Security Policy

2.1. Scope of Document

This document describes the services that the IBM 4758, with IBM Miniboot software resident in ROM and FLASH, provides to a population of security officers, and the security policy governing access to those services.

2.2. Applicable Documents

- the FIPS 140-1 standard, the *Derived Test Requirements*, and on-line implementation guidelines
- DES: FIPS PUB 46-2, FIPS PUB 74, and FIPS PUB 81
- SHA-1: FIPS PUB 180-1
- DSS: FIPS PUB 186
- Pseudorandom Number Generation: Appendix 3 of FIPS PUB 186.
- Optimal Asymmetric Encryption Padding (OAEP), developed by M. Bellare and P. Rogaway and specified in the *Secure Electronic Transaction (SET) Specification Book 2: Programmer's Guide and Book 3: Formal Protocol Definition*
- *Digital Signature Scheme Giving Message Recovery*: ISO/IEC 9796
- the draft 3DES standard, ANSI X9.52, *Triple Data Encryption Algorithm Modes Of Operation*
- the draft RSA standard, ANSI X9.31

2.3. Secure Coprocessor Overview

2.3.1. General Overview

A multi-chip embedded product, the IBM 4758 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 any 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 postal meters, to digital media distribution.

Miniboot Our 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 (such as postal meters) 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 Figure 2.1.

Our current device provides full outgoing authentication for Layer 1 software.

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 need never see application software
- rewritable software can fail, or behave with malice
- 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 IBM4758 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 Figure 2.3), we make it easy for developers to build and deploy secure, FIPS-validatable applications—since they simply have to prepare validation documentation for their additional software, and have it evaluated for secure operation on this module.

Validating our platform at Level 4—the highest level—gives customers the flexibility to design to *any* FIPS level they require.

More Information For more details on the security architecture of the IBM 4758, see:

- S. W. Smith, S. H. Weingart. *Building a High-Performance, Programmable Secure Coprocessor.* Research Report RC21102. IBM T.J. Watson Research Center. February 1998.

(A preprint is available on the IBM Security web site; a revised version will appear in the Elsevier journal *Computer Networks and ISDN Systems* Special Issue on Network Security.)

2.3.2. Architecture and Resources

The IBM 4758 incorporates state-of-the-art hardware security and cryptography technology, including:

- hardware-noise random number generation
- modular exponentiation hardware
- DES hardware
- protective, tamper-respondent membrane
- tamper detection and response circuitry

See Figure 2.2.

Physical Security Our device is tamper-protected for life, from the moment it leaves the factory vault. Physical penetration will change the electrical properties of the membrane; circuitry that is *always* active detects these changes and near-instantly *zeroizes* internal secrets, by explicitly crowbarring the memory devices. The protection circuitry also detects and responds to other environmental attacks, including via radiation, temperature, and voltage.

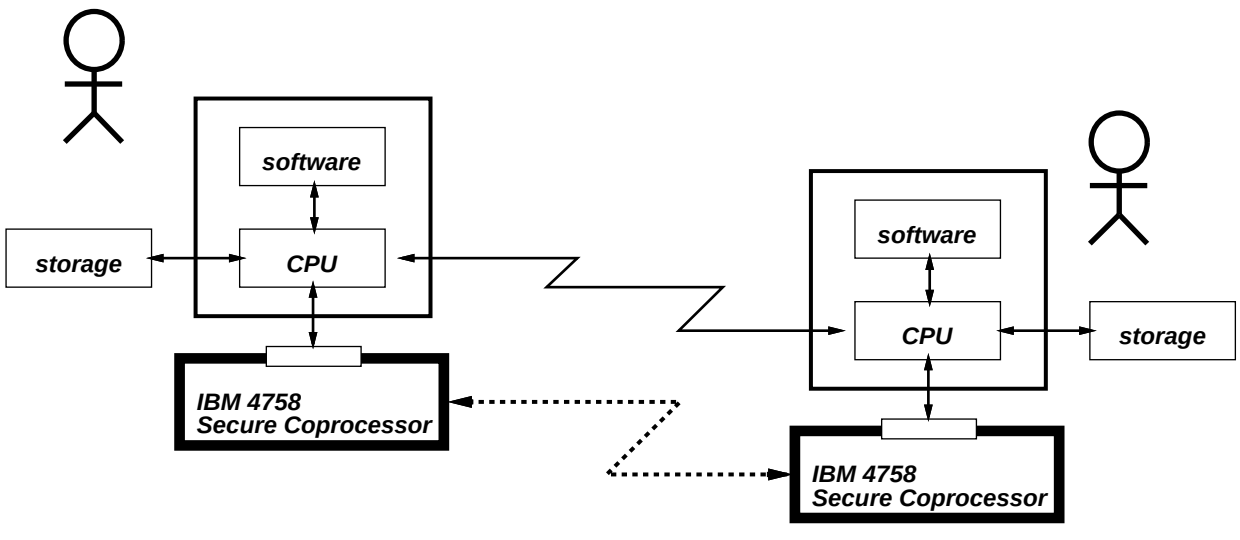

Figure 2.1 Our goal is to enable users, who have never met, to buy our hardware, download software from their chosen security officers, then interact securely—each able to verify that they are talking to the *real thing*, doing the *right thing*.

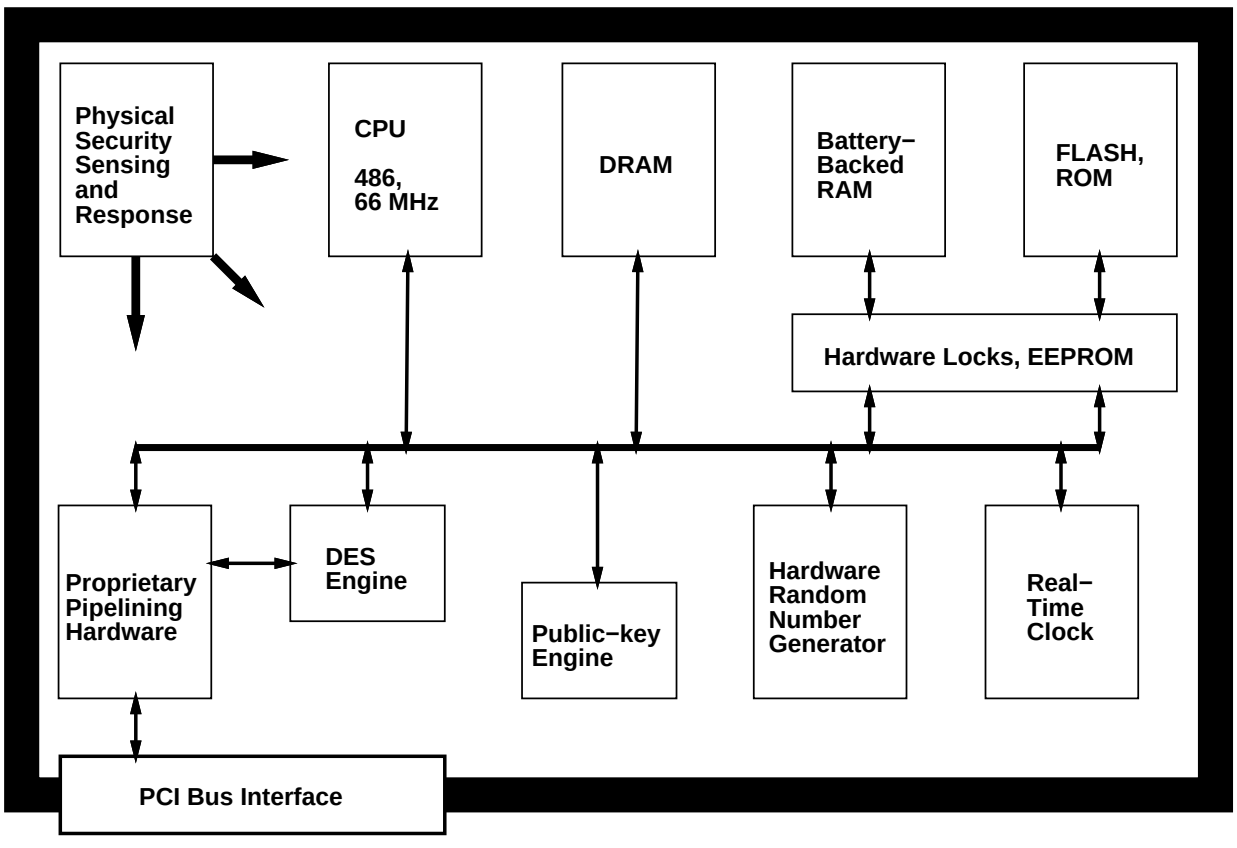

Figure 2.2 IBM4758 hardware architecture.

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

The upper two layers customize the operation of each individual device.

- 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 Figure 2.3.

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.

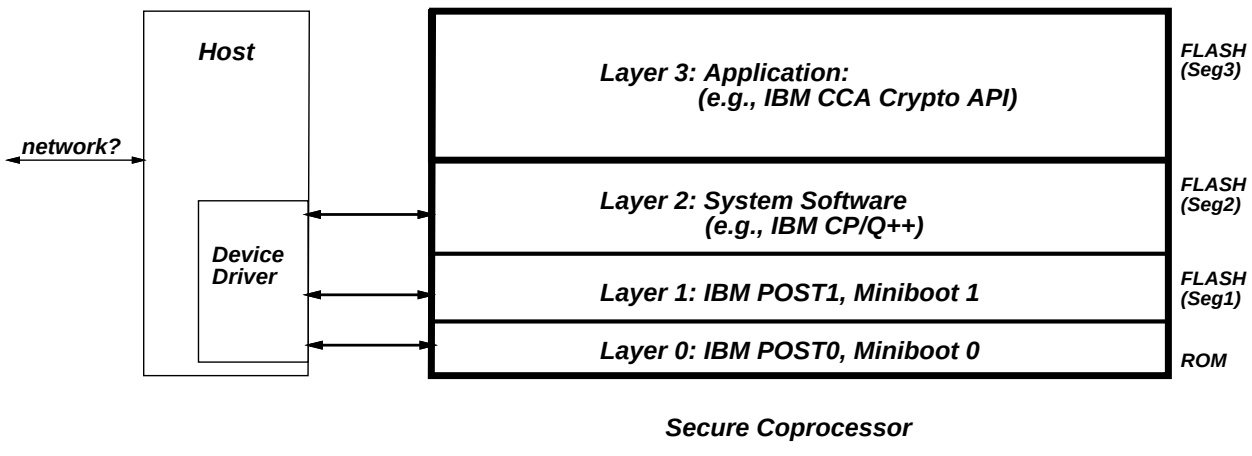

Figure 2.3 IBM4758 software architecture.

[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 repair, 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. Section 12.4 contains more information on how we provide atomicity for all Miniboot operations.

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 *rachet.* 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.* Once this bit is turned off (indicating the device is about to venture from the secure factory into the cruel world), the HLM will never let it be turned on again.

2.3.3. Included Algorithms

The module includes these NIST-approved algorithms:

- DSS
- SHA-1
- DES

The module also includes these algorithms:

- RSA (for signing, and also for encryption)
- 3DES
- OAEP padding for public-key encryption
- ISO9796 padding for public-key signatures
- hardware random number generation

2.4. Cryptographic Module Security Level

This module is intended to be Level 4.

2.5. Roles and Services

2.5.1. 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.

Figure 2.4 through Figure 2.6 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.)

We additionally have a notion of *User*: someone who happens to be communicating with the card wherever it is installed. (See also Section 2.7).

Specific application programs may define other classes of principals.

2.5.2. Services

Our module provides three types of services:

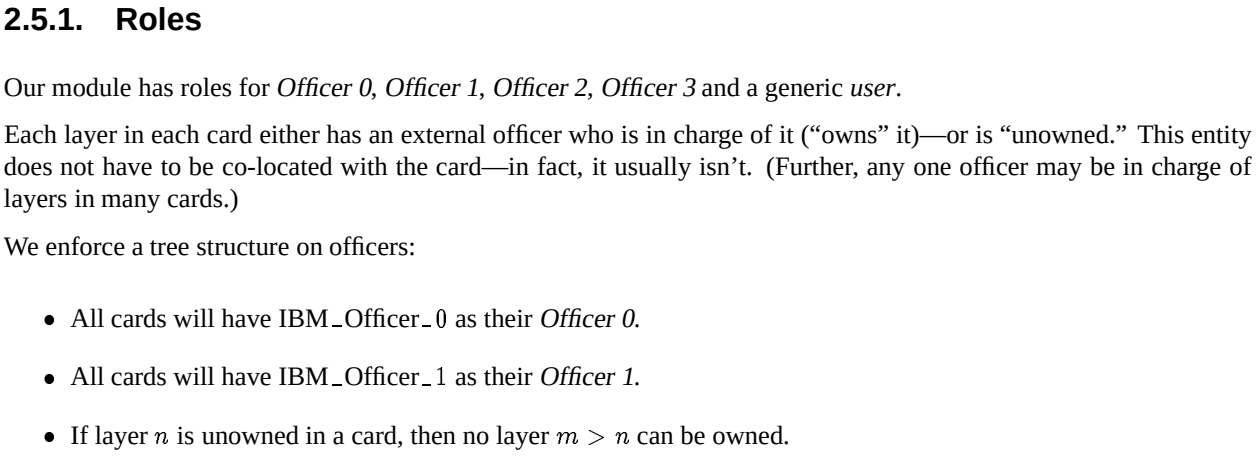

Figure 2.4 Although each device has at most one officer in charge of each layer. The space of *all* officers over *all* devices is organized into a tree. This diagram shows an example hierarchy.

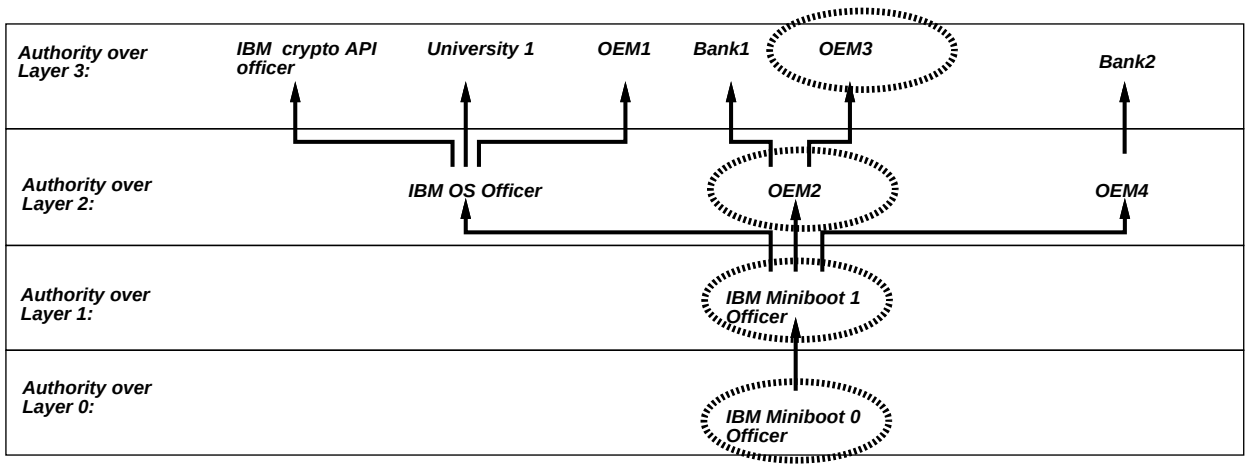

Figure 2.5 Within this example owner hierarchy, one family of devices might have a Layer 2 controlled by “OEM2” and a Layer 3 controlled by “OEM 3.”

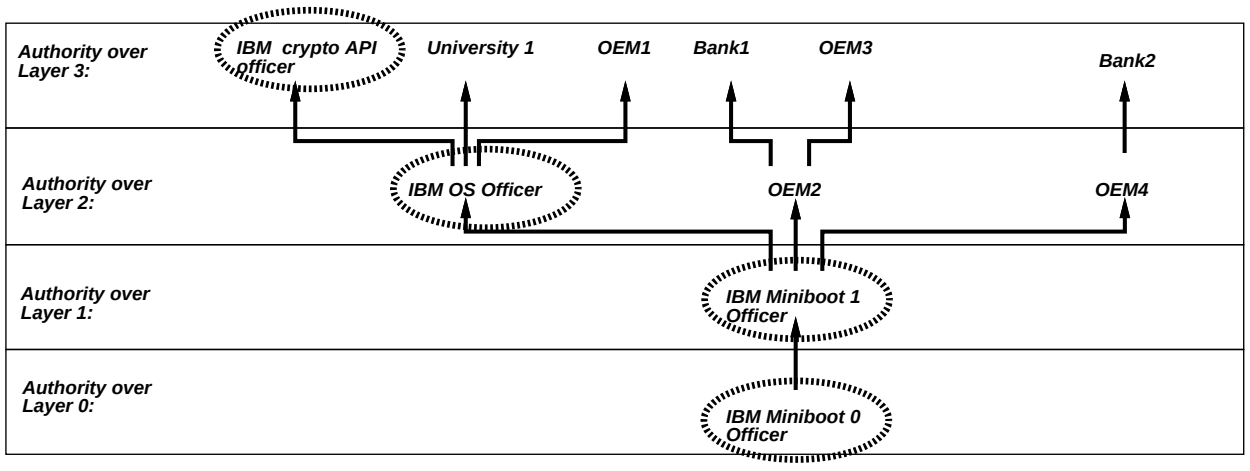

Figure 2.6 Within this example owner hierarchy, another family of devices might have the IBM OS/Control Program in Layer 2 and the IBM crypto API in Layer 3.

- Miniboot *queries*
- Miniboot *commands*
- run-time *BBRAM requests*

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 486 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.

When they are executing, *Program_2* or *Program_3* may present run-time BBRAM queries to the module.

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
- 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.

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 “Add-on Reset” signal in bit 24 in the Bus Master Control/Status Register

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.

2.5.3. 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.

Miniboot authenticates the Officer 0 commands (used for emergency repairs when the device is returned to the IBM factory vault) using secret-key authentication based on DES. Use of any of these commands destroys any other officer secrets that may remain in the device.

2.5.4. SRDI

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

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

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

- the identity of the officer over this layer
- that officer's public key

Since Layer 1 *is* Miniboot 1, its BBRAM state includes the *device private key* that provides the foundation of that untampered device's outgoing authentication ability.

The Layer 0 state includes the DES secrets used for secret-key authentication of Officer 0.

2.5.5. Queries and Commands

Table 2.1 below summarizes the queries and commands that Miniboot offers.

Miniboot 0 Queries Miniboot 0 provides two queries:

- *Query: Status*. This query returns general status information about the card software versions, card identification.
- *Query: Secret Key Authentication Certificate.* This query returns the Secret Key Authentication (SKA) Certificate in Segment 1, if one can be found. (Since the SKA Certificate is necessary for recovering from a damaged Segment 1, it is recommended that users retain a back-up copy off-card.)

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.
- *Emergency Burn 1*. Install a new Program 1 and public key for *Officer 1*, in an untampered card that has gone out into the cruel world but been returned to the factory for repair.
- *Revive*. Resurrect an (allegedly) untampered card that has been zeroized and been returned to the factory for repair. (However, revival of production cards is not part of our current business practice.)
- *Refresh SKA.* Replace the SKA secrets and SKA certificate.

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

Note that all Miniboot 0 commands except "Continue" are carried out within an IBM secure facility.

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 "factory sticky bit" off forever. (This command is rejected if sticky bit is already off.)
- *Field Certify.* Cause an initialized card with no keypair to generate a new device keypair and have it certified.
- *Re-Certify.* Atomically replace the device certificate and empty the transition certificate list. (It's a good idea to verify that you know the public key of the card first!)
- *Establish Owner* n, for $n > 1$. Give an UNOWNED layer n to someone.
- *Surrender Owner* n, for $n > 1$. Give up ownership of Layer n.
- *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.

2.5.6. Run-time BBRAM Requests

Resources The module has two types of BBRAM.

- Lockable BBRAM (*L-BBRAM*) is accessed via the HLM. (We use the term *Page_n* to refer to the layer-n area here. Layer 2's area also consists of a seperate *KeyArea_2*.
- The *RTC-BBRAM* is accessed directly by the 486. (Only Layer 2 and Layer 3 have regions here.) We use the term *Region_n* to refer to the layer-n area here.)

7 As noted earlier, each layer owns a section of BBRAM.

- Layer 0 owns *Page_0*.
- Layer 1 owns *Page_1*.
- Layer 2 owns *Page_2*, the *KeyArea_2*, and *Region_2*.
- Layer 3 owns *Page_3* and *Region_3*.

Service Run-time BBRAM requests consist of HLM requests, and direct access to the *L-BBRAM*. We document the post-bootstrap requests here because our module has two goals:

- If *Program_n* advances the ratchet before crossing a "trust boundary," the private data it stores in *Page_n* will be protected.
- Miniboot will see that *Page_n* and *Region_n* will be cleared when configurations change in specified untrusted ways.

Here are the post-bootstrap *RTC-BBRAM* services:

- *Read Region* n, *Write Region* n, $n \geq 2$.

Here are the post-bootstrap *L-BBRAM* services:

- *Advance Ratchet to* n. Advance the trust ratchet. (Miniboot 1 will advance it to $n = 2$ *before* passing control to Program 2.)
- *Read Page* n, *Read Key Area 2* ($n \geq 2$). Read that page from *L-BBRAM* into *DRAM*.
- *Atomic Write Page* n, *Atomic Write Key Area* 2 ($n \geq 2$). Atomically write data into that page. (Done via an "open-write-commit" sequence of HLM commands.)
- *Atomic Clear Page* n, *Atomic Clear Key Area* 2 ($n \geq 2$). Atomically write zeros into that page.
- *Read EEPROM*. Read public *EEPROM* data.

2.5.7. Roles vs. Services

Table 2.1 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.

Figure 2.7 illustrates how the commands change initialization of the device; Figure 2.8 illustrates how the command change the configuration of Segment 2 and Segment 3.

2.5.8. 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).

Chapter 7 contains more information on the formal statements of these goals, and the formal proof that our system provides them.

			Services	Roles				
				Officer 0 (IBM)	Officer 1 (IBM)	Officer 2	Officer 3	User
QUERIES			Query: SKA Cert	Permitted for anyone who asks.				
			Query: Status					
			Query: Signed Health					
			Query: Certlist					
COMMANDS	Run		Continue to Seg1					
			Continue to Seg2					
	Security		IBM Initialize	Permitted for anyone who asks, while STILL IN FACTORY				
			Field Certify	YES				
			Re−Certify		YES	YES, if privileged	YES, if privileged	
			Revive	YES				
			Refresh SKA	YES				
	Officers		Establish Officer 2		YES			
			Establish Officer 3			YES		
			Surrender Officer 2			YES		
			Surrender Officer 3				YES	
	Code	Burn new program 1, public key for officer 1	At the factory (IBM Burn	Permitted for anyone who asks, while STILL IN FACTORY				
			Ordinary Burn1		YES			
			Emergency Burn1	YES				
		Burn new program 2, public key for officer 2	Ordinary Burn2			YES		
			Emergency Burn2		YES			
		Burn new program 3, public key for officer 3	Ordinary Burn3				YES	
			Emergency Burn3			YES		

Table 2.1 Miniboot command/query policy.

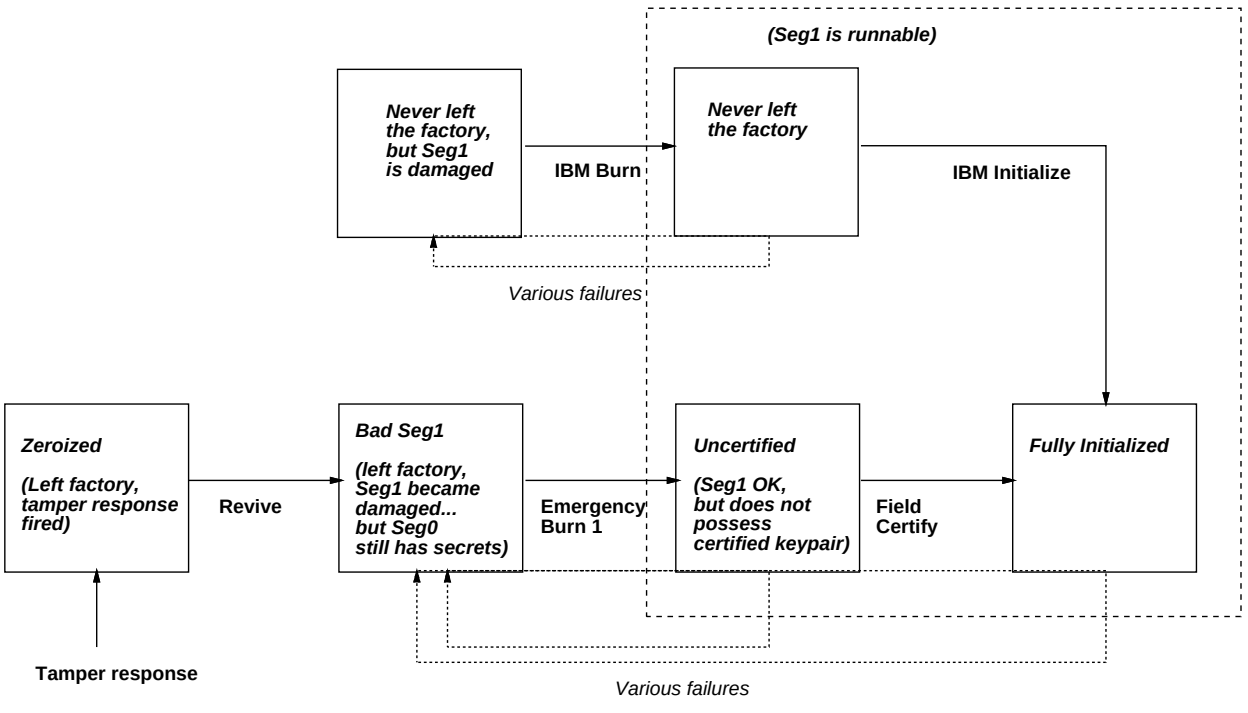

Figure 2.7 A sketch of the configurations and main flows for device initialization and Segment 1.

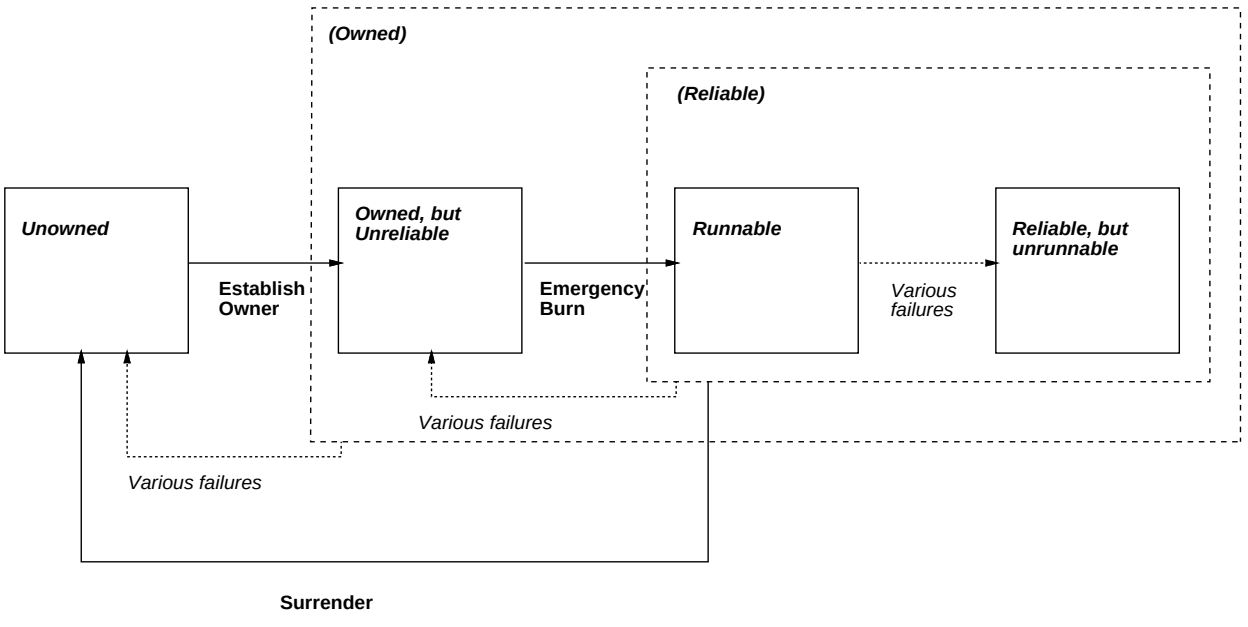

Figure 2.8 A sketch of the configurations and main flows for Segment n, for $n > 1$. Note that, in addition to the transitions shown, one can also “Burn” or “Emergency Burn” from any of the *reliable* states into *Runnable*.

2.6. 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 rewritable software layers to be installed and maintained in a hostile field, without the use of trusted couriers or on-site security officers. See Figure 2.9.

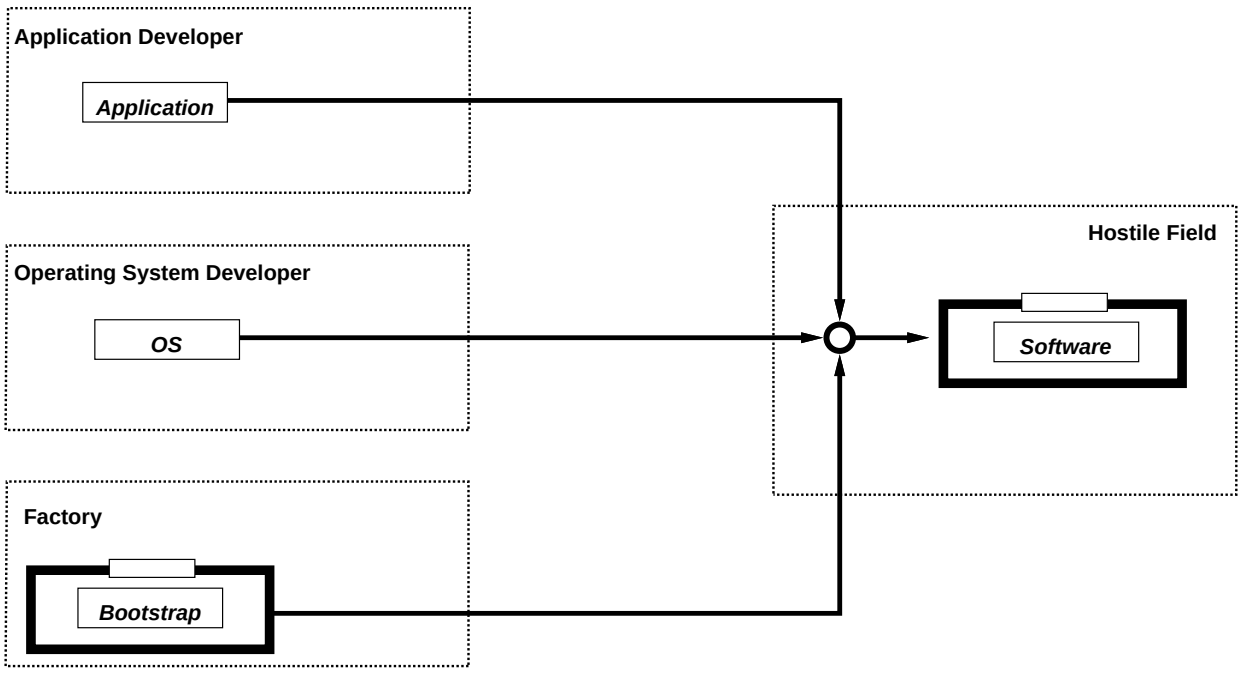

Figure 2.9 The IBM4758 supports three layers of rewritable software, from potentially mutually suspicious developers, configurable in a hostile field location, with neither trusted courier nor on-site security officer.

2.7. 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 140-1 validation:

- Officer 0 relates to the “Cryptographic Officer 0” role. (This officer operates only in the secure factory)
- Officer 1 through Officer 3 relate to the “User 1” through “User 3” roles, respectively. (These entities use the validated software to control, install, and maintain the configuration of the device, once it’s shipped.)

(Recall also the Miniboot SRDI discussion in Section 2.5.4.)

2.8. Module Configuration for FIPS 140-1 Compliance

2.8.1. Miniboot

To be a FIPS-compliant module, the device must be loaded with Version 3 or later of Miniboot 1. Official IBM Seg1 codeloads with revision 1.3.1 or later will contain this version.

To then operate in a FIPS-complaint mode, each officer must choose DSS for their public keys (until such time as RSA is accepted as a NIST-approved algorithm).

2.8.2. 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 SRDI[2] that Miniboot depends on (in protected FLASH and the Miniboot region of lockable BBRAM) cannot be compromised by Layer 2 or Layer 3. Miniboot will still run securely the next time the device is reset.

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 unvalidated code to run in the device, then the operating system/Orange Book requirements of FIPS 140-1 will not apply to the OS/control program residing in Layer 2.

2.8.3. Determining Mode of Operation

The "Signed Heath Query" to Miniboot 1 will return the identities and revisions of each layer's programs, and the signature-algorithm chosen by each officer.

[2] *Security Relevant Data Item (SRDI)* is a term used in the FIPS 140-1 Derived Test Requirements.