar to the Model 2 4758 with enhanced infrastructure capabilities, in terms of performance, enhanced capabilities of its PCIX interface, and RAS features. Several variants of this family exist with mainly hardware variations, and little user-visible firmware differences.

Firmware capabilities of 4764 releases correspond to earlier 4758's; a few additional services have been introduced for hardware. The main functional difference is processor remap, and similar functionality-transparent reorganization.

10 Glossary

CA Certificate Authority, in the Miniboot case, a module in the factory issuing certificates for Miniboot on new cards

CCPs are *card configuration parameters*, security-critical configuration state of a module, which is not confidential. Such critical information includes segment code and ownership (i.e., officer public keys).

EDC Error Detection Code.

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, through *outbound authentication* (OA).

Firmware identifier is an unambiguous status identifier ("Segment 1 hash"), used to quickly summarize firmware contents. It is the SHA-256 hash of firmware contents, possibly including hardware, such as an FPGA bitfile.

Segments are identified by their own segment hashes, but this document only specifies the single applicable firmware Segment 1. Modules loaded with validated Segment 2 and 3 must specify their specific validated configurations.

FWID Abbreviation of *Firmware identifier*

HLM *Hardware Lock Microcontroller*, a dedicated microcontroller which assisted previous 47xx generations with access control and management of persistent storage.

While current generations no longer contain an actual HLM controller, some of the relevant functionality has been retained. Documentation refers to these features as "HLM (infrastructure)" for historical reasons. circuitry, within the "high-speed erase BBRAM" (HSEB).

HSEB High-speed erase BBRAM, a dedicated BBRAM chip actively erased upon tamper. The most valuable Miniboot secrets reside within this region, which is wiped within milliseconds of detecting a tamper event.

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, module-specific hardware, and act as the module security controller.

OA *Outbound Authentication*, infrastructure capable of signing by a card-resident, non-exportable private key.

External parties, including other modules, can verify that signed content has been generated by untampered module firmware (Segment 1). An extension allows OA to manage private keys for OS or applications (Segment 2 or 3).

PCIe PCI Express, the external interface of our module (also abbreviated as PCI-E).

PN *Part Number*

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

RAS Abbreviation of *Reliability, Availability, Serviceability*

SSP *Security Service Processor*, a dedicated processor executing Miniboot and most of POST (i.e., all privileged code).

Segment 1F Segment 1F is the rewritable part of card infrastructure, including the FPGA programming file, and POST 2, all protected as part of Segment 1. Used only when the FPGA bitfile is explicitly mentioned in Segment 1 operations.