Hardware Security Modules:
Attacks and Secure Configuration

Graham Steel
Secure Hardware History

**Military:**
WW2 Enigma machines
- captured machines used to help break codes
NSA devices with explosive tamper resistance
- 
http://www.nsa.gov/about/cryptologic_heritage/museum/

**Commercial:**
IBM: Cryptoprocessors for mainframes
- tamper-resistant switches on case
ATMs (cash machines)
- Encrypted PIN Pads (EPPs) and Hardware Security Modules (HSMs)
Cryptographic Smartcards
- chip contains cryptoprocessor and keys in memory
- used in SIM cards, credit cards, ID cards, transport...

Authentication tokens
- generate One-Time Passwords, sometimes USB connection

Trusted Platform Module (TPM)
- now standard (but unused) in most PC laptops

The future..
- Secure Elements in mobile phones, cars, ...
Example - Cash Machine Network

- Introduced in the UK in the late 1960s
- First modern machines (with DES) in the 70s and 80s
- More than 2 million ATMs worldwide
- Network is now global and ubiquitous (at least in cities)
Simplified Network Schematic

ATM

Maestro UK

SocGen

HSBC
HSMs

- Manufacturers include IBM, nCipher, Thales, Utimaco, HP
- Cost around $20 000
A Word About Your PIN

IBM 3624 method:

1. Write account number (PAN) as 0000AAAAAAAAAAAAA

2. 3DES encrypt under a PDK (PIN Derivation Key), decimalise first digits

3. PIN = IPIN + Offset (modulo 10 each digit)

NB: Offset NOT secure!
API attack example: VSM (Bond, 2001)
Example: Print Customer PIN

Host → HSM : PAN, \{ PDK1 \} Km
HSM → Printer : \{ PAN \} PDK1
Example: Send PDK to Terminal

Host → HSM : \{PDK1\}_Km, \{TMK1\}_Km
HSM → Host : \{PDK1\}_TMK1
Terminal Comms Key

\[ \text{KM} \xrightarrow{\{ \text{MSG} \}_{TC}} \text{TMK1} \]
Managing Key Types

Host machine

\{ \text{TMK1} \} \_\text{KM}

\{ \text{PDK1} \} \_\text{KM}

\{ \text{TC1} \} \_\text{KM2}

VSM

\_\text{KM}

\_\text{KM}

\_\text{KM2}
Example: Enter TC key

TC1 → KM, KM2

HSM → Host: {TC}

Host → HSM: TC

VSM

{TC1} KM2

Km2
Example: Send TC to Terminal

Host $\rightarrow$ HSM : \{ TC \} $Km_2$, \{ TMC1 \} $Km$

HSM $\rightarrow$ Host : \{ TC \} $TMK1$
Attack - Step 1

Spy → HSM : PAN
HSM → Spy : \{ PAN \} _{KM2}

Graham Steel - HSM Attacks and Secure Configuration
Spy $\rightarrow$ HSM :  $\{ \text{PAN} \}^\text{Km2}, \{ \text{PDK1} \}^\text{Km}$

HSM $\rightarrow$ Host :  $\{ \text{PAN} \}^\text{PDK1}$
IBM 4758 CCA API
The Common Cryptographic Architecture (CCA) API uses the same ‘master key’ architecture as the VSM.

However, the (patented) type system is much richer.

Before encrypting a working key, the master key is XORed against a ‘control vector’ indicating the type of the key.

The control vectors are public values (they can be found in the programmers’ manual), but the master key is secret.

Control vectors can be composite, i.e. they may consist of a number of values XORed together.
CCA Types - 2

Host machine

{ TC1 } km + data

{ PDK1 } km + pin

HSM

km
CCA API - Examples

Encrypt Data:

Host → HSM : \{ d1 \} \text{km} \oplus \text{data}, \text{message} \\
HSM → Host : \{ \text{message} \} \text{d1}

Verify PIN:

Host → HSM : \{ \text{PINBlock} \} _{p1}, \text{PAN}, \{ \text{pdk1} \} \text{km} \oplus \text{pin}, \text{OFFSET}, \{ \text{p1} \} \text{km} \oplus \text{ipinenc} \\
HSM → Host : yes/no
Bootstrapping

A common problem in the use of secure hardware

How to get the initial secrets onto the device (or encrypted by the device’s master key) in a secure way?

A common solution is ‘separation of duty’: several members of staff are given individual parts of a secret.

Each individual part is worthless, so only collusion between several staff members can expose the secret.
Importing Key Parts

Separation of duty between e.g. 2 security officers

Key \( k = k_1 \oplus k_2 \)

\[
\begin{align*}
\text{Host} & \rightarrow \text{HSM} : k_1, \text{TYPE} \\
\text{HSM} & \rightarrow \text{Host} : \{ k_1 \} \ km \oplus \text{kp} \oplus \text{TYPE}
\end{align*}
\]

\[
\begin{align*}
\text{Host} & \rightarrow \text{HSM} : \{ k_1 \} \ km \oplus \text{kp} \oplus \text{TYPE}, k_2, \text{TYPE} \\
\text{HSM} & \rightarrow \text{Host} : \{ k_1 \oplus k_2 \} \ km \oplus \text{TYPE}
\end{align*}
\]

This is a tedious and expensive process, so usually used to import a ‘key encrypting key’ \( (\{ \text{KEK} \} \ km \oplus \text{imp}) \)
Importing Encrypted Keys

Exported from another 4758 encrypted under KEK ⊕ TYPE

Key Import:

Host → HSM : { KEY1 } KEK⊕TYPE, TYPE, { KEK } km⊕imp
HSM → Host : { KEY1 } km⊕TYPE
PIN derivation key: \{ pdk \} \text{kek} \oplus \text{pin}

Have key part \{ \text{kek} \oplus k2 \} \text{km} \oplus \text{imp} \oplus \text{kp} for known k2

Host $\rightarrow$ HSM : \{ \text{kek} \oplus k2 \} \text{km} \oplus \text{kp} \oplus \text{imp}, k2 \oplus \text{pin} \oplus \text{data}, \text{imp}$

HSM $\rightarrow$ Host : \{ \text{kek} \oplus \text{pin} \oplus \text{data} \} \text{km} \oplus \text{imp}$
Key Import

\[
\begin{align*}
\text{Host} & \rightarrow \text{HSM} : \{ pdk \} \_\text{kek} \oplus \text{pin} \oplus \text{data} , \\
\text{HSM} & \rightarrow \text{Host} : \{ pdk \} \_\text{km} \oplus \text{imp}
\end{align*}
\]

Encrypt data

\[
\begin{align*}
\text{Host} & \rightarrow \text{HSM} : \{ pdk \} \_\text{km} \oplus \text{data} , \text{pan} \\
\text{HSM} & \rightarrow \text{Host} : \{ \text{pan} \} \_\text{pdk} (= \text{PIN}!)
\end{align*}
\]
IBM Recommendations

Published in response to Bond’s attacks

1. Use asymmetric key crypto for key import – 2 officer protocol to generate key pair at destination, transfer public key to source – `PKA_Symmetric_Key_Import` command

2. More access control – security officers access fewer commands

3. Procedural controls to check entered key parts

2 and 3 verified in a few seconds, but 1 has a simple attack.
Attack on 1 (Cortier, Keighren & S. ’07)

\[
\{\text{kek} \cdot \text{IMP}\}_{PK} \rightarrow \{\text{kek}\}_{\text{KM} \oplus \text{IMP}} \\
\{k \cdot \text{EXP}\}_{PK} \rightarrow \{k\}_{\text{KM} \oplus \text{EXP}} \\
\{\text{pdk}\}_{\text{kek} \oplus \text{PIN}, \text{PIN}} \rightarrow \{\text{pdk}\}_{\text{KM} \oplus \text{PIN}} \\
\{\text{pdk}\}_{\text{KM} \oplus \text{PIN}, \text{PIN}} \rightarrow \{\text{pdk}\}_{k \oplus \text{PIN}}
\]

PKA Symmetric Key Import
PKA Symmetric Key Import
Key Import
Key Export
Summary of First half

- Secure hardware is more and more prevalent
- The API of the hardware is a security critical part of design
- Have seen attacks on VSM, CCA
- In the next half we’ll look at specific attacks on PIN processing
Further reading


V. Cortier, G. Keighren and G. Steel, *Automatic Analysis of the Security of XOR-based Key Management Schemes*, TACAS ’07

The Analysis of Security APIs Workshop,
http://www.lsv.ens-cachan.fr/~steel/asa/
Introduction to PIN Processing

Processing of PINs in the international cash machine network is one of the oldest and most widespread uses of cryptographic hardware.

International standards (ISO 9564, ANSI X9.8) and de-facto standards (e.g. Visa’s requirements documents) regulate the network.

According to ANSI X9.8 secure hardware must be configured so that “The system shall not be capable of being used or misused to determine a PIN by exhaustive trial and error”.
Verizon Breach Report 2008

Released April 2009

“While statistically not a large percentage of our overall caseload in 2008, attacks against PIN information represent individual data-theft cases having the largest aggregate exposure in terms of unique records,”

“In other words, PIN-based attacks and many of the very large compromises from the past year go hand in hand.”

“We’re seeing entirely new attacks that a year ago were thought to be only academically possible,”

“What we see now is people going right to the source [..] and stealing the encrypted PIN blocks and using complex ways to un-encrypt the PIN blocks.”

(Quotes from Wired Magazine interview with report author, Bryan Sartin)
Known Major Breaches

2008 Feb Citibank $3.6M
2008 Nov RBS worldpay $9.4M
2011 May FIS $13M
2012 December RAKBANK $11M
2013 February Bank of Muscat $45M

Symatec report: "tens of millions” lost to ATM heists in 2012 just in Europe:
Simplified Network Schematic
PIN Processing API

We can see that the PIN processing API of the HSM will have to (at least):

- Translate PINs
- Verify PINs
- Generate PINs

We’ll look at these APIs (and attacks on them).

There are also functions to

- Change PINs
- Print PINs

which we won’t have time for today
Deriving a PIN: IBM 3624 Method

1. Write account number (PAN) as 0000AAAAAAAAAAAAAA
2. 3DES encrypt under a PDK (PIN Derivation Key)
3. Take 4 leftmost hexadecimal digits of result
4. Decimalise using a mapping table (‘dectab’) 
   
   0123456789ABCDEF  
   0123456789012345 

5. PIN = IPIN + Offset (modulo 10 each digit)
PIN Verification

Verify PIN:

\[ \{\text{PIN}\}_K, \text{PAN, Dectab} \rightarrow \text{Offset} \]

yes/no

K, PDK
The Mastermind Game

- Invented by the Israeli postmaster and telecommunications expert Mordecai Meirowitz in 1970;
- 4 pegs from 6 possible colors, duplicates are allowed.
- The codemaker chooses a sequence of 4 pegs, the codebreaker has to guess it.
- Goal: Minimize the number of guesses.
Decimalisation Table Attack (Clulow '02, Bond & Zeilinski '03)

Suppose in a hacked switch, an attacker has a set
\{PIN\}_K, PAN, Dectab, Offset that verifies PIN is correct

Original Dectab
0123456789ABCDEF
0123456789012345

Dectab’
0123456789ABCDEF
1123456789112345

Repeat verification command with Dectab’
Successful verification indicates no 0s in PIN
More dectab attack

To find the 0s, try changing the offset

<table>
<thead>
<tr>
<th>Attacker set offset</th>
<th>Result from HSM</th>
<th>Knowledge of PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Incorrect PIN</td>
<td>????</td>
</tr>
<tr>
<td>0010</td>
<td>Incorrect PIN</td>
<td>????</td>
</tr>
<tr>
<td>0100</td>
<td>Incorrect PIN</td>
<td>????</td>
</tr>
<tr>
<td>1000</td>
<td>Incorrect PIN</td>
<td>????</td>
</tr>
<tr>
<td>0011</td>
<td>Incorrect PIN</td>
<td>????</td>
</tr>
<tr>
<td>0101</td>
<td>Correct PIN</td>
<td>?0?0</td>
</tr>
</tbody>
</table>
What’s the Best Strategy for the Dectab Attack?

Allows us to assess the seriousness of the attack

- Bond’s original scheme (2003) : 16.5 guesses on average
  - Go through all the different dectabs, get a list of digits
  - For each digit, use the offset to find out where it is

  - Find a digit, use the offset to find out where it is, repeat

- Focardi + Luccio (2010) : 13.463 guesses
  - Start by trying two digits at a time

(not known if this is optimal, but $\log_2 10000=13.362$ to 3dp)
PIN Block Formats

Padding formats to encode a PIN for encryption
Must be 64 bits long for 3DES
Must allow PIN length to be determined (e.g. in Italy, 5 digits)

VISA format 3

PPPPFFFFFFFFFFFFFFFF

Same PIN block for different users with same PIN
ISO 9564 Formats

Diversify PIN blocks using PAN

ISO 9564 format 0

04PPPPFFFFFFFFFFFFF
0000AAAAAAAAAAAAAA

Two lines are XORed together before encryption
Diversify blocks using randomness
ISO 9564 format 1

14PPPPRRRRRRRRRRR

Requires source of randomness
The Translate API

Translate PIN:

\{ \text{PINBlock} \} \quad p_1, \quad \text{Format1, Format2, [PAN]} \quad \rightarrow \quad \text{HSM}\quad p_1, p_2

\{ \text{PINBlock} \} \quad p_2 \quad \leftarrow

If the decrypted block is not in the correct format, an error is returned.
ISO-0 Reformatting attack

(Clulow, 2003)

\[04PPPPFFFFFFFFFF\]
\[0000AAAAAAAAAAAAAAA\]

Suppose attacker calls ‘Translate’ function with modified PAN:
first A digit \(A' = A \oplus 8\)
Error check \(0 \leq P \leq 9\) leaks information
Attacking digit 3 of the PIN by changing digit 1 of the PAN

<table>
<thead>
<tr>
<th>Change to digit 1</th>
<th>Result from HSM</th>
<th>Knowledge of PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ok</td>
<td>$0 \leq P_3 \leq 9$</td>
</tr>
<tr>
<td>⊕8</td>
<td>Format error</td>
<td>$0 \leq P_3 \leq 7$</td>
</tr>
<tr>
<td>⊕12</td>
<td>ok</td>
<td>$4 \leq P_3 \leq 5$</td>
</tr>
</tbody>
</table>

Limitations:
Always get two possible values for the digit
Only works for digits 3 and 4
Extended Reformat Attack

Masquerade ISO-0 as VISA format 3

04PPPPFFFFFFFFFFFF
0000000000000000
0604PPPPFFFFFFFF

Now two possible errors: non-decimal PIN digit, or padding digit ≠ F
Can now uniquely determine digits
### Example

<table>
<thead>
<tr>
<th>Change to digit 1</th>
<th>Result from HSM</th>
<th>Knowledge of PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ok</td>
<td>$0 \leq P_3 \leq 9$</td>
</tr>
<tr>
<td>$\oplus 8$</td>
<td>Format error</td>
<td>$2 \leq P_3 \leq 7$</td>
</tr>
<tr>
<td>$\oplus 12$</td>
<td>ok</td>
<td>$4 \leq P_3 \leq 5$</td>
</tr>
</tbody>
</table>

Now masquerade the block as VISA

| $\oplus 10$       | ok                     | $P_3 = 5$               |

Note reformatted block has digits shifted two places to the right - so attack can be repeated on PIN digits 1 and 2
Generate Encrypted PIN:

PAN, dectab, offset → Format

{ PINBlock } p1 ← p1, PDK
Statistical Attack

(see Bond & Clulow 2004)

First calculate a table of encrypted PINs for each offset 0000-9999
Now for a fixed offset, generate EPB for random PDKs
Note that with a fixed standard dectab 0123456789012345, digits 0-5 are twice as likely as 6-9.
After generating enough EPBs, we can ‘line up’ distribution and decrypt all the PINs
(Köpf & S.)

Bond and Clulow quote “2000-10000 calls” for the learning part of their attack.

With Boris Köpf we implemented Maximum Likelihood Estimation for the attack.

On average 216 calls suffice.
You can download an open source simulator of one of the most widely-used payment HSM, the Thales 8000. More than 70% of transactions go through a 8000 series HSM.

https://thalessim.codeplex.com/
Further Reading


