



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



Institut für  
Technische Informatik und  
Kommunikationsnetze

# Best Practices to Securely Operate Hardware Security Modules in a High Availability Setup

Benedikt Köppel

Semester Project  
October 2012 until January 2013  
Advisors: Dr. Stephan Neuhaus and Michael Zeier  
Supervisor: Prof. Dr. Bernhard Plattner

## **Acknowledgements**

I would like to express my gratitude to my supervisors Dr. Stephan Neuhaus and Michael Zeier, as well as SafeNet and my employer for providing the Hardware Security Modules and answering our support questions.

## **Abstract**

The latest generation of SafeNet Hardware Security Modules have specific features for high availability and fault tolerance built in. These HSMs are certified by the FIPS PUB 140-2 standard. In this term project, we evaluated the high availability implementation of the SafeNet HSMs. We have conducted interviews with users, support staff and risk officers to determine requirements for such a high availability setup, and ran functional, performance and security tests on the HSM cluster. We identified functional, performance and security problems which directly arise from the client implementation of the high availability features. Based on our findings from the tests and interviews, we provide a best practices guide for the setup and maintenance of such security modules in a high availability cluster.



# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>13</b> |
| <b>2</b> | <b>Related Work</b>   | <b>15</b> |
| 2.1      | FIPS PUB 140-2 . . . . .  | 15        |
| 2.2      | Operating Policy . . . . .  | 16        |
| 2.3      | Audit and Backup Procedures for Hardware Security Modules . . . . . | 16        |
| <b>3</b> | <b>Methodology</b>  | <b>17</b> |
| 3.1      | HSM Introduction . . . . .  | 17        |
| 3.2      | HSM Setup . . . . .   | 18        |
| 3.2.1    | Base Setup . . . . .  | 18        |
| 3.2.2    | Stand-Alone Hardware Security Module . . . . .                      | 19        |
| 3.2.3    | High Availability Cluster . . . . .                                 | 20        |
| 3.3      | Interviews . . . . .  | 21        |
| 3.4      | Software Testing . . . . .  | 22        |
| 3.4.1    | Key Distribution . . . . .  | 22        |
| 3.4.2    | Failure Detection . . . . .   | 22        |
| 3.4.3    | Failover . . . . .  | 23        |
| 3.4.4    | Recovery . . . . .  | 23        |
| 3.4.5    | Changing Object Attributes . . . . .                                | 23        |
| 3.4.6    | Creating Objects in a Split Brain Situation . . . . .               | 24        |
| 3.4.7    | Deleting Objects in a Split Brain Situation . . . . .               | 25        |
| 3.4.8    | Performance Scalability (Multiple Threads) . . . . .                | 25        |
| 3.4.9    | Performance Scalability (Multiple Processes) . . . . .              | 26        |
| 3.4.10   | Performance Scalability (High Availability Overhead) . . . . .      | 26        |
| 3.4.11   | Performance Impact of a Slow HSM . . . . .                          | 26        |
| 3.4.12   | Different Red Cloning iKeys in a HA Group . . . . .                 | 27        |
| 3.4.13   | Different Black Partition iKeys in a HA Group . . . . .             | 27        |
| 3.5      | Threat Model . . . . .  | 28        |
| <b>4</b> | <b>Results</b>  | <b>29</b> |
| 4.1      | Requirements Gathered from the Interviews . . . . .                 | 29        |
| 4.2      | HSM Functionality . . . . .   | 30        |
| 4.2.1    | High Availability . . . . .   | 30        |
| 4.2.2    | Key Exchange . . . . .  | 30        |
| 4.2.3    | Load Balancing . . . . .  | 31        |
| 4.2.4    | Failure Detection . . . . .   | 31        |
| 4.2.5    | Recovery . . . . .  | 31        |
| 4.3      | Testing Results . . . . .   | 31        |
| 4.3.1    | Successful Functional Tests . . . . .                               | 31        |
| 4.3.2    | Unsuccessful Functional Tests . . . . .                             | 32        |
| 4.3.3    | Performance Tests . . . . .   | 33        |

---

|          |  |           |
|----------|--|-----------|
| 4.3.4    | Security Tests . . . . .                   | 34        |
| 4.4      | Other Insights . . . . .                   | 36        |
| 4.4.1    | vtl Slot Information Discrepancy . . . . . | 36        |
| 4.4.2    | Clearing Partitions . . . . .              | 36        |
| <b>5</b> | <b>Discussion</b>                          | <b>39</b> |
| 5.1      | High Availability Implementation . . . . . | 39        |
| 5.1.1    | Functionality . . . . .                    | 39        |
| 5.1.2    | Performance . . . . .                      | 40        |
| 5.1.3    | Security . . . . .                         | 42        |
| 5.1.4    | Possible Improvements . . . . .            | 42        |
| 5.2      | Operating Policy . . . . .                 | 43        |
| 5.2.1    | Setup Guidelines . . . . .                 | 43        |
| 5.2.2    | Application Recommendations . . . . .      | 45        |
| 5.2.3    | Requirements and Tradeoffs . . . . .       | 45        |
| <b>6</b> | <b>Conclusion</b>                          | <b>49</b> |
|          | <b>References</b>                          | <b>51</b> |

# List of Figures

|     |   |    |
|-----|---|----|
| 4.1 | Performance Scalability (High Availability HSM) . . . . . | 38 |
| 4.2 | Performance Scalability (Stand-alone HSM) . . . . .       | 38 |





# List of Tables

|     |   |    |
|-----|---|----|
| 3.1 | HSM Network and Partition Settings . . . . .                                  | 18 |
| 4.1 | Performance Scalability (Number of Accessing Threads) . . . . .               | 33 |
| 4.2 | Performance Scalability (Number of Accessing Threads and Processes) . . . . . | 34 |
| 4.3 | Performance Scalability (HA Overhead) . . . . .                               | 34 |
| 4.4 | Performance Scalability (HA Overhead, Multiple Threads) . . . . .             | 35 |
| 4.5 | Performance Impact of a Slow HSM . . . . .                                    | 35 |
| 5.1 | Example HSM Journal . . . . .   | 46 |
| 5.2 | High Availability Log Messages . . . . .                                      | 46 |



# Listings

|     |   |    |
|-----|---|----|
| 3.1 | Base Setup of a HSM . . . . .   | 19 |
| 3.2 | Setup Instructions for One HSM . . . . .                                    | 19 |
| 3.3 | Setup Instructions for Multiple HSMs with High Availability . . . . .       | 20 |
| 4.1 | High Availability Recovery Attempts in the Logfile . . . . .                | 32 |
| 4.2 | Manually Synchronizing a Cluster after Changing Object Attributes . . . . . | 32 |
| 4.3 | vtl Slot Information Discrepancy . . . . .                                  | 36 |
| 4.4 | Clearing a High Availability Partition . . . . .                            | 36 |



# Chapter 1

## Introduction

Hardware Security Modules (HSM) are specialized cryptographic processors. They provide cryptographic functions and software to manage cryptographic keys. A HSM offers these functions as crypto co-processor to a host system. The module is built in such a way that attempts to tamper with it will destroy all stored cryptographic material [1].

In situations that require the use of highly sensitive or exposed cryptographic keys, such Hardware Security Modules are used to generate and store keys, enable ciphering and authentication and digitally sign documents in a secure fashion.

Such cryptographic modules are certified according to the FIPS PUB 140-2 security standard [2]. In addition to FIPS PUB 140-2, the operator of a HSM is ultimately responsible for the security and has to acknowledge the residual risk. The security of such HSM modules in a stand-alone usage case was studied before ([3]). Procedures and best practice ceremonies to securely operate Hardware Security Modules was described in [1].

At the moment, the author's employer is replacing their Hardware Security Modules by a new generation. The new modules will allow users to operate multiple HSMs in a clustered setup for high availability. In this project, the high availability features of the new generation of Hardware Security Modules were evaluated and tested for security, operability and performance. The test plans we use are described in Chapter 3. We gathered the requirements from business users, support staff and the risk officer. The requirements and testing results are documented in Chapter 4. In Chapter 5, we discuss the testing results and suggest an operating policy to cater to the high availability functionality.

For this work, we have used the SafeNet Luna SA 5 Hardware Security Modules. SafeNet has provided us with answers to our support queries, but not influenced this work. The author's employer has provided us with the Hardware Security Modules. I was not paid or employed to work on this project or write this report.



## Chapter 2

# Related Work

### 2.1 FIPS PUB 140-2

Security Requirements for Cryptographic Modules (FIPS PUB 140-2) [2] is a security standard published by the National Institute of Standards and Technology (NIST). The publication specifies security elements of Hardware Security Modules. There are four levels of security with increasing security requirements. HSMs which are used by the US Federal agencies and departments have to fulfil the elements defined in the FIPS PUB 140-2 security standard. The Hardware Security Modules used by the author's employer are certified by FIPS PUB 140-2. The modules are validated by the Cryptographic Module Validation Program (CMVP)<sup>1</sup>.

The validation requires a security policy which documents various components of the cryptographic module. For the SafeNet Luna SA 5 HSMs, this documentation can be found at [4]. The precise commands and instructions how to use these features are described in SafeNet's technical guide [5]. For the high availability features that we use in this project, the following parts of the policy are especially relevant:

**Partition Capabilities:** The security officer can allow or disallow certain capabilities for each logical storage (partition) within the HSM. The following capabilities are relevant for our purpose:

- Allow/disallow high availability
- Allow/disallow private and secret key cloning
- Allow/disallow network replication

**Identification and Authentication:** The authentication tokens for the SafeNet Luna HSM are stored on USB tokens, so called iKeys. Different colors are used to identify the capabilities of the tokens. The red iKey is used to store the tokens used for backup and cloning of cryptographic keys. The red iKey is thus also responsible for the cloning features used in the network replication and high availability setup:

«Red (Domain) iKey – for the storage of the cloning domain data, used to control the ability to clone from a cryptographic module to a backup token» (Level 3 Non-Proprietary Security Policy For Luna@PCI-e Cryptographic Module [4], 2011, p. 16)

---

<sup>1</sup><http://www.nist.gov/cmvp>

In FIPS PUB 140-2, there are no specifications for Hardware Security Modules operated in a high availability cluster. In our work, we want to see whether the standardization of a stand-alone HSM is sufficient to guarantee the security of HSMs in a high availability setup, or whether there should be any additional specifications for highly available HSM clusters.

## 2.2 Operating Policy

FIPS PUB 140-2 and the corresponding security policy issued by the vendor itself can not guarantee any security, if the administrators of the Hardware Security Module are not trustworthy. The author's employer has written an additional guide which extends the official publications and describes verifiable ceremonies as explained in [6]. This operating policy guide consists of the following sections:

- Use cases of the HSM
- Key owners and their responsibilities
- Setup instructions for new HSMs
- Storage, transport and decommissioning of a HSM
- Business continuity plan and backup strategies
- Maintenance tasks

As part of this work, the operating policy was extended by the section «High Availability HSM Instructions».

## 2.3 Audit and Backup Procedures for Hardware Security Modules

OpenHSM [3] is an open source implementation of a Hardware Security Module, aimed at the use in universities. In Audit and Backup Procedures for Hardware Security Modules [1], the authors described algorithms implemented in OpenHSM. In particular, the backup schemes are described in great detail in Chapter 5.2 and following. These procedures could also be used in a high availability setup, where keys need to be exchanged between HSMs regularly. However, the authors of [1] assume that the backup can be manually inspected by an auditor group before the backup is restored. This is required to make sure that the backup file is in fact from a legitimate HSM and not a cryptographic container created elsewhere.



## Chapter 3

# Methodology

### 3.1 HSM Introduction

The SafeNet Luna SA 5 HSM uses a few technical concepts that we would like to explain before going into the actual HSM setup and test cases. These concepts are also documented in the SafeNet Luna SA Help System [5].

**Hardware Security Module Card:** The actual Hardware Security Module is a PCIe card, consisting of a cryptographic processor and secured memory for the key storage.

**Hardware Security Module Appliance:** The PCIe crypto-card is built into a server and then shielded with additional tamper detection. The server appliance can be managed using a VT100 serial connection or through SSH. We use the VT100 serial terminal to manage the HSM in our work.

**iKeys:** The HSM user and operators are identified using physical key tokens. SafeNet calls those tokens iKeys. Each token allows the user to carry out specific operations. iKeys are labelled with colors and allow the following tasks: Access to the content of a partition (Partition Owner, black iKey), configure, start and restore backups and network replication (Cloning Domain Key Owner, red iKey) and general HSM administration (Security Officer, blue iKey). The tokens are also used to encrypt objects for backup and during replication.

**Partition:** The secure HSM storage is divided into logical sections, so called partitions. Each partition can hold a certain number of keys. Before a partition can be accessed, the partition has to be activated. If auto-activation is enabled, the partition becomes activated automatically after a reboot of the HSM. To access the partition, its black iKey and password have to be used.

**Client Library:** The `libCryptoki2.so` library offers a PKCS#11 interface to the client application. The library presents the HSM as a PKCS#11 slot, takes care of setting up the NTLs link, and translates PKCS#11 calls from the application to the HSM. The client library is configured using the `Chrystoki.conf` configuration file.

**Network Trust Link:** The Network Trust Link (called NTLs by SafeNet), is an encrypted tunnel between the client server and the Hardware Security Module appliance. The NTLs link is encrypted using X509 certificates. In the initial setup of the HSM, we have to generate certificates for the HSM and the client, exchange the certificates and configure the `libCryptoki2.so` library

to use those in the NTLS connection. When a client application loads the `libCryptoki2.so` library and specifies a particular PKCS#11 slot, the library initiates the NTLS connection to the HSM.

To manage the HSM and conduct our tests, we use the PKCS#11 command line tools provided by SafeNet:

**vtl:** The Virtual Token Library is used to manage the PKCS#11 client configuration. It configures the high availability group and displays the HA status.

**cmu:** The Certificate Management Utility provides functions to generate, import and export asymmetric key pairs and certificates on the HSM. We use it to generate objects and test the synchronization.

**multitoken2:** Multitoken is the demo and test tool to perform basic cryptographic functions on the HSM. The tool measures the number of operations performed per second. We use `multitoken2` to measure the performance of the HA cluster. Using the `-nslots` parameter, we can configure how many threads access a PKCS#11 slot. `-nslots 1x12,2x12` starts 12 threads accessing slot 1, and 12 threads accessing slot 2 in parallel.

**ckdemo:** This interactive demo tool is also provided by SafeNet. The tool allows us to create, list, modify and delete objects on the HA cluster.

## 3.2 HSM Setup

For the setup, we assume the settings in Table 3.1. `testhsm` is used in the setup with just one HSM, `hsmone`, `hsmtwo` and `hsmthree` are used for the high availability setup.

| hostname              | network      |             | name                | partition          |                     |
|-----------------------|--------------|-------------|---------------------|--------------------|---------------------|
|                       | address      | gateway     |                     | password           | serial <sup>a</sup> |
| <code>testhsm</code>  | 192.168.1.20 | 192.168.1.1 | <code>myPart</code> | <code>myPwd</code> | 65005000            |
| <code>hsmone</code>   | 10.0.1.21    | 10.0.1.1    | <code>haPart</code> | <code>haPwd</code> | 65005001            |
| <code>hsmtwo</code>   | 10.0.1.22    | 10.0.1.1    | <code>haPart</code> | <code>haPwd</code> | 65005002            |
| <code>hsmthree</code> | 10.0.1.23    | 10.0.1.1    | <code>haPart</code> | <code>haPwd</code> | 65005003            |

<sup>a</sup> The partition serial numbers are automatically generated by the HSM.

**Table 3.1:** HSM Network and Partition Settings

The following setup instructions are covered in the documentation provided by SafeNet [5] and are partly outlined in [7]. In the listings, we use `host:~` to identify commands which have to be executed on the application server, and `testhsm:>` respectively `hsmone:>`, `hsmtwo:>` and `hsmthree:>` for commands which are run on the HSMs.

### 3.2.1 Base Setup

For the basic setup of the HSMs, we follow the instructions from the technical documentation [5]. The steps are covered in Listing 3.1 and have to be executed in the VT100 emulator directly on the module. These instructions are exemplary for `testhsm`. To set up the other HSMs, the parameters have to be changed accordingly to Table 3.1.

First, the HSM is reset to its factory default (Line 1). Then, the time and network settings are configured (Line 3 to 9). For the initialization of the HSM (Line 10),

the Pin Entry Device (PED) is required. In this step, the cryptographic tokens for the security officer and the cloning domain are generated. The HSM wide policies are set to allow cloning (Line 13), allow network replication (Line 14) and remote authentication (Line 15). Then, the Network Trust Link connection (NTLS) is set up (Line 17 to 19). The application and HSM will communicate through this NTLS SSL connection. The fingerprint of the server certificate (Line 20) will be used later to identify the server during the client setup.

```

1  testhsm:> hsm factoryReset
2  testhsm:> sysconf timezone set Europe/Zurich
3  testhsm:> sysconf time <HH:MM> <YYYYMMDD>
4  testhsm:> status time
5  testhsm:> net interface -device eth0 -ip 192.168.1.20 -netmask \
    255.255.0.0 -gateway 192.168.1.1
6  testhsm:> net hostname testhsm
7  testhsm:> net domain dummy.example.com
8  testhsm:> network show
9  testhsm:> status interface
10 testhsm:> hsm init -label testhsm
11 testhsm:> hsm login
12 testhsm:> hsm showPolicies -configonly
13 testhsm:> hsm changePolicy -policy 7 -value 1
14 testhsm:> hsm changePolicy -policy 16 -value 1
15 testhsm:> hsm changePolicy -policy 20 -value 1
16 testhsm:> hsm showPolicies -configonly
17 testhsm:> sysconf regenCert
18 testhsm:> ntl bind eth0
19 testhsm:> ntl show
20 testhsm:> sysconf fingerprint ntl

```

**Listing 3.1:** Base Setup of a HSM

### 3.2.2 Stand-Alone Hardware Security Module

Once the HSM is initialized (Listing 3.1), we configure one HSM to be used stand-alone. First we create a new partition to store the cryptographic objects (Line 2). Take a note of the challenge string and change it to `myPwD` in the next step. Once the password of the partition was changed, we can allow the activation (Line 5) and auto activation (Line 6) policies, and then activate the partition (Line 7). Auto activation ensures that the partition will become active after the reboot of the HSM automatically. Before we can use the NTLS connection, the certificate of the HSM has to be downloaded to the client machine and imported to the trusted HSM certificates (Line 8 and Line 10). Make sure that the fingerprint from Line 20 of Listing 3.1 matches with the fingerprint of the downloaded certificate (Line 9). Next, a certificate for the client is created (Line 11), uploaded to the HSM (Line 12) and then registered in the HSM (Line 14). On Line 15, the HSM storage `myPart` is assigned to the new client. Finally, the setup is verified on Line 16.

```

1  testhsm:> hsm login
2  testhsm:> partition create -partition myPart
3  testhsm:> partition changePw -partition myPart -oldpw \
    eyK4-dbX5-QNC8-M9nB -newpw myPwD
4  testhsm:> partition showPolicies -partition myPart -configonly
5  testhsm:> partition changePolicy -partition myPart -policy 22 -value 1
6  testhsm:> partition changePolicy -partition myPart -policy 23 -value 1
7  testhsm:> partition activate -partition myPart -password myPwD
8  host:~ scp admin@192.168.1.20:server.pem ./cert/server/
9  host:~ openssl x509 -noout -fingerprint -in ./cert/server/server.pem
10 host:~ vtl addServer -n testhsm -c ./cert/server/server.pem
11 host:~ vtl createCert -n 192.168.1.10
12 host:~ scp ./cert/client/192.168.1.10.pem admin@192.168.1.20:

```

```

13 host:~ ssh admin@192.168.1.20
14 testhsm:> client register -client 192.168.1.10 -hostname 192.168.1.10
15 testhsm:> client assignPartition -client 192.168.1.10 -partition myPart
16 host:~ vtl verify

```

**Listing 3.2:** Setup Instructions for One HSM

### 3.2.3 High Availability Cluster

Before multiple HSMs can be set up as high availability cluster, we have to do the base setup as described in Listing 3.1, by replacing the parameters with those given in Table 3.1. Because the individual HSMs in the high availability setup will have to exchange keys, they must be set up using the same red domain iKey (on Line 10 of Listing 3.1). During the setup, the administrator is asked to reuse an existing iKey as opposed to overwrite it.

For the high availability setup itself, we follow the instructions in Listing 3.3. For our tests we have used three identical SafeNet Luna SA 5 HSMs.

On the host system, a client certificate is generated (Line 2) and then uploaded to all HSMs (Line 3 to 5). Then the host certificate of each HSM is downloaded and installed (Line 6 to 14). On each HSM, the partition is created, activated and the client is registered to it (Line 19 to 28). It is important to take a note of the partition serial numbers, as they will be used later when the HA group is configured. The private (Line 22) and secret key cloning (Line 23) as well as the high availability recovery policies (Line 24) have to be enabled. When the partitions are created, each partition has to have the same red cloning domain iKey and the same partition password.

Now we can register the partitions of each HSM into one high availability group. First, a new HA Group is created (Line 62) with just the partition of the first HSM (the partition with the serial number 65005001). The `newGroup` command will auto-generate a group serial number, for example 74007601. This group serial number will be used in the following steps to add the other partitions to the group (Line 63 and 64). With the `show` command, the HA Group slot number should be printed out (Line 69). The HA group can now be accessed through the PKCS#11 library at the given slot number.

```

1 # Host
2 host:~ vtl createCert -n 192.168.1.10
3 host:~ scp ./cert/client/192.168.1.10.pem admin@10.0.1.21:
4 host:~ scp ./cert/client/192.168.1.10.pem admin@10.0.1.22:
5 host:~ scp ./cert/client/192.168.1.10.pem admin@10.0.1.23:
6 host:~ scp admin@10.0.1.21:server.pem ./cert/server/10.0.1.21.pem
7 host:~ scp admin@10.0.1.22:server.pem ./cert/server/10.0.1.22.pem
8 host:~ scp admin@10.0.1.23:server.pem ./cert/server/10.0.1.23.pem
9 host:~ openssl x509 -noout -fingerprint -in ./cert/server/10.0.1.21.pem
10 host:~ openssl x509 -noout -fingerprint -in ./cert/server/10.0.1.22.pem
11 host:~ openssl x509 -noout -fingerprint -in ./cert/server/10.0.1.23.pem
12 host:~ vtl addServer -n hsmone -c ./cert/server/10.0.1.21.pem
13 host:~ vtl addServer -n hsmtwo -c ./cert/server/10.0.1.22.pem
14 host:~ vtl addServer -n hsmthree -c ./cert/server/10.0.1.23.pem
15
16 # HSM one
17 host:~ ssh admin@10.0.1.21
18 hsmone:> hsm login
19 hsmone:> partition create -partition haPart
20 hsmone:> partition changePw -partition myPart -oldpw \
    eyK4-dbX5-QNC8-M9nB -newpw haPwd
21 hsmone:> partition showPolicies -partition haPart
22 hsmone:> partition changePolicy -partition haPart -policy 0 -value 1
23 hsmone:> partition changePolicy -partition haPart -policy 4 -value 1

```

```

24 hsmone:> partition changePolicy -partition haPart -policy 21 -value 1
25 hsmone:> partition changePolicy -partition haPart -policy 22 -value 1
26 hsmone:> partition changePolicy -partition haPart -policy 23 -value 1
27 hsmone:> partition activate -partition haPart -password myPwd
28 hsmone:> client register -client 192.168.1.10 -hostname 192.168.1.10
29 hsmone:> client assignPartition -client 192.168.1.10 -partition haPart
30
31 # HSM two
32 host:~ ssh admin@10.0.1.22
33 hsmtwo:> hsm login
34 hsmtwo:> partition create -partition haPart
35 hsmtwo:> partition changePw -partition myPart -oldpw \
    eyK4-dbX5-QNC8-M9nB -newpw haPwd
36 hsmtwo:> partition showPolicies -partition haPart
37 hsmtwo:> partition changePolicy -partition haPart -policy 0 -value 1
38 hsmtwo:> partition changePolicy -partition haPart -policy 4 -value 1
39 hsmtwo:> partition changePolicy -partition haPart -policy 21 -value 1
40 hsmtwo:> partition changePolicy -partition haPart -policy 22 -value 1
41 hsmtwo:> partition changePolicy -partition haPart -policy 23 -value 1
42 hsmtwo:> partition activate -partition haPart -password myPwd
43 hsmtwo:> client register -client 192.168.1.10 -hostname 192.168.1.10
44 hsmtwo:> client assignPartition -client 192.168.1.10 -partition haPart
45
46 # HSM three
47 host:~ ssh admin@10.0.1.23
48 hsmthree:> hsm login
49 hsmthree:> partition create -partition haPart
50 hsmthree:> partition changePw -partition myPart -oldpw \
    eyK4-dbX5-QNC8-M9nB -newpw haPwd
51 hsmthree:> partition showPolicies -partition haPart
52 hsmthree:> partition changePolicy -partition haPart -policy 0 -value 1
53 hsmthree:> partition changePolicy -partition haPart -policy 4 -value 1
54 hsmthree:> partition changePolicy -partition haPart -policy 21 -value 1
55 hsmthree:> partition changePolicy -partition haPart -policy 22 -value 1
56 hsmthree:> partition changePolicy -partition haPart -policy 23 -value 1
57 hsmthree:> partition activate -partition haPart -password myPwd
58 hsmthree:> client register -client 192.168.1.10 -hostname 192.168.1.10
59 hsmthree:> client assignPartition -client 192.168.1.10 -partition haPart
60
61 # Host
62 host:~ vtl haAdmin -newGroup -serialNum 65005001 -label haGroup \
    -password haPwd
63 host:~ vtl haAdmin -addMember -group 74007601 -serialNum 65005002 \
    -password haPwd
64 host:~ vtl haAdmin -addMember -group 74007601 -serialNum 65005003 \
    -password haPwd
65 host:~ vtl verify
66 host:~ vtl haAdmin -HAOnly -enable
67 host:~ vtl haAdmin -HALog -enable
68 host:~ vtl haAdmin -autoRecovery -interval 60 -retry -1
69 host:~ vtl haAdmin -show -syncStatus

```

**Listing 3.3:** Setup Instructions for Multiple HSMs with High Availability

### 3.3 Interviews

While the above steps were given by the SafeNet documentation, the whole key ceremonies and failover procedures for the high availability setup were discussed with the appropriate teams in the company. Three profiles were included in the interviews: The Crypto User, the Support Analyst and the Risk Officer. The interviews were conducted in an unstructured, non-directive manner. During the interviews,

the following topics were covered: Setup, maintenance and decommissioning of a HSM; key generation, import and export; and performance requirements.

- The Crypto User, typically member of an application management team, uses the functionality of the HSM in his applications. They are required to use a HSM because their applications uses sensitive key material, for example an E-Banking web server.
- The Support Analyst provides support to the Crypto User in the usage and maintenance of their HSMs.
- The Risk Officer is responsible for the security of the cryptographic material stored in the HSM. His task is to set the guidelines and requirements from a risk perspective.

## 3.4 Software Testing

To understand the failover and recovery mechanisms, the key exchanges and performance of the high availability cluster, we have designed the following tests.

### 3.4.1 Key Distribution

**Test Goal:** Is a new key automatically distributed to all HSMs?

**Test Steps:**

1. Using the `cmu` command, create a new object on the HSM cluster:  
`cmu gen -modulusBits=2048 -publicExp=65537 -sign=T -verify=T -slot=1 -password=haPwd.`
2. Connect to `hsmone` directly and display the newly created object with `partition showContents -partition haPart -password haPwd.`
3. Repeat step 2 on `hsmtwo` (partition `haPart`) and `hsmthree` (partition `haPart`).

**Expected Result:** The object should immediately exist on all three HSMs.

### 3.4.2 Failure Detection

**Test Goal:** Can the client detect if a HSM has failed?

**Test Steps:**

1. Run `vtl haAdmin -show -syncStatus` and make sure the cluster is fully operational and synchronized.
2. Remove one HSM from the HA cluster by plugging the network cable out or disabling the NTLS service (`service stop ntl`s).
3. Run `vtl haAdmin -show -syncStatus` again.

**Expected Result:** The `vtl` command should inform about the failure of one HSM.

### 3.4.3 Failover

**Test Goal:** Can a running application continue, when one HSM fails during the operation?

**Test Steps:**

1. Using the `multitoken2` tool, start a batch of key signing operations on the HSM cluster: `multitoken2 -mode rsasigver -key 2048 -nslots 1x12 -password haPwd -f -v`.
2. Remove one HSM from the HA cluster by plugging the network cable out.

**Expected Result:** The signing operation automatically continues on the remaining HSMs.

### 3.4.4 Recovery

**Test Goal:** If an object is generated while one HSM is offline, will the HSMs be synchronized automatically on recovery?

**Test Steps:**

1. Remove one HSM from the cluster by plugging the network cable out or disabling the NTLS service (`service stop ntl`).
2. Using the `cmu` command, create a new cryptographic object on the cluster: `cmu gen -modulusBits=2048 -publicExp=65537 -sign=T -verify=T -slot=1`.
3. Run `vtl haAdmin -show -syncStatus`.
4. Reconnect the HSM (`service start ntl`).
5. Run `vtl haAdmin -show -syncStatus` again.
6. Connect to `hsmone` directly and display the newly created object with `partition showContents -partition haPart -password haPwd`.
7. Repeat Step 6 on `hsmtwo` (partition `haPart`) and `hsmthree` (partition `haPart`).

**Expected Result:** Before the removed HSM is added back to the cluster, `vtl haAdmin -show -syncStatus` should display that the cluster is out of sync (Step 3). After the recovery, the synchronization should happen automatically. `vtl haAdmin -show -syncStatus` should again show that the cluster is fully operational and in sync (Step 5). The created object should exist on all three HSMs and have the same properties (Steps 6 and 7).

### 3.4.5 Changing Object Attributes

**Test Goal:** If object attributes are changed while one of the HSMs is offline, will the updated attributes be synchronized correctly?

**Test Steps:**

1. Run `vtl haAdmin -show -syncStatus` and make sure the cluster is fully operational and synchronized.
2. Create a new object on the whole cluster: `cmu gen -modulusBits=2048 -publicExp=65537 -sign=T -verify=T -slot=1 -label=first`.

3. Remove the first HSM from the cluster by plugging the network cable out or stopping NTLS with `service stop ntl`.
4. Change the `CKA_LABEL` attribute from `first` to `renamed`. For this, we use the interactive `ckdemo` utility:
  - Start the tool: `ckdemo`
  - Select ( 1) `Open Session` and connect to `slot#1` as `normal user[1]`
  - Select ( 3) `Login` and log in as `Crypto-Officer[1]` using `haPwd`
  - Select (25) `Set attribute`, list all objects with the `-1` option, then select the `first` object
  - Select (1) `Add Attribute`, choose attribute `CKA_LABEL` and set the label to `renamed`
  - Then select (0) `Accept Template`. The object is now renamed to label `renamed`
  - Close `ckdemo` with ( 0) `Quit demo`
5. Re-add the first HSM back to the cluster.
6. Run `vtl haAdmin -show -syncStatus` again.
7. Run `cmu list` to show the current objects on the HSMs.
8. On each HSM, list the partition contents using `partition showContents -partition haPart`.
9. Run `vtl haAdmin -show -syncStatus` again.

**Expected Result:** The key is first renamed only on `hsmtwo` and `hsmthree`. When `hsmone` is recovered, the cluster is out of sync (Step 6). After connecting a client to the cluster again (Step 7), the renamed key should exist on all HSMs with the new label. Step 8 will list the key with the new label on all HSMs, and Step 9 shows that the cluster is back in-sync.

### 3.4.6 Creating Objects in a Split Brain Situation

**Test Goal:** If two clients see different parts of the split cluster and create new objects, will the cluster be able to sync correctly after the recovery?

**Test Steps:**

1. Remove the first HSM from the cluster by plugging the network cable out or stopping NTLS with `service stop ntl`.
2. In this configuration, create a new object: `cmu gen -modulusBits=2048 -publicExp=65537 -sign=T -verify=T -slot=1 -label=one`.
3. Re-add the first HSM back to the cluster, then remove the second and third HSM.
4. Repeat Step 2 on the second part, with `-label=two`. Now the two parts of the cluster have different information (“Split Brain”).
5. Using `partition showContents -partition haPart`, verify that both parts of the cluster have generated a different key. One HSM should have objects `one`, the other HSMs objects `two`.
6. Re-add the second members back to the cluster.
7. Repeat the command from Step 5 on all HSMs.

**Expected Result:** When the members are added back to the HA group (Step 6), `vtl -show -syncStatus` will notice that the cluster is out of sync. Once the HSM recovers, the created key will be automatically distributed to all HSMs. In Step 7, all HSMs will have the same objects (`one` and `two`).



### 3.4.7 Deleting Objects in a Split Brain Situation

**Test Goal:** If the cluster is split into two parts, will the deletion of objects be synchronized correctly?

**Test Steps:**

1. Using `cmu`, create two new objects on the whole cluster: `cmu gen -modulusBits=2048 -publicExp=65537 -sign=T -verify=T -slot=1 -label=allone` and `...-label=alltwo`.
2. List the contents of the partitions and take a note of the `allone` and `alltwo` keys: `cmu list -slot=1 -password=haPwd`.
3. Remove the first HSM from the cluster by plugging the network cable out or stopping NTLs with `service stop ntl`.
4. In the first part of the cluster, remove the first key `allone` from Step 1: `cmu delete -slot=1 -handle=9 -password=haPwd`.
5. Re-add the first HSM back to the cluster, then remove the second and third HSM.
6. In the second part of the cluster, remove the second key `alltwo`: `cmu delete -slot=1 -handle=10 -password=haPwd`.
7. Using `partition showContents -partition haPart` on each HSM, verify that both HSMs have deleted a different key. One HSM should have objects `alltwo`, the other HSM `allone`.
8. Re-add the second members back to the cluster.
9. Repeat the command from Step 7 on both HSMs.

**Expected Result:** The keys deleted in Steps 4 and 6 should be deleted across the whole cluster once it is synchronized.

### 3.4.8 Performance Scalability (Multiple Threads)

**Test Goal:** How does the performance scale with increasing number of accessing threads?

**Test Steps:**

1. Using the `multitoken2` tool, start multiple parallel batches of key signing operations on the HSM cluster: `multitoken2 -mode rsasigver -key 2048 -password haPwd -t 30 -f -v -nslots 1x1`
2. Repeat Step 1 for the following number of threads:
  - 1 (`-nslots 1x1`) to 20 (`-nslots 1x20`)
  - 30 (`-nslots 1x30`)
  - 60 (`-nslots 1x60`)
  - 100 (`-nslots 1x100`)
3. Start multiple key generations on the HSM cluster using `multitoken2 -mode rsakeygen -key 2048 -password haPwd -f -v -nslots ...` for the `-nslots` parameters listed in Step 2.

**Expected Result:** SafeNet writes in its help documentation [5] “To achieve maximum performance with Luna SA 5.x, client applications must spawn 50+ threads”. In our setup with three HSMs, we would thus expect that 150 threads utilize the HSMs full performance. Because `multitoken2` can only handle up to 100 threads, we do not expect to see a saturation of the performance in this test. In Section 3.4.9, we will repeat this test with multiple processes, which should maximize the performance.

### 3.4.9 Performance Scalability (Multiple Processes)

**Test Goal:** How does the performance scale with multiple concurrent processes?  
Are requests distributed across the cluster to balance the load?

**Test Steps:**

1. Open two shells. In each shell, start a single-threaded `multitoken2` command: `multitoken2 -mode rsasigver -key 2048 -nslots 1x1 -password haPwd -f -v`.
2. Observe the speed of the two `multitoken2` processes.
3. Repeat this test with the same number of threads used in Test 3.4.8.

**Expected Result:** The performance of two processes accessing the HSM cluster will be similar to the performance we found in Test 3.4.8 with two threads. The library dispatches the requests to different HSMs in the high availability group.

### 3.4.10 Performance Scalability (High Availability Overhead)

**Test Goal:** How big is the performance penalty in the high availability cluster, compared to using three individual HSMs?

**Test Steps:**

1. Disable the `-HAOnly` setting of the client with `vtl haAdmin -HAOnly -disable`.
2. List the available slots with `vtl listSlots` and take a note of the first LunaNet Slot and the HA Virtual Card Slot number.
3. In three separate terminals, run `multitoken2` commands opening threads to each HSM individually: `multitoken2 -mode rsasigver -key 2048 -nslots 1x1 -f`. Replace the slot number with the other LunaNet Slot numbers from Step 2.
4. Stop the `multitoken2` commands again.
5. Start one `multitoken2` command using the HA Virtual Card Slot number: `multitoken2 -mode rsasigver -key 2048 -nslots 4x3 -f`.
6. Repeat Steps 3 and 5 with the `-mode rsakeygen` option to generate RSA keys.

**Expected Result:** The number of operations of signatures calculated should be very similar in both cases, because each operation is dispatched to either HSM, which can calculate the signature independently (Step 3 and 5). However, the generation of RSA keys is slower because each key has to be distributed to the other HSMs (Step 6).

### 3.4.11 Performance Impact of a Slow HSM

**Test Goal:** Is a slow HSM limiting the performance of the HA cluster?

**Test Steps:**

1. Disable the `-HAOnly` setting of the client with `vtl haAdmin -HAOnly -disable`.
2. List the available slots with `vtl listSlots` and take a note of the first LunaNet Slot and the HA Virtual Card Slot number.

3. Measure the number of key signings on one HSM using `multitoken2`:  
`multitoken2 -mode rsasigver -key 2048 -nslots 1x12 -v -f`.
4. Repeat Step 3 on the HA Virtual Card Slot.
5. Start `multitoken2` on the first LunaNet Slot and let it run: `multitoken2 -mode rsasigver -key 2048 -nslots 1x12 -v -f`.
6. Repeat Step 4 while `multitoken2` from Step 5 is still running.
7. Repeat all steps using `-mode rsakeygen`.

**Expected Result:** The key generation is limited by the slowest HSM, because the created object has to be distributed across all HSMs. As a result, in the `rsakeygen` mode, `multitoken2` blocks until each key is distributed. A slow HSM will be the bottleneck and slow the operation down. The slow module will have a linear impact on the signing operations.

### 3.4.12 Different Red Cloning iKeys in a HA Group

**Test Goal:** Is the red iKey information used to encrypt the objects during the replication?

**Test Steps:**

1. On each HSM, create a new partition (Line 19 of Listing 3.3), but do not reuse the same red iKey. Each partition should have its own red iKey.
2. Create a new HA group using `vtl` (Line 62 of Listing 3.3) with the partitions from Step 1. Note that each partition in the HA group has its own red domain/cloning iKey now.
3. Try to create a new key object on the HA group using `cmu gen -modulusBits=2048 -publicExp=65537 -sign=T -verify=T -slot=1 -password=haPwd`.

**Expected Result:** The `cmu` command will fail, because the partitions do not share the cloning iKey.

### 3.4.13 Different Black Partition iKeys in a HA Group

**Test Goal:** Is the black iKey information used to encrypt the objects during the replication?

**Test Steps:**

1. On each HSM, create a new partition (Line 19 of Listing 3.3), but do not reuse the same black iKey. Each partition should have its own black iKey.
2. Create a new HA group using `vtl` (Line 62 of Listing 3.3) with the partitions from Step 1. Note that each partition in the HA group has its own black partition iKey now.
3. Try to create a new key object on the HA group using `cmu gen -modulusBits=2048 -publicExp=65537 -sign=T -verify=T -slot=1 -password=haPwd`.

**Expected Result:** It is a priori not clear whether the black partition iKey is used for the replication. The documentation lists that the red domain/cloning iKey is responsible for the network replication. However, the black partition iKey controls access to the partition and might thus be required for the replication as well. In the backup procedures, both red and black iKey are required.

### 3.5 Threat Model

In our threat model, we assume that we can trust a stand-alone HSM. However, we suspect security relevant implementation issues in the high availability features of the HSM and the library. Also, we look at the security implications of a malicious operator and malfunctioning application.

# Chapter 4

## Results

### 4.1 Requirements Gathered from the Interviews

In the interviews with the Crypto User, Support Analyst and Risk Officer, we found the following requirements:

- Crypto User:
  - The setup of the HSMs has to be coordinated and executed by the security engineering team or the Support Analyst. The Crypto User should only be present to provide his black user iKey during the setup.
  - When a HSM is joined in the HA group, the new HSM should become visible through the PKCS#11 interface without the need to restart the application.
  - After a manual import of key material, the HSMs have to distribute the new keys automatically. There should be no need to enter the key material manually on multiple HSMs.
  - If a HSM fails, the Crypto User should be notified about the failure and the degraded high availability. The failover to another HSM should be transparent for the application.
  - The partitions of the HSM have to be auto-enabled after recovery. The Crypto User should not be required to provide his black key to restart a HSM.
  - When a HSM recovers from a failure or is replaced, it should join the HA group without the intervention of the Crypto User.
- Support Analyst:
  - If a HSM fails, the Support Analyst should be notified about the failure and the degraded high availability. The Support Analyst should also be informed if there is a synchronization issue between the HSMs.
  - There should be a log of the high availability events. Status changes of each HSM as well as the whole HA group should be logged.
  - The setup of the HSMs should follow defined instructions, and all actions have to be recorded in a journal.
  - All partitions in a HA group should have the same partition name and password.
  - The partition and HA group serial numbers have to be logged in a journal.

- Maintenance actions have to be logged in a journal. The Support Analyst has to be able to record information about the high availability, the other HSMs in the cluster, as well as any outstanding tasks to be done on the other HSMs in the cluster.
- The individual HSMs should be distributed amongst multiple data centers to reduce the impact of a data center failure.
- Risk Officer:
  - The HSM high availability solution must work over unencrypted, routed network between data centers.
  - During the replication of objects between the HSMs, the objects have to be encrypted with a key derived from two token iKeys (2-tokens-respectively 4-eyes-principle).
  - The change of a label and exportable/mutable flags of a key has to be distributed across the cluster.
  - The HSM cluster has to be accessible from two different data centers at the same time. Also, the HSMs in a cluster have to be spread over to data centers.

## 4.2 HSM Functionality

The following descriptions are based on the SafeNet Luna SA Help System [5] and the findings from our functionality tests.

### 4.2.1 High Availability

From our functionality tests, we found that the high availability functionality is implemented solely in the `libCryptoki2.so` client library. The library bundles the HSMs of the high availability cluster as an individual PKCS#11 slot, with the description `HA Virtual Card Slot`. Applications using this slot will automatically be routed to one of the physical Hardware Security Modules.

The `libCryptoki2.so` library handles failure and recovery of a HSM transparently to the client. Through a configuration file, the administrator can specify the addresses of the HSMs and configure the recovery behavior of the client.

### 4.2.2 Key Exchange

Because the complete high availability functionality is implemented in the client, the library takes care of the key exchanges. Whenever a key is generated on one of the HSMs, the key has to be transferred to the other HSMs in the cluster.

Based on information from SafeNet, we know that the key replication works in the following way:

1. The client initiates the key generation operation.
2. The `libCryptoki2.so` dispatches this request to the first free HSM.
3. The HSM generates a random key.
4. The newly generated key is wrapped using a derivation of the red domain iKey secret key, and sent back to the client library.
5. The client forwards the wrapped key to the other HSMs in the cluster.
6. Each HSM unwraps the key and imports it into its partition storage.

### 4.2.3 Load Balancing

Load Balancing is implemented on the client side. Each thread is mapped to exactly one HSM. A single-threaded application will utilize the first HSM in the HA group first. If this HSM fails, then the thread is migrated to use the second HSM.

### 4.2.4 Failure Detection

If one of the HSMs in the cluster fails, this is not noticed until the client library tries to make the next PKCS#11 call. Once the library detects the failed HSM, connections to this HSM are moved to a different HSM and operations continue to run.

### 4.2.5 Recovery

For the failover and recovery of a HSM, the library will perform the following steps:

1. Detect that a HSM has dropped out
2. Remove the HSM from the list of available HSMs for load balancing
3. Periodically check if the failed HSM is back online
4. Once the HSM is back, detect whether any key has been changed while the HSM was offline
5. Synchronize keys if necessary
6. Add the HSM back to the list of available HSMs for load balancing

From the tests we saw that the re-synchronization (Step 5) works only, if the client was running when the HSM failed (Step 1) and kept running during the outage until the HSM has recovered (Step 4). An example for such a process is the `multitoken2` command, started before the outage and stopped only after the failed HSM has recovered. In this case, the cluster will be synchronized automatically by the library loaded by `multitoken2`.

If there was no client running when the HSM failed, or the client had stopped while the HSM was offline, then the re-synchronization will not work. In this case, the cluster remains in an unsynchronized state. To bring the cluster back in sync, the operator has to run `vtl haAdmin -synchronize` manually. An example for this case is Test 3.4.4, where the `cmu` utility started after we stopped the first HSM (Step 1). The cluster was not able to synchronize once the HSM recovered later.

Because we had not expected this behavior when we designed our test cases, we did not include this manual synchronization step in the instructions. However, in the test cases where the cluster did not synchronize automatically, we repeated the test with a parallel running `multitoken2` command or by running `vtl haAdmin -synchronize` manually. We list the outcome of this also in the testing results.

## 4.3 Testing Results

### 4.3.1 Successful Functional Tests

The following tests passed as expected:

- Key Distribution (Test 3.4.1)
- Failure Detection (Test 3.4.2)

- Failover (Test 3.4.3)
- Creating Objects in a Split Brain Situation (Test 3.4.6) (\*)

When new keys are generated, they are automatically distributed to the other cluster nodes which are online at that time. If a HSM fails, this can be seen in the client. An example is `vtl haAdmin -show` which will list any failed HSM. Applications can continue to run without interruption when a HSM fails.

Once the failed HSM is recovered and the cluster is synchronized, objects generated during the outage are distributed to the recovered HSM. If the client restarts during the outage, the cluster cannot be synchronized automatically afterwards. An operator has to run `vtl haAdmin -synchronize` manually (see Chapter 4.2.5).

Test 3.4.6 (\*) was only successful when we ran `vtl haAdmin -synchronize -group haGroup` before executing Step 7.

### 4.3.2 Unsuccessful Functional Tests

#### Recovery (Test 3.4.4)

This test did not complete successfully, the keys in Steps 6 and 7 were out of sync. This is caused because the `cmu` command did not run when the HSM failed and recovered again. As a result, the library was not loaded and could not take care of the synchronization. In order to bring the cluster back in sync, we had to run `vtl haAdmin -synchronize` manually.

We repeated the test with a `multitoken2` command running simultaneously. This client had the PKCS#11 library open the whole time. Once the HSM was recovered, the library took care of the synchronization. In this case, the log file of the library shows the recovery attempts:

```

1 14:32:54 [25148] HA group: 74007601 has dropped member: 65005001
2 14:33:28 [25148] HA group: 74007601 recovery attempt #1 failed for
   member: 65005001
3 14:34:29 [25148] HA group: 74007601 recovery attempt #2 succeeded for
   member: 65005001

```

**Listing 4.1:** High Availability Recovery Attempts in the Logfile

We could not configure at which time the first recovery attempt is made. Every following attempt is started in 60 second intervals.

The Test Creating Objects in a Split Brain Situation (3.4.6) failed for the same reason. However, when running a `multitoken2` process parallel to the test case, the `multitoken2` was able to synchronize the cluster automatically. Thus, automatic synchronization upon recovery is possible.

#### Changing Object Attributes (Test 3.4.5)

Due to the arguments explained in Chapter 4.2.5, starting any client such as `cmu` after the first HSM has recovered does not synchronize the cluster. In Test 3.4.5 we tried to manually synchronize the cluster by using `vtl haAdmin -synchronize` after the change of the label attribute. However, the command informed us that the changed object already exists on the target:

```

1 host:~ vtl haAdmin -synchronize -group haGroup -password haPwd
2 Info: object already exist on target
3 Info: object already exist on target
4 Info: object already exist on target
5 Info: object already exist on target
6 Synchronization completed.

```

**Listing 4.2:** Manually Synchronizing a Cluster after Changing Object Attributes



After the manual synchronization, which seems to have completed successfully, `vtl haAdmin -show -syncStatus` still shows that the cluster is out of sync. Even repeated synchronization attempts are not able to bring the cluster back in sync. Attribute changes, in our case on the `CKA_LABEL` attribute, fail to automatically synchronize to the recovered HSM and can not even be manually synchronized using `vtl haAdmin -synchronize`.

#### Deleting Objects in a Split Brain Situation (Test 3.4.7)

Due to the reasons explained in Section 4.2.5, this test failed as well. Because the client library was not loaded during the outage, it could not take care of the re-synchronization and we had to trigger it manually using `vtl haAdmin -synchronize`. However, `vtl` could not determine that the deleted keys had to be removed from all HSMs. Instead, the command transmitted those keys again to the HSMs on which they were previously deleted.

### 4.3.3 Performance Tests

#### Performance Scalability (Tests 3.4.8, 3.4.9 and 3.4.10)

In the Test Performance Scalability (Multiple Threads) (3.4.8), we tested how many threads are required to utilize the HSM cluster completely. We have measured the speeds with 1 to 100 parallel threads. The testing results are listed in Table 4.1 and shown in Figure 4.1.

When calculating RSA signatures, a multi-threaded application will have a speed advantage over a single-threaded application. The individual signing requests are distributed across all HSMs, which increases the total throughput.

However, when generating RSA keys, those keys have to be distributed to all HSMs. This process is not parallelized. As a result, a multi-threaded client is not faster than a single-threaded one.

| Threads | Signatures <sup>a</sup> / Second | Key Generations <sup>a</sup> / Second |
|---------|----------------------------------|---------------------------------------|
| 1       | 87.315                           | 1.032                                 |
| 3       | 265.577                          | 0.899                                 |
| 10      | 468.872                          | 0.865                                 |
| 20      | 702.077                          | 1.032                                 |
| 60      | 1287.616                         | 1.065                                 |
| 100     | 1560.108                         | N/A <sup>b</sup>                      |

<sup>a</sup> 2048 bit RSA key

<sup>b</sup> Multitoken could not handle 100 threads for key generation

**Table 4.1:** Performance Scalability (Number of Accessing Threads)

With Performance Scalability (Multiple Processes) (Test 3.4.9), we wanted to find out if a multi-process application has an advantage over a single-process (but multi-threaded) application. The testing results are listed in Table 4.2 and shown in Figure 4.1.

The test shows that multiple processes with fewer threads have roughly the same performance as a single-threaded application with more threads. Three processes with each 20 threads achieve a performance of 1259 signings per second, while one process with 60 threads achieves 1288 signings.

In Test Performance Scalability (High Availability Overhead) (3.4.10) we wanted to determine the performance penalty in the high availability cluster. We have

| Threads/Process | Signatures <sup>a</sup> / Second |             |                       |
|-----------------|----------------------------------|-------------|-----------------------|
|                 | 1 Process <sup>b</sup>           | 2 Processes | 3 Processes           |
| 1               | 87.315                           | 168.277     | 253.897               |
| 3               | 265.577                          | 485.632     | 637.541               |
| 10              | 468.872                          | 758.383     | 886.203               |
| 20              | 702.077                          | 991.724     | 1258.795              |
| 30              | 847.944                          | 1322.468    | 1572.186              |
| 60              | 1287.616                         | 1619.471    | 1792.804              |
| 100             | 1560.108                         | 1826.082    | 2539.625 <sup>c</sup> |

<sup>a</sup> 2048 bit RSA key

<sup>b</sup> Results from Test 3.4.9

<sup>c</sup> This data point is an outlier. See Figure 4.1 and Chapter 5.1.2.

**Table 4.2:** Performance Scalability (Number of Accessing Threads and Processes)

measured the performance of three individual HSMs and compared it to the performance of a HA cluster with three HSMs. The testing results are listed in Table 4.3 and shown in Figure 4.2.

The test shows that there is a significant overhead for the generation of new keys. The keys have to be distributed to all three HSMs. For signatures, the high availability setup seems to have a big overhead too. However, we have repeated this test with more parallel threads. These additional measurements are listed in Table 4.4. At 24 threads the performance overhead of the high availability setup becomes smaller than the scaling overhead of the stand-alone HSMs. One process with 24 threads, accessing the high availability setup, achieves 731 signatures per second, while three processes with each 8 threads to individual HSMs achieve 720 signatures.

|                                    | Signatures <sup>a</sup> / Second | Key Generations <sup>a</sup> / Second |
|------------------------------------|----------------------------------|---------------------------------------|
| Three Individual HSMs              | 765.3                            | 3.262                                 |
| High Availability HSM <sup>b</sup> | 265.6                            | 1.032                                 |

<sup>a</sup> 2048 bit RSA key

<sup>b</sup> High Availability cluster with three HSMs

**Table 4.3:** Performance Scalability (HA Overhead)

#### Performance Impact of a Slow HSM (Test 3.4.11)

We have measured the impact of a slow HSM. The scenario could be a HSM which is in single-threaded use by another application, or which is placed in a different data center with a slow link or high round trip time.

The impact of a slow HSM in the HA cluster is significant when generating keys. If one HSM is completely used by another process, the key generation on the cluster completely stalls. In addition to the outlined test steps, we tried to create just one single key using `cmu gen -modulusBits=2048 -publicExp=65537 -sign=T -verify=T`, while one HSM was busy. The average time to generate a key was 268.54s. Calculating signatures is also slower, but does not stop completely.

The testing results are listed in Table 4.5.

#### 4.3.4 Security Tests

To understand how keys are distributed between the HSMs, we ran the following two tests.

| Total Threads | 3 stand-alone HSMs |         |                                 | HA HSM setup <sup>b</sup> |         |                                 |
|---------------|--------------------|---------|---------------------------------|---------------------------|---------|---------------------------------|
|               | Processes          | Threads | Signatures <sup>a</sup> /Second | Processes                 | Threads | Signatures <sup>a</sup> /Second |
| 3             | 3                  | 1       | 765.3                           | 1                         | 3       | 265.577                         |
| 9             | 3                  | 3       | 689.7                           | 1                         | 9       | 431.964                         |
| 15            | 3                  | 5       | 784.8                           | 1                         | 15      | 603.655                         |
| 21            | 3                  | 7       | 733.8                           | 1                         | 21      | 702.062                         |
| 24            | 3                  | 8       | 719.7                           | 1                         | 24      | 730.997                         |
| 27            | 3                  | 9       | 782.4                           | 1                         | 27      | 804.557                         |
| 30            | 3                  | 10      | 692.7                           | 1                         | 30      | 847.944                         |
| 45            | 3                  | 15      | 709.2                           | 1                         | 45      | 1077.446                        |
| 60            | 3                  | 20      | 749.4                           | 1                         | 60      | 1287.616                        |

<sup>a</sup> 2048 bit RSA key

<sup>b</sup> High Availability cluster with three HSMs

**Table 4.4:** Performance Scalability (HA Overhead, Multiple Threads)

| HSM                | Signatures <sup>a</sup> / Second | Key Generations <sup>a</sup> / Second |
|--------------------|----------------------------------|---------------------------------------|
| One                | 222.7                            | 1.730                                 |
| HA                 | 331.8                            | 0.975                                 |
| HA <i>one busy</i> | 297.3                            | 0 <sup>b</sup>                        |

<sup>a</sup> 2048 bit RSA key

<sup>b</sup> The key generation completely stalls, without any error messages.

**Table 4.5:** Performance Impact of a Slow HSM

### Different Red Cloning iKeys in a HA Group (Test 3.4.12)

Two partitions were set up with different red cloning iKeys. Those partitions were then successfully grouped into a HA group. But when we tried to create a key on the cluster, the request failed with the following error: **Error: The HA group members do not share the same cloning domain. Synchronization not possible. (-1073741566)**

This test shows that the secret key from the red iKey is used in the key distribution and cloning algorithms.

### Different Black Partition iKeys in a HA Group (Test 3.4.13)

Similar to Test 3.4.12, we also wanted to find out if the information from the black iKey is used in the key replication. We created two partitions with different black partition iKeys and added them into one HA group. This group was fully operational and was able to distribute the keys between the HSMs without any errors. Thus, the black key is not required for the high availability setup.

## 4.4 Other Insights

### 4.4.1 vtl Slot Information Discrepancy

Using the `vtl` command, the HA groups are configured and information about the HSM is displayed. `vtl listslots` lists all current PKCS#11 slots of the `libCryptoki2` library. Direct access to the physical HSMs can be disabled using `vtl haAdmin -HAOnly -enable`. As a result, the clients can only communicate with the HA Virtual Card Slot token, so that the library can ensure that all HSMs stay synchronized.

However even if the `-HAOnly` flag is enabled, `vtl listslots` will list the physical HSMs as slots 1 to 3. Only `vtl haAdmin -show` displays the correct slot information.

```

1 host:~ vtl haAdmin -HAOnly -enable
2 host:~ vtl listslots
3 Slot #   Description           Label           Serial #   Status
4 =====
5 slot #1  LunaNet Slot             haPart         65005001  Present
6 slot #2  LunaNet Slot             haPart         65005002  Present
7 slot #3  LunaNet Slot             haPart         65005003  Present
8 slot #4  HA Virtual Card Slot     haGroup        74007601  Present
9 host:~ vtl haAdmin -show
10 ===== HA Group and Member Information =====
11
12                HA Group Label: haGroup
13                HA Group Number: 74007601
14                HA Group Slot #: 1
15                Synchronization: enabled
16                Group Members: 65005001, 65005002, 65005003
17                Standby members: <none>
18
19 Slot #   Member S/N           Member Label     Status
20 =====
21         -   65005001           haPart          alive
22         -   65005002           haPart          alive
23         -   65005003           haPart          alive

```

**Listing 4.3:** vtl Slot Information Discrepancy

On Line 1, we disable the direct access to the HSM slots. Notice that `vtl listslots` (Line 2) still shows the HSMs directly as slots 1 to 3 (Line 5 to 7) and the high availability group as slot 4. With `vtl haAdmin -show` (Line 9), we get the correct slot information on Line 14. Applications will have to use slot 1 to access the high availability cluster.

### 4.4.2 Clearing Partitions

To clear a whole partition in a stand-alone setup, the administrator can log in to the HSM and simply delete all objects on the partition using `partition clear -partition myPart -password myPwd`. In the high availability setup, this is not so simple, as the cleared partition might be restored by a running client.

The following script clears a partition on a HSM cluster:

```

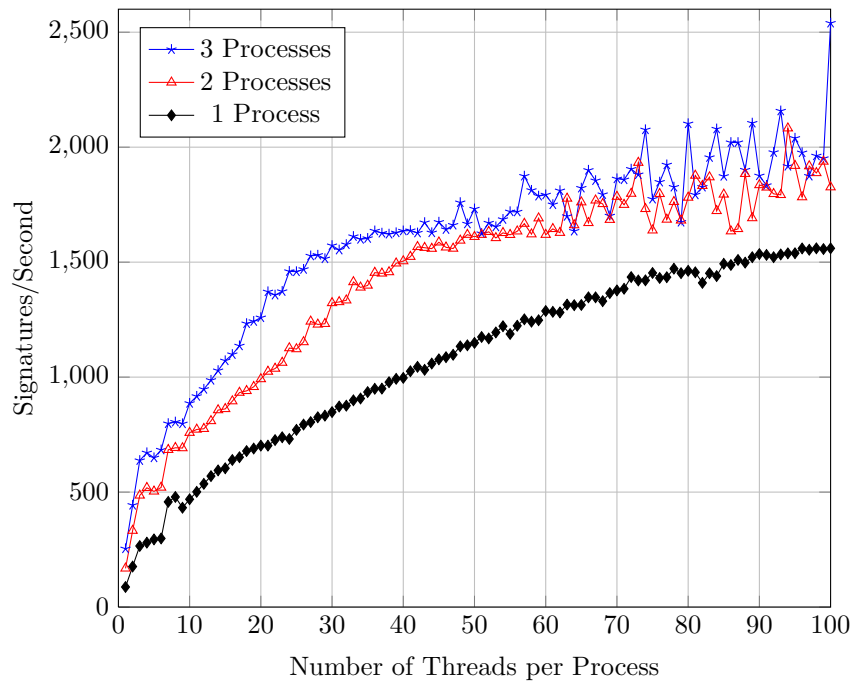
1 #!/bin/bash
2 SLOT=1
3 PASSWORD=haPwd
4
5 cmu list -slot $SLOT -password $PASSWORD | \
6 sed 's/handle=([0-9]*\).*\/1/' | \
7 sort -n -r | \
8 while read handle;
9 do

```

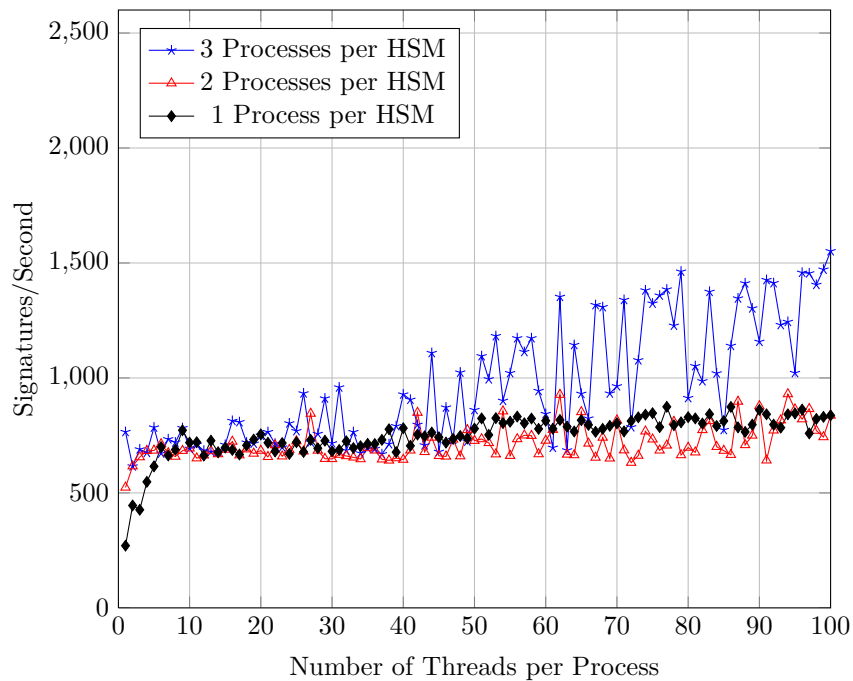
```
10     echo "Deleting handle=$handlenum"
11     cmu delete -slot $SLOT \
12         -password $PASSWORD \
13         -handle $handlenum -f
14 done
```

**Listing 4.4:** Clearing a High Availability Partition

It is important to start at the highest handle number. Whenever a token is deleted, the handles are shifted back. When handle 1 is deleted, the token which was registered at handle 2 before becomes visible at handle 1.



**Figure 4.1:** Performance Scalability (High Availability HSM)  
Each process is accessing all 3 HSMs as high availability group.



**Figure 4.2:** Performance Scalability (Stand-alone HSM)  
Each process is accessing only one stand-alone HSM. This setup is replicated three times: 3, 6 respectively 9 processes accessing 3 HSMs.

# Chapter 5

## Discussion

### 5.1 High Availability Implementation

We already discussed the basic functionality of the HSM high availability solution in Section 4.2. We established that all high availability features are implemented in the client library. This design decision causes the following problems.

#### 5.1.1 Functionality

##### Configuration

The configuration of the HSM has to be repeated on each HSM manually. The operator has to make sure that the modules have the same configurations and policies.

As we have seen in the setup instructions (Line 62 to 69 of Listing 3.3), the setup of the high availability group is done on the client. If a HA group has multiple clients, then the client configuration needs to be updated manually for each client. This leads to a significant maintenance overhead. If a HSM is added to the cluster, the configuration for all clients has to be updated at the same time. Otherwise, a client might not know about the newly added HSM and thus not transfer new keys to this HSM. This would immediately result in an unsynchronized cluster.

Also, the client configuration is essential for the security of the cluster. If a partition is removed from the high availability group, or replaced with another partition, this might compromise the security of the HSM cluster. The operators thus have to make sure that the configuration file can only be modified using clearly defined release procedures.

##### Recovery

Only the client can detect whether all HSMs are available and if the cluster needs to be re-synchronized. If no client is running, the failure of a HSM and a potential subsequent recovery are not detected.

As explained in Chapter 4.2.5, the cluster will remain out of sync once a failed HSM is restored, if the client application was restarted during the outage.

The reason for this is because the state of the HA cluster is solely stored in the client. There is no action log or clean/dirty flags on the HSM itself. When the client is restarted, the whole high availability state is lost. As a result, the client does not know whether it has to replicate or delete keys on the cluster. The current implementation of the high availability features could be improved by using an action log on each HSM. Whenever a HSM recovers, the client could replay

the action log from one HSM on the recovered HSM. This would ensure that the appropriate keys are distributed, while deleted keys are also deleted on the recovered HSM.

Also, the client only checks the status of the HSMs when it is starting an operation on the cluster. As a result, the client can not notify the operator when the HSM actually failed, but only when it tries to make the first request thereafter. The client or cluster can not send out SNMP traps or any other notifications to a monitoring application.

As a work around to these problems, one could write a small monitoring client. This application would simply load the library, log in to the partition and calculate a signing (with a small key) regularly. The library would be loaded the whole time and could thus manage the re-synchronization if one of the HSMs fails. Because it calculates a signature at regular intervals, it would also detect the failure of one HSM and could send out a notification. The drawback of this approach is that the monitoring application needs to know the partition password. Otherwise, it would not be able to synchronize the partitions upon recovery. A monitoring client which runs without knowing the partition password would still be able to detect the failure of a HSM though.

### Changing Object Attributes

As seen in Test 3.4.5, changing object attributes such as the object label, while one of the HSMs is offline, brings the cluster into an unsynchronized status, which `vtl` can not bring back in-sync. The only way to recover this cluster is to inspect the objects on all cluster nodes manually, for example by using `ckdemo`, comparing all key attributes and then deleting the outdated objects on individual nodes manually. Once the outdated objects are deleted, `vtl haAdmin -synchronize` can be used to copy the up-to-date objects again to all HSMs.

We assume that this happens for two reasons: Firstly, each object has a fingerprint attribute (`CKA_FINGERPRINT_SHA1`). Whenever we change one of the other attributes (such as the label), the fingerprint attribute changes as well. `vtl haAdmin -show -syncStatus` seems to query each HSM for the fingerprint attributes of all its objects and then compares the fingerprints across the cluster. Secondly, each object also has a `CKA_OUID` identifier, which stays constant even when we modify attributes. This seems to be a unique identifier for objects on the HSM. When `vtl haAdmin -synchronize` is executed, it will copy all objects which need to be synchronized to the target HSM. However, the target detects that it already has an object with the same `CKA_OUID` and refuses the object instead of updating the changed attributes.

### Deleting Keys

Because the server appliances do not keep a log of operations, especially delete operations, they can not properly synchronize deleted objects. As a result, when a HSM re-joins the cluster, deleted keys might be distributed again, instead of being deleted everywhere.

## 5.1.2 Performance

### Multiple Threads and Processes

In Section 4.3.3, we have seen that a multi-threaded and multi-process application has a speed advantage over a single-threaded application when calculating signatures. Multi-threading allows the library to dispatch the individual requests to different HSMs. SafeNet recommends to have more than 50 threads to maximize



the performance on a single HSM. Thus, a client should reach the performance maximum on a cluster with three modules with around 150 concurrent threads. Because the `multitoken2` test tool can not handle more than 100 threads, we have used concurrent processes to simulate such a load.

In Table 4.1, we see that there is a significant step in the performance between 3 processes with 60 threads each, and with 100 threads each. In Figure 4.1, we plotted the speeds of all tests from 1 to 100 threads and can see that the measurement with 3 processes at 100 threads is an outlier. Based on the figure, we see that there is indeed a performance saturation at around 150 concurrent threads (2 processes with 75 threads, or 3 processes with 50 threads). The measurements for more threads become unreliable. This might be caused by the individual startup times of the parallel processes and the fact that the load balancing is solely implemented in the client library.

For the generation of new keys, multiple threads do not have any impact on the performance. The library will block until the generated key is distributed to each HSM, and thus can not parallelize the requests. However, if a second process is running at the same time, the key generation on the HSM can stall completely, as we have seen in Performance Impact of a Slow HSM (Test 3.4.11).

### High Availability Overhead: Signatures

In our test, we compared the performance of three stand-alone HSMs with the setup of a high availability cluster consisting of three HSMs. We have seen that 24 parallel threads will achieve roughly the same performance on three stand-alone HSMs as on a high availability setup.

For applications which rarely generate any new keys, the operators might decide to use the three Hardware Security Modules each in a stand-alone setup. Whenever a new key is generated, the operator would then manually transmit this key (using the backup and restore functionality) onto the other HSMs. Such applications will have a performance advantage, if they use less than 24 parallel threads.

Applications using a high availability setup should have at least 24 parallel threads in order to benefit from the performance of the setup. For applications with less threads, the operators have to decide if it would make sense to forego the automatic key replication and increased reliability and use stand-alone HSMs instead.

### High Availability Overhead: Key Generation

For the key generation, the high availability overhead becomes significant. The newly generated key has to be encrypted, transferred to the client, relayed to the other HSMs, where it can be decrypted and installed again. The library will wait until the newly generated key is distributed to each HSM before the operation returns. This has the disadvantage that subsequent key generations will have to wait before they can proceed. This might especially become problematic if an application creates many session objects.

If another application is using the whole performance of just one HSM, then the key generation might completely stall. This could be a problem in a production setup, where multiple single-threaded applications access the HSM cluster.

### Provisioning

There is no mention of a resource provisioning in SafeNet's documentation [5]. It is not possible to allocate resources to a specific partition on the appliance or to a specific client. As a result, one can not guarantee the performance to an application, if there is more than one application using the cluster.

### 5.1.3 Security

#### Partition Cloning

The black iKey is responsible to grant access to the contents of the partition. The red iKey has the cloning domain secret stored. In the non-HA setup, both the red and black iKeys are required to create or restore a backup of a partition. In the HA setup, only the red iKey is used in the network distribution of new objects and the synchronization of the partitions. Thus, the red iKey alone is enough to clone a partition in the high availability setup.

Suppose the red iKey owner is a malicious operator. He is the legitimate owner of the red iKey for a partition in a HA setup. Because of the segregation of duties principles in his organization, he does not have access to either the black partition iKey nor the blue Security Officer iKey. Using an empty black and blue iKey, he sets up a completely new HSM. When he creates the new partition, he re-uses the red iKey of which he is the legitimate owner. Then, he adds his new partition into the high availability group in his client and start the synchronization. The synchronization will work properly because he re-used the red cloning domain iKey. As a result, he has an identical clone of the source partition, but now also owns the corresponding black and blue iKeys. This gives him the power to use all secrets on the partition, as if he was the real owner of the data. It gives him the possibility to create a backup of a HSM without needing the legitimate black iKey. Thus, it undermines the security of the backup and restore process.

However, for this attack to work, the red iKey owner has to know the administrator password of the HSM or the root password for the application server. To enforce the four-eyes principle, we can split the red iKey using the M-of-N key sharing scheme [8] and have at least two independent red iKey owners.

In the non-HA setup, it is important to turn the Network Replication partition policy Off, such that a synchronization command from the client is refused.

#### Deleting Keys

Deleting keys on a cluster only works properly if all HSMs are online when the key is deleted. Otherwise, a disconnected HSM will still store the compromised key and synchronization will bring this key back onto the other HSMs once the failed HSM recovers. As a result, an application might continue to use a compromised key, even though the operators have deleted the key previously on the HSM cluster.

As the HSM cluster does not leak any secrets, we do not consider this to be a security problem of the HSM itself. Rather, it becomes a potential security problem if a malfunctioning application does not verify whether the keys it uses are still valid.

Additionally, this problem might lead to a denial of service as the HSM partitions have a limited size. If a large number of previously deleted keys are un-deleted, the partition size might be reached and subsequent key generations will fail.

### 5.1.4 Possible Improvements

The above discussed problems could be mitigated by making the following improvements on the high availability solution. The main functionality could still remain in the client library, but the HSM itself should be aware of the high availability setup.

**Client Configuration:** Instead of having to configure each client manually, the configuration could be stored on the HSM cluster. Whenever a new client is set up, it could (using a minimal bootstrap configuration) retrieve the global configuration from the HSM setup. This would make it easier to add and remove a HSM from the cluster, because the operators do not have to update each client's configuration manually anymore.

**HSM Configuration:** The HSMs should have a way to synchronize their configuration. Each HSM in a high availability cluster has to have the same HSM and partition policies. If one of the modules is configured differently, some applications might fail on one HSM, but succeed on another. An automatic synchronization of the HSM configuration would solve this problem.

**Key Deletion:** The automatic synchronization of keys after an outage could be facilitated by keeping an action log on each HSM. Whenever a HSM recovers from an outage, the action log from the other HSMs is replayed. This would ensure that deleted keys are consistently deleted across the cluster. As an alternative, the two-phase commit protocol could be implemented for the deletion of keys.

**Key Attributes:** The replication of changes in the key attributes has to be distributed across the whole cluster. When a HSM receives an object with an `CKA_` `UUID` which it already stores itself, it has to compare the two `CKA_` `FINGERPRINT_SHA1` properties of the objects and update the key attributes if necessary.

**Encryption during Replication:** When synchronizing HSMs, the key information from both black and red iKey should be used. This ensures that a rogue red iKey owner can not violate the four-eyes principle and clone a HSM himself.

As described in Towards Robust Distributed Systems [9], a distributed system can have at most two of the following three properties: Consistency, Availability and Tolerance to network Partitions. The above suggestions focus on consistency and availability and addresses the security relevant issues discussed in Chapter 5.1.3.

## 5.2 Operating Policy

Based on the requirements gathered from the users and the testing results and implications, we have designed the following guidelines to configure the HSM in a high availability cluster. We list recommendations for the application developers how to maximize the performance of the HSM cluster.

### 5.2.1 Setup Guidelines

#### HSM Policies

The following HSM policies have to be enabled in order to use the high availability features:

- Allow cloning
- Allow network replication

On each partition used in the HA group, the following partition policies need to be enabled:

- Allow private key cloning
- Allow secret key cloning
- Allow high availability recovery

In addition to the partition policies above, also the activation policies should be enabled. If those policies remain disabled, a HSM can not automatically re-join a HA group when the HSM recovers from an outage.

- Allow activation
- Allow auto-activation

In the client, the following settings should be made:

- Using `vtl`, enable the `HAOnly` setting in order to disable the direct access to the HSMs. This is important for two reasons: Firstly, a misconfigured application might access the HSM directly, if the physical slots are exposed. This would immediately lead to an unsynchronized cluster. Secondly, if the physical slots are exposed and a HSM fails, slot numbers will shift while clients are connected. In our tests, the `multitoken2` and the `ckdemo` could not handle this situation and crashed.
- Set the recovery interval to 60 seconds and the maximum number of recovery tries to infinity. The client will detect a recovered HSM as soon as possible, re-added it to the HA group and start the synchronization. If the recovery interval is larger, the risk will increase that the application shuts down before the recovered HSM is detected and re-synchronized. This would lead to an unsynchronized cluster which requires manual intervention to re-synchronize again.

### Naming, Passwords and iKeys

- All partitions in a HA group must have the same partition password. This is a technical requirement.
- In order to simplify the administration, all partitions in the HA group should have the same partition name. Also, the HA group name should be derived from the partition names.
- Because the clients and HSMs communicate through the NTLS link, the authenticity of the client and server has to be verified by checking the fingerprints of the SSL certificates.
- As we have discussed in Section 5.1.3, a M-of-N key sharing should be used for the red domain iKey. The red iKey should be split into at least two parts, owned by two independent key holders.

### Journaling

The operators of the HSM should maintain a journal where all HSM operations are logged. In addition to the existing logging conventions, the following static information and actions should be noted:

- Partition names and serial numbers, HA group name and serial number
- IP addresses of the HSMs in the HA cluster
- Fingerprints of all client and server SSL certificates
- Changing the partition policies \*
- Adding or removing a client to a partition \*
- Creating a new HA group, adding and removing partitions to it ⊕

- Enabling or disabling the HAOnly setting ☉
- Changing the auto recovery interval and number of retries ☉
- Manually synchronizing a HA group

The actions denoted with an asterisk \* have to be executed on each HSM manually. Actions with a circled asterisk ☉ have to be done on all clients which use this HA group. Note in the log whether the action was done on all HSMs/clients or if it has to be executed on one of the HSMs/clients at a later time. We suggest a journal format like shown in Table 5.1.

### Monitoring

In addition to the existing HSM monitoring, the high availability logging should be enabled with `vtl haAdmin -HALog -enable` and the `haErrorLog.txt` log file should be monitored for the messages in Table 5.2.

The first message in Table 5.2 indicates that a HSM has failed. The second and third messages do not require any direct action of the operators. The fourth message indicates that the client library has restarted while one of the HSMs was offline. As a result, the library is not able to synchronize the recovered HSM. The operators have to inspect the objects on each HSM and then perform a manual synchronization.

### 5.2.2 Application Recommendations

In order to take advantage of the high availability features, applications should follow these recommendations:

- Signing operations should be made in multiple parallel threads. This way, the requests can be distributed across the whole HSM cluster, which yields in a higher performance. The best performance can be achieved with 150 threads.
- If the application is deleting keys, it has to make sure that the cluster is fully operational when a key is deleted. Otherwise, the deleted key might reappear when a recovered HSM is synchronized (see Section 5.1.3). The application has to keep a list of deleted keys in its own database and cross-check the keys validity before using it on the HSM cluster.
- At the moment, changing object attributes only works when all HSMs are on-line. To us, this looks like a bug in the implementation of the high availability features. We hope that this problem can be solved by SafeNet soon. As long as the bug exists, applications should not change any key attributes while one of the HSMs is offline.
- Because each generated key has to be distributed to each HSM, an application which generates many session objects will suffer from the performance penalty.

### 5.2.3 Requirements and Tradeoffs

#### Fulfilled Requirements

In Chapter 4.1, we listed the requirements from the Crypto User, Support Analyst and Risk Officer. With the policies outlined in Chapter 5.2, we have met all requirements from the Crypto User. Due to limitations in the high availability implementations, we could not fulfill all requirements of the Support Analyst and the Risk Officer.

| Date          | Action                                     | Clients  |          |     | HSMs   |        |          | Complete | Notes                  |
|---------------|--|----------|----------|-----|--------|--------|----------|----------|------------------------|
|               |  | Client 1 | Client 2 | ... | hsmone | hsmtwo | hsmthree |          |                        |
| 24. Jan. 2013 | Enable HAOnly                              | ✓        | ✓        | ✓   | -      | -      | -        | yes      |                        |
| 24. Jan. 2013 | Remove partition 65005001 from haGroup     | ✓        | ×        | ✓   | -      | -      | -        | no       | To be done on Client 2 |
| 25. Jan. 2013 | Enable Auto-Recovery policy (21) on haPart | -        | -        | -   | ✓      | ✓      | ✓        | yes      |                        |
| 25. Jan. 2013 | Assign client 192.168.1.10 to haPart       | -        | -        | -   | ×      | ✓      | ✓        | no       | To be done on hsmone   |
| ...           | ...  |          |          |     |        |        |          |          |                        |

✓ successful action  
 × unsuccessful action (needs to be executed later)  
 - action not necessary

Table 5.1: Example HSM Journal

|    | Perl Regular Expression   | Message Type | Example Message   |
|----|---|--------------|---|
| 1. | /^.*HA group: (\d+) has dropped member: \(\d+\)\$/  | Alert        | 14:32:54 [25148] HA group: 74007601 has dropped member: 65005001  |
| 2. | /^.*HA group: (\d+) recovery attempt #\d+ \ failed for member: (\d+)\\$/  | Alert        | 14:33:28 [25148] HA group: 74007601 recovery attempt #1 failed for member: 65005001   |
| 3. | /^.*HA group: (\d+) recovery attempt #\d+ \ succeeded for member: (\d+)\\$/   | Recovery     | 14:34:29 [25148] HA group: 74007601 recovery attempt #2 succeeded for member: 65005001  |
| 4. | /^.*HA group: (\d+) unable to reach member: \(\d+\) Manual Recover or Auto Recovery will \ not be able to recover this member\\$/ | Alert        | 14:38:22 [25148] HA group: 74007601 unable to reach member: 65005001. Manual Recover or Auto Recovery will not be able to recover this member |

Table 5.2: High Availability Log Messages

### Tradeoffs

We have not met the following two requirements from the Support Analyst and Risk Officer:

- *If a HSM fails, the Support Analyst should be notified about the failure and the degraded high availability. The Support Analyst should also be informed if there is a synchronization issue between the HSMs.*

Monitoring of the HSMs only works if an application is connected and performing operations on the HSM. Thus, the support staff can not be alerted as soon as the HSM fails. The client only detects the failure when it tries to execute the next operation on the cluster.

- *The change of a label and exportable/mutable flags of a key has to be distributed across the cluster.*

As explained in Section 5.1.1, changes of the object attributes are not distributed correctly. This seems to be a bug in the implementation for which SafeNet hopefully provides a patch.





## Chapter 6

# Conclusion

While FIPS PUB 140-2 [2] governs general security features of Hardware Security Modules, specific high availability functionality is not covered by the standard. Therefore, it is important to have additional procedures and policies in place when using HSM modules in a production environment.

We have built a HSM high availability setup using three SafeNet Luna SA 5 HSMs. On this cluster, a series of functional, performance and security tests were conducted. In addition to running the tests, we interviewed users, support staff and the risk officer to gather requirements for a HA setup.

From the testing results, we see that the basic high availability features of the cluster are implemented in the client library. This leads to three important conclusions. First of all, the recovery procedures can only work as long as a client is connected to the cluster during the failure, outage and recovery of a HSM. Otherwise, newly created cryptographic objects can not be distributed across the HSMs, resulting in an un-synchronized cluster. Secondly, deleted keys can re-appear when a cluster is synchronized after an outage. This can potentially lead to a security breach when deleted keys are used later again by a malfunctioning application. The application has to make sure that it only uses valid keys. Thirdly, the owner of the cloning domain red iKey can set up a new HSM and join it into the high availability group without using the black iKey. This violates the four-eyes principles. Splitting the red iKey using a M-of-N scheme and distributing the individual key parts to distinguished key owners resolves this issue.

High availability, clustering and distributed systems are topics which are well studied in research already. Two-phase commits, heartbeat monitoring and quora are widely used in other clustering solutions and could also be implemented in a HSM high availability solution. We gave specific suggestions for improving SafeNet's current high availability implementation.

All findings from the testing and the interviews were compiled into the Operating Policy. This document specifies how HSMs in a high availability setup can be operated in a secure manner. In addition, we gave design recommendations for applications using those HA features. It is important to have such a policy documentation in addition to the official FIPS PUB 140-2 certification. Our tests have shown that the high availability features are not covered well enough by the FIPS certification itself.

While we have looked at the high availability features in the SafeNet Luna SA 5 HSM, there are other vendors offering HA features in their HSMs as well. Most likely, they follow completely different approaches for the implementation of their solution. It would be interesting to see advantages and disadvantages of their implementations and compare them to SafeNet's high availability solution.



# References

- [1] T. C. S. de Souza, J. E. Martina, and R. F. Custódio, “Audit and backup procedures for Hardware Security Modules,” 2008.
- [2] National Institute of Standards and Technology, Information Technology Laboratory, “Security Requirements For Cryptographic Modules FIPS PUB 140-2,” May 2001. [http://www.nist.gov/manuscript-publication-search.cfm?pub\\_id=902003](http://www.nist.gov/manuscript-publication-search.cfm?pub_id=902003).
- [3] J. E. Martina, T. C. S. de Souza, and R. F. Custodio, “OpenHSM: An Open Key Life Cycle Protocol for Public Key Infrastructure’s Hardware Security Modules,” *Springer LNCS 4582, Lecture Notes in Computer Science*, pp. 220–235, 2007.
- [4] SafeNet, Inc., “Level 3 Non-Proprietary Security Policy For Luna®PCI-e Cryptographic Module,” 2011.
- [5] SafeNet, Inc., “SafeNet Luna SA Help System.” Technical guides for the Luna SA HSM, provided by SafeNet to its customers.
- [6] C. Ellison, “Ceremony Design and Analysis,” *ePrint Archive*, vol. Report 2007/399, 2007. <http://eprint.iacr.org/>.
- [7] M. Silverboard, “Luna SA HSM Concepts.” <http://geekcredential.wordpress.com/2012/07/02/luna-sa-hsm-concepts/>, accessed on 1. October 2012.
- [8] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, pp. 612–613, Nov. 1979.
- [9] E. A. Brewer, “Towards robust distributed systems (abstract),” in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, (New York, NY, USA), pp. 7–, ACM, 2000.