Thales HSM

Digital Key Management
Concepts and Exercises

Chris Snow
Colin Cook
Nick Bitounis

Thales HSM: Digital Key Management Concepts and Exercises

by Chris Snow, Colin Cook, and Nick Bitounis

Table of Contents

List of Figures

List of Tables

List of Examples

Preface

About this book

A primary role of a HSM is the secure management of digital keys. This book introduces digital key management concepts and reinforces those concepts with exercises that the reader can perform on an open source Thales HSM Simulator.

The reader should have a basic understanding of symmetric and asymmetric cryptography. Appendix A, *Introductory Books in Cryptography* provides a list of introductory books in cryptography for those wishing to learn the basics or just wanting to refresh their knowledge in the field of cryptography.

TODO: This book is written sequentially? Chapters x to y (TODO) should be read in order?

Audience

Undergraduate and graduate students should find that this book supplements their studies in the theoretical concepts of cryptography with practical applications.

Software Engineers and Architects designing and building security solutions using the Thales brand of HSM will learn concepts and patterns of key management that can be applied to their designs.

Book Organisation

Part I, “HSM Concepts” introduces digital key management concepts and provides the reader with practical exercises to perform on a Thales HSM Simulator.

Chapter 1, *Introduction* describes the role fulfilled by a HSM. The Thales series of HSM's are then introduced with a short history of their evolution. Finally, an open source Thales Simulator project is presented, with hands on exercises for the reader to install the Simulator and then connect to it from Java, C#, and ruby clients.

Chapter 2, *HSM Local Master Keys (LMKs)* covers in detail the concept of the LMKs. The knowledge gained in this chapter is fundamental to your understanding of the Thales HSM. Almost all other chapters depend on you understanding the material covered in this chapter. This chapter concludes with exercise with the Thales Simulator to help instill the concepts of LMKs.

Chapter 3, *Key Concepts* describes general key concepts that you need to know in addition to the material covered in the previous chapter about LMKs. The knowledge in this chapter is of vital importance when interacting with the Thales HSM. Exercises are provided with the Thales Simulator to put the concepts learnt in this chapter into practise.

Chapter 4, *Secure Key Exchange* two sites that are secured with HSMs need to have a set of keys that are shared between the HSMs. This chapter describes how keys are created and shared. Exercises are given to setup two demo sites with the Thales Simulator and generate and share keys between the demo sites.

Chapter 5, *Dynamic Key Exchange* ... TODO

Part II, “Inside the Thales Simulator” ... TODO ... This part takes you on a code walk through of the Thales Simulator project.

Conventions

TODO

Part I. HSM Concepts

TODO: Intro to this part.

Chapter 1. Introduction

Abstract

what is a hsm?

why do we need a hsm?

Thales hsm

Thales Simulator project

Introduction

A primary role of a HSM is the secure management of digital keys. This document describes digital key management concepts, and also describes some key management patterns for solving specific problems.

Other acronyms for Hardware Security Modules include:

Other acronyms and definitions for HSM

`HSM`	Hardware Security Module
	Host Security Module
`TRSM`	Tamper Resistant Security Module
`SCD`	Secure Cryptographic Devices

Why digital key management? Life without a HSM?

Key encrypting Key versus Data Encrypting Key

Thales Simulator project overview.

Thales History and Versions

Introduction to the Thales Simulator

Console Commands versus Host Commands

Console Commands

Host Commands

Exercises

Setting up the Thales Simulator

In this exercise, you will download, install and run the Thales Simulator on a Windows machine.

Download and Install the Thales Simulator

1. Download ThalesSim.Setup.0.9.6.x86.zip from: http://thalessim.codeplex.com/releases/view/88576

2. Unzip the downloaded file and execute the file `ThalesWinSimulatorSetup.msi`, accepting the default options.

Starting the Thales Simulator

1. Navigate to the folder where you installed the Simulator (E.g. C:\Program Files (x86)\NTG\Thales Simulator)

2. Execute `ThalesWinSimulator.exe` (if your are running Windows 7, right click the file and select Run As Administrator)

3. Click the Start Simulator button. TODO: insert image.

4. In the Application Events window, the simulator will inform you that it could not find a file containing the LMK keys so it will create a new set of keys for you. When the simulator creates new keys in this manner, the same keys will always be created. TODO: insert image.

The Thales Simulator Console

In this section, we will connect to the HSM Console and run a basic command, **Query Host (QH)** to test connectivity to the HSM.

For a full list of console commands, refer to the “Console Reference Manual” which is available from Thales. Note that the Thales Simulator only implements a subset of the commands. A list of implemented console commands can be found here http://thalessim.codeplex.com/wikipage?title=list%20of%20supported%20console%20commands.

Connecting to the console

1. Start the simulator as described in Starting the Thales Simulator.

2. Click Connect to console.

3. Enter the command **QH** followed by **ENTER**. You should see something similar to this: TODO

Thales Simulator Java client application

In this session, we connect to the HSM over TCP/IP using Java. When we connect using Java, we can send Host Commands to the HSM.

In the code example, below, we send the command **Perform Diagnostics (NC)**, and print the response to System.out.

For a full treatment of Host Programming the Thales HSM, refer to the Thales documentation “Host Programmer’s Manual”. For a full list of Host Commands, refer to the Thales documentation “Host Command Reference Manual”

```
package demo;

import java.io.BufferedOutputStream;
```

```
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class Main {

    /**
     * Runs the Command "Perform Diagnostics (NC)" and
     * prints the response to System.out
     *
     * An example response is:
     *
     * !0000ND007B44AC1DDEE2A94B0007-E000
     */
    public static void main(String[] args) throws Exception {

           Socket socket = new Socket("127.0.0.1", 9998);

           // by default the Thales Simulator has a header of 4 bytes
           // so set these to 0000
           String command = "0000NC";

           // leave two bytes for inserting the command length
           byte [] commandBuffer = ("  " + command).getBytes();

           // populate the command length
           commandBuffer[0] = (byte) (command.length() / 256);
           commandBuffer[1] = (byte) (command.length() % 256);

           // Write the command to the HSM
           OutputStream out = socket.getOutputStream();
           BufferedOutputStream bufferedOut =
                        new BufferedOutputStream(out, 1024);
           bufferedOut.write(commandBuffer);
           bufferedOut.flush();

           // Read the response from the HSM
           InputStream in = socket.getInputStream();
           int result;
           while ((result = in.read()) != -1) {
                  System.out.print((char)result);
           }
           socket.close();
    }
}
```

Summary

Chapter 2. HSM Local Master Keys (LMKs)

Abstract

TODO: abstract

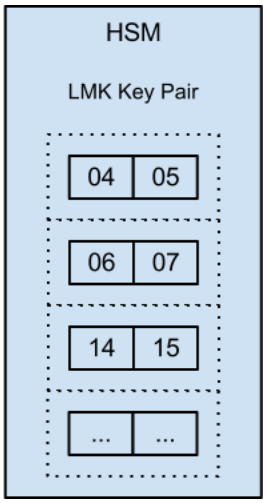
LMK Overview

Local Master Keys are a set of DES or triple DES keys. They are stored securely in the HSM making it very difficult for an attacker to gain access to them. LMKs are the only keys that are stored in the HSM.

LMKs are not used for encrypting data, but are instead used to encrypt and decrypt other keys as these enter or leave the HSM. LMKs are used to ensure that even if the data traffic between the HSM and an application is recorded, the clear values of any exchanged keys are not compromised.

LMKs come in pairs and the Thales HSM contains several LMK pairs. Different LMK pairs are used to encrypt/decrypt different types of security keys. LMK pairs are identified by two numbers, for example LMK pair 04-05, LMK pair 14-15, etc. See Figure 2.1, “HSM Local Master Keys”.

Figure 2.1. HSM Local Master Keys



Each LMK pair is assigned a code. LMK pair 04-05 is assigned code 00, while LMK pair 14-15 is assigned code 02. The full list of HSM key pairs are listed in Table 2.1, “LMK Key Pairs”, below. Note that HSM key pairs do not start at 00-01, instead the numbering starts at 04-05, and runs non-contiguously to 38-39.

Table 2.1. LMK Key Pairs

Key Pair	Code
04-05	00
06-07	01
14-15	02
16-17	03
18-19	04

Key Pair	Code
20-21	05
22-23	06
24-25	07
26-27	08
28-29	09
30-31	10
32-33	11
34-35	12
36-37	13
38-39	14

Each HSM has a unique set of LMK pairs that can be either randomly generated or loaded from smart cards. HSM users jealously guard the LMKs because the integrity of the key management security scheme depends upon them.

LMK Variants

Back when the HSM had only a handful of LMK pairs, more than the type of keys that had to be encrypted, a way had to be found to ensure that different key types can be used but also provide a way to identify parity errors with these key types. Variants are an easy way to pseudo-multiply your LMK pairs. (TODO validate this)

Keys are encrypted under LMK pairs using either the clear value of the LMK or a variant of the LMK. An LMK variant is created by performing a XOR operation with a value on the LMK key. For example, variant 1 of an LMK is created by XORing the LMK with the value 00000000000000000000000000000A6. The Thales HSM supports 10 variants for each LMK pair, with variant 0 being the clear LMK itself. The full list of variant calculation functions can be seen in Table 2.2, “LMK Variant Calculation Function”

Figure 2.2. HSM Variants

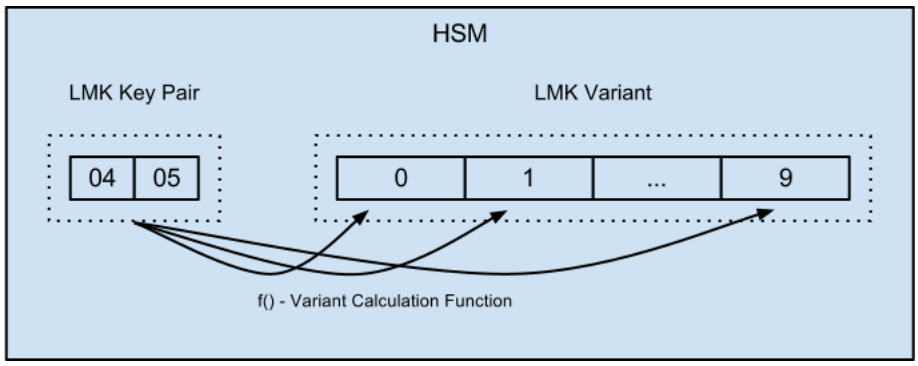


Table 2.2. LMK Variant Calculation Function

Variant number	Variant Calculation Function XOR LMK with:
0	Not applicable - use clear value of LMK

Variant number	Variant Calculation Function XOR LMK with:
1	0000000000000000000000000000000A6
2	00000000000000000000000000000005A
3	00000000000000000000000000000006A
4	0000000000000000000000000000000DE
5	00000000000000000000000000000002B
6	000000000000000000000000000000050
7	000000000000000000000000000000074
8	00000000000000000000000000000009C
9	0000000000000000000000000000000FA

Exercises

TODO - what exercises could be performed on the Simulator around LMK Concepts?

Chapter 3. Key Concepts

Abstract

TODO: abstract

Key Types

Each different key supported by the Thales HSM has a unique code called the key type code. The key type is a three-digit number and is made up from the code of the LMK pair and the LMK variant. Therefore, keys encrypted under LMK pair 04-05 using a variant 1 will have a key type equal to “00” + “1” = 001. It is important to understand key types since several Thales commands expect key type codes as parameters. The full list of key types can be seen in Table 3.1, “Key Types”

Table 3.1. Key Types

LMK Key Pair	LMK Code	Variant	Key Type Code	Key Type
04-05	00	0	000	ZMK
04-05	00	1	001	ZPK

Key Check Value (KCV)

The check value of a key is derived by DES encrypting zeroes[1] using that key. For example, the KCV for key 0123456789ABCDEF is D5D44FF720683D0D.

The purpose of a KCV is to ensure that a key has been correctly transmitted between different parties and that they key is also configured correctly in systems. It is common practice to transmit the KCV of a key along with the key itself. Sometimes, the complete result of the DES encrypt operation is used, but typically only the first six digits are used (in the example, D5D44F).

[1] TODO: xxx zeros to be precise.

Key Formats

The Thales HSM uses two major formats when processing security keys. These are *Variant* and *ANSI*.

The variant concept is similar to the one used to form LMK variants: a key is XORed with a value to form the key variant. Double-length keys are XORed with the value:

```
00000000000000A6 000000000000005A
```

and triple-length keys are XORed with the value:

```
00000000000000A6 000000000000005A 000000000000006A
```

TODO: how does the variant concept differ to the LMK variant concept?

The ANSI format is much simpler: a key is used as-is without performing any additional operations on it.

Key Scheme

Depending on their length and key format, keys are designated by a key scheme that helps to quickly identify the nature of a key. Key schemes are the following:

Table 3.2. LMK Key Schemes

Key Scheme	Description
Z	Single-length ANSI keys.
U	Double-length variant keys.
T	Triple-length variant keys.
X	Double-length ANSI keys.
Y	Triple-length ANSI keys.

Exercises

TODO - what exercises could be performed on the Simulator around LMK Concepts?

Chapter 4. Secure Key Exchange

Abstract

TODO: abstract

Key export, key import and security implications.

Overview

To securely exchange information between two users using the DES encryption scheme, it is vital to securely share a set of initial keys. This role is fulfilled by the Zone Master Keys. Unlike an LMK which does not leave the HSM, ZMKs are intended to be shared between sites to create *secure Zones*. The ZMK is distributed manually between the sites. The ZMK allow future (data encrypting) keys to be automatically shared between sites by encrypting those future keys with the ZMK. In that regard, they work very much like the LMKs with the important exception that they can be shared between users.

ZMKs have a key type code of 000 (they are encrypted under LMK pair 04-05 with a variant of 0).

TODO diagram

The transfer of the ZMK between sites is performed manually. The ZMK is shared in parts (components) so that no one person will see the key.

TODO diagram

The data encryption key is a Zone PIN Key (ZPK). The ZPK was historically used to encrypt PINs for transfer between sites (e.g. between Acquirer and Issuer).

For local storage (e.g. on the application server using the ZMK), the ZMK is encrypted under one of the LMK keys.

Example 4.1. Secure Key Exchange

Two parties want to exchange a ZMK. One party generates a random ZMK using three clear components which are the following:

```
2CBF0D8FA4E66ECE 6B239E25B9BAD934
 B60825E3790D31CE 4A4AA74397461C13
 29BFE3C1D0C1E50B CD7038A42CFB160B
```

TODO: describe what is meant by clear components

Each of these clear components are kept by a separate custodian that works for the first party and are delivered to different custodians of the second party. To create the complete ZMK, each custodian enters their component to the HSM which combines them to form the ZMK. Most typically, the clear components are simply XORed to form the ZMK. In the example, the ZMK value is:

```
B308CBAD0D2ABA0B EC1901C20207D32C
```

When generating the ZMK, the first party also gives the KCV of the ZMK to the second party (for the example key the KCV is 6CE4CF). That way, the second party can verify the correct reception and data entry of the ZMK components.

TODO: See “Laboratory 04 – Creating a ZMK” for hands on experience creating a ZMK.

Exercises

Creating a Zone Master Key (ZMK)

In this exercise, we create a Zone Master Key (ZMK) using console commands.

The ZMK is distributed manually as components. To create the ZMK, we first create three ZMK components.

Generate the ZMK Components

Generate three ZMK components using the console command **Generate Component (GC)**. Repeat the command three times as shown below:

TODO: describe the various command options and how to choose which option values to use.

```
GC
Key length [1,2,3]: 2
Key Type: 000
Key Scheme: U
Clear Component: 79CD 2380 9B4F C1C4 7F9E FB2A DF2A 674A
Encrypted Component: U 1BA5 185A FCF1 5A1B 274B E1E0 03B4 7C2A
Key check value: 7A5B C7
GC
Key length [1,2,3]: 2
Key Type: 000
Key Scheme: U
Clear Component: 0157 B3DF 6116 3402 372C 54FD 62F2 1C91
Encrypted Component: U FCE4 7AF7 FFF8 9F40 2407 A35A F063 D3E1
Key check value: 1E79 CB
GC
Key length [1,2,3]: 2
Key Type: 000
Key Scheme: U
Clear Component: 7AEA B5A4 1A9E 9B68 EF80 494C 0819 4ADA
Encrypted Component: U EE8D 4F9E C8B2 ADF4 9CD2 F0D2 7F5C 95C5
Key check value: 277A 5F
```

Generate the ZMK from the components

This step uses the **FK** command to generate the ZMK from the three ZMK components previously generated:

TODO: describe the various command options and how to choose which option values to use.

```
FK
Key length [1,2,3]: 2
Key Type: 000
Key Scheme: U
Component type [X,H,E,S]: E
Enter number of components (2-9): 3
```

```
Enter component #1: U 1BA5 185A FCF1 5A1B 274B E1E0 03B4 7C2A
Enter component #2: U FCE4 7AF7 FFF8 9F40 2407 A35A F063 D3E1
Enter component #3: U EE8D 4F9E C8B2 ADF4 9CD2 F0D2 7F5C 95C5
Encrypted key: U 104C 4216 A751 FEEE FF55 698B 26C5 7789
Key check value: BA0F C3
```

Chapter 5. Dynamic Key Exchange

Abstract

TODO: abstract

Overview

TODO: Why dynamic key exchange?

Zone PIN Key (ZPK)

The Zone PIN Key (ZPK) is a data encrypting key. It is used to encrypt the data that is transmitted in a security zone. For transfer between sites, the ZPK is encrypted under the ZMK. When stored locally (e.g. on the application server), the ZPK is encrypted using one of the LMK Keys.

ZPKs have a key type code of 001 (encrypted under LMK pair 06-07 with a variant of 0).

TODO: See “Laboratory 05 – Creating a ZPK” for hands on experience creating a ZPK.

Example 5.1. Dynamic Key Exchange

Assume that the ZMK presented in the previous example has been exchanged between two parties. One party, then, generates a random ZPK equal to:

```
ADD3B5C7B576D3AE 38B90B7C0EB67A7C, KCV = CB59C0
```

The party then encrypts this ZPK under the ZMK to safely transmit this to the other party. The ZPK under the ZMK is:

```
C9A62E96ADFB52A7 815BE8D7E730B24E, KCV = CB59C0
```

Key Translation

In our previous example, the value C9A62E96ADFB52A7 815BE8D7E730B24E represents the randomly created ZPK encrypted under the previously created ZMK. But imagine that one of the parties that have exchanged this ZPK needs to transmit it to another party with which they share a different ZMK which we’ll call ZMK2 . To properly transmit the ZPK to the other zone that is secured with ZMK2, the ZPK has to be:

- Decrypted under ZMK.
- Encrypted under ZMK2.

This process is called key translation. Key translation does not happen with specific key types as the ZMK that was used in the previous example but is a more general process - for example it is possible to translate a key from encryption under the ZMK to encryption under an LMK.

Key translation always takes place within the HSM to avoid exposing the clear value of the key being translated.

Translating a ZPK

(between ZMK encryption and LMK encryption)

The HSM provides functionality to translate a ZPK between ZMK and LMK. This is used when a ZPK is received by Site B. In this case, Site A sends the ZPK encrypted with the ZMK to Site B. Site B translates the ZPK to LMK encryption.

The ZPK under LMK encryption is used for encrypting/decrypting the data sent between the sites.

TODO: insert diagram

Exercises

Creating a Zone PIN Key (ZPK)

In this exercise, we create a Zone PIN Key (ZPK) using console commands.

When prompted for the ZMK, use the encrypted ZMK value from the section called “Creating a Zone Master Key (ZMK)”.

```
KG
Key length [1,2,3]: 2
Key Type: 001
Key Scheme (LMK): U
Key Scheme (ZMK) [ENTER FOR NONE]: X
Enter encrypted ZMK [ENTER FOR NONE]: U 104C 4216 A751 FEEE FF55 698B 26C5 7789
Enter ZMK check value [ENTER TO SKIP CV TEST]:
Key under LMK: U 8586 51EC 83AF CA66 8175 804F 5B7D CD6B
Key encrypted for transmission: X BAA5 18AA D10D 28A2 D32A 5688 317F 44EB
Key check value: 6543 F4
```

Chapter 6. PIN block creation (clear PIN blocks).

Chapter 7. PIN block encryption and security zones. ZPKs and TPKs.

Chapter 8. PIN translation.

Chapter 9. MACing.

Chapter 10. CVV/CVV2/iCVV.

Part II. Inside the Thales Simulator

TODO: Intro to this part.

Chapter 11. Codebase overview

Chapter 12. Implementing a new host command

Appendix A. Introductory Books in Cryptography

Index