

# Hardware Security Modules and their APIs

---

Honza Krhovják  
Vašek Matyáš

Faculty of Informatics  
Masaryk University  
Czech Republic

# Basic terminology

---

- ❑ Hardware security modules (HSM)
  - Cryptographic coprocessors
  - Cryptographic accelerators
  - Cryptographic smartcards
- ❑ Host devices, API
- ❑ Security & attacks on HSMs
  - Physical attacks
  - Logical attacks
    - ❑ Attacks on and with API
  - We are not interested in any form of DoS attacks!
- ❑ Top-level crypto keys – always stored inside HSM
  - Other keys can be stored outside HSM encrypted by these
- ❑ Trusted platform modules (TPM)



# Attacks on and with API

---

- Examples of commonly used API
  - Public Key Cryptographic Standard (PKCS) #11
  - Common Cryptographic Architecture (CCA)
- Three major problems of cryptographic API
  - Insufficient ensuring integrity of keys
    - Problems with backward compatibility (e.g., support of DES/RC2)
    - Meet in the Middle Attack, 3DES Key Binding Attack, Conjuring Keys From Nowhere ...
  - Insufficient checking of function parameters
    - Banking API & working with PINs => PIN recovery attacks
    - Decimalisation Table Attacks, ANSI X9.8 Attacks, EMV Secure Messaging Attacks ...
  - Insufficient enforcing of security policy
    - PKCS #11 – only set of functions, designed for one-user tokens

# Known Key Attack I

---

- Surprising attack on Visa Security Module (VSM)
  - VSM mostly installed in banks and first generation ATMs
  - Keys generated, securely encrypted inside VSM by master key (KM), and stored outside VSM
    - To perform key generation API contains command `GenerateKeyPart()`
  - Shared terminal key (KMT) in ATM established manually
    - XORed from at least two parts (Dual Control Policy) by using API command `CombineKeyParts()`
  
- Correctly entered enciphered key parts  $K_{MT1}$  &  $K_{MT2}$ 
  - $\text{CombineKeyParts}(E_{KM}(KMT1), E_{KM}(KMT2)) = E_{KM}(KMT1 \text{ xor } KMT2) = E_{KM}(KMT)$

# Known Key Attack II

---

- Attack misuses a lack of separation of key parts
  - You can enter the same (e.g., first) part twice
  - $\text{CombineKeyParts}(E_{KM}(KMT1), E_{KM}(KMT1)) = E_{KM}(KMT1 \text{ xor } KMT1) = E_{KM}(0)$
  
- The attacker (e.g., malicious banking programmer) now know the terminal key of ATM
  - Can be misused to export Pin Derivation Key (KPD)
  - KPG uses bank to PIN generation and verification
    - PIN verification in ATM only if network is down
  - With knowledge of KPD and personal account number (PAN) can attacker generate PIN for arbitrary account

# A “Two-time Type” Attack I

---

- ❑ Separating of different key types in VSM ensured by enciphering with different master key (KM)
  - VSM support 9 types and have thus 9 master keys
  
- ❑ Terminal key (KMT) often used to protect transfer of other secret communication keys (KC)
  - Short-term keys used to protect ATM communication
    - ❑ No restriction for enciphering/deciphering
  - Existence of function `InsertKey()` for entering clear KC
    - ❑ KC automatically encrypted by particular KMT (say KMC)
  - Existence of function `ReEncrypt()` for reencrypting KC by different KMT (say KMD)

# A “Two-time Type” Attack II

---

- Correct calls of these functions
  - Entering of clear KC
    - $\text{InsertKey}(\text{KC}) = E_{\text{KMC}}(\text{KC})$
  - Reencrypting of KC by KMD
    - $\text{ReEncrypt}(E_{\text{KMC}}(\text{KC}), E_{\text{KM}}(\text{KMD})) = E_{\text{KMD}}(\text{KC})$
- PIN Generating Key (PGK) and various terminal keys (e.g., KMC, KMD, ...) have the same type
  - This imply that they are enciphered by the same KM
- The attack allows easily derive PIN from PAN
  - $\text{InsertKey}(\text{PAN}) = E_{\text{KMC}}(\text{PAN})$
  - $\text{ReEncrypt}(E_{\text{KMC}}(\text{PAN}), E_{\text{KM}}(\text{KPD})) = E_{\text{KPD}}(\text{PAN}) = \text{PIN}$
- Observation: encrypted data are still sensitive

# Meet in the Middle Attack I

---

- Attack is based on three independent flaws
  - Poorly designed key types separation
  - No upper limits on key generation
  - Small length of single DES key
- Attack technique is follows
  - HSM capable to generate keys very fast
    - More then ten thousand keys after a few minutes
  - Attacker use  $2^{16}$  keys to encrypt(& store) the same data
    - Typically encrypted block of binary zeros (test vector)
  - Attacker perform exhaustive parallel search for keys
    - Each key is used to encrypt these data and compared with all  $2^{16}$  stored ciphertexts
    - Equality imply that one key is successfully found

# Meet in the Middle Attack II

---

- Keyspace for search is roughly  $2^{56}/2^{16} = 2^{40}$
- In the case of VSM is possible to generate all  $2^{16}$  keys as terminal keys
  - Compromised key is thus also a terminal key
  - 8 of 9 types of keys can be compromised by this attack due to bad key type separation
- Efficient variant of attack allows to crack top-level master key of Prism HSM
  - After loading each part of master key is returned test vector encrypted by the current master key
  - No limits for entering key parts
  - Attacker gather  $2^{16}$  encrypted vectors and search the key

# Conjuring Keys From Nowhere

---

- ❑ Unauthorized generating of keys stored outside HSM
  - Random value of encrypted key is given to HSM
    - ❑ Older HSMs used this technique to legitimate key generation
    - ❑ Today is it considered as attack
      - Even modern HSMs (e.g., IBM 4758 with CCA) are vulnerable
  - After decryption is the value of key also random
    - ❑ In the case of DES has with probability  $1/2^8$  good parity
      - DES key is stored with odd parity – LSB in each octet is parity bit
    - ❑ In the case of two-keyed 3DES-2 has a good parity with probability  $1/2^{16}$  (and this is still achievable)
  - These keys can served to form more complicated attacks
- ❑ The defense lies in carefully designed key formats  
=> e.g., add before encryption checksum & timestamp

# 3DES Key Binding Attack

---

- ❑ Misuse insufficient binding of 3DES key parts
- ❑ Applicable on IBM 4758 with CCA
  - Attacker generate large number of 3DES-2 keys with the same parts
  - By using Meet in the Middle Attack find two of them
    - ❑ Searching in keyspace  $2^{41}$
  - Exchange of key halves lead to creating two 3DES keys with different halves
    - ❑ If generated keys were export keys then also the found key is an export key
    - ❑ This key can be used to export all exportable keys
  - Exchange of known key half with half of no exportable key lead to decreasing its keyspace from  $2^{112}$  to  $2^{56}$

# PIN Generation and Verification

---

- Terminology
  - Personal Identity Number (PIN) & Account Number (PAN)
  - Clear PIN block (CPB); Encrypted PIN block (EPB)
  
- Techniques of PIN generation and verification
  - IBM 3624 and IBM 3624 Offset
    - Based on validation data (e.g., account number – PAN)
    - Validation data encrypted with PIN derivation key
    - The result truncated, decimalised => PIN
    - IBM 3624 Offset – decimalised result called IPIN (Intermediate PIN)
    - Customer selects PIN:  $\text{Offset} = \text{PIN} - \text{IPIN} \pmod{10}$
  - Verification process is the same
    - result is compared with decrypted EPB (encrypted PIN from cash-machine)

# PIN Verification Function

---

- ❑ Simplified example of verification function and its parameters:
  1. PIN (CPB) encryption/decryption key
  2. PIN derivation key – for PIN generation process
  3. PIN-block format
  4. Validation data – for PIN extraction from EPB (e.g., PAN)
  5. Encrypted PIN-block
  6. Verification method
  7. Data array – contains decimalisation table, validation data and offset
- ❑ Clear PIN is not allowed to be a parameter of verification function!

# PIN Verification – IBM 3624 Offset

---

## □ Inputs – (4-digit PIN)

- PIN in EPB is 7216 (delivered by ATM)
- Public offset (typically on card) – 4344
- Decimalisation table – 0123 4567 8901 2789
- Personal Account Number (PAN) is  
4556 2385 7753 2239

## □ Verification process

- PAN is encrypted => 3F7C 2201 00CA 8AB3
- Truncated to four digits => 3F7C
- Decimalised according to the table => 3972
- Added offset 4344, generated PIN => 7216
- Decrypt EPB and compare with the correct PIN

# Decimalisation Table Attacks I

---

- Attacks utilising known PINs
  - Assume four-digit PINs and offset 0000
  - If decim. table (DT) is 0000 0000 0000 0000  
generated PIN is always 0000
  - PIN generation function with *zero* DT outputs EPB with PIN 0000
  - Let  $D_{orig} = 0123\ 4567\ 8901\ 2345$  is original DT
  - $D_i$  is a *zero* DT with "1" where  $D_{orig}$  has  $i$   
e.g.,  $D_5 = 0000\ 0100\ 0000\ 0001$
  - The attacker calls 10x verification function with EPB of 0000 PIN and with  $D_0$  to  $D_9$
  - If  $i$  is not in PIN, the "1" will not be used and verification against 0000 will be successful

# Decimalisation Table Attacks II

---

## □ Results

- All PIN digits (but not their order) are discovered
- PIN space reduced from  $10^4$  to 36
  - Worst case for four digit PINs with three different digits

## □ Extended attack without known PINs

- Assume, that we obtain customers EPB with correct PIN
- $D_i$  are DTs containing  $i-1$  on positions, where  $D_{orig}$  has  $i$ , e.g.,  $D_5 = 0123 \ 4\mathbf{4}67 \ 8901 \ 234\mathbf{4}$
- Verification function is called with intercepted EPB and  $D_i$
- Position of PIN digits is discovered by using *offset* with digits incremented individually by 1
  - Bold "4" changes to "5"

# DT Attacks – Example

---

- Let PIN in EPB be 1492, offset is 1234
  - We want to find position of "2"
  - Verification function with  $D_2$  results in  $1491 \neq 1492$   
=> fails
  - Offsets 2234, 1334, 1244, 1235 increment resulting generated PIN (2491, 1591, ...)
  - Eventually the verification is successful with the last offset => 2 is the last digit
  
- To determine four-digit PIN with different digits is needed at most 6 calls of verification function

# Clear PIN Blocks (CPB)

---

- Code Book Attacks => PIN-block formats
    - CPB in fact describes padding of PIN before its symmetric encryption in ECB mode
  
  - ECI-2 format for 4 digits PINs
    - ECI-2 CPB = pppprrrrrrrrrrrrrrr
  - Visa-3 format for 4–12 digits PINs
    - Visa-3 CPB = ppppFxxxxxxxxxxxx
  - **ANSI X9.8 format for 4–12 digits PINs**
    - $P_1 = ZlppppffffffffffF$
    - $P_2 = ZZZZaaaaaaaaaaaaa$
    - ANSI X9.8 CPB =  $P_1 \text{ xor } P_2$
- p – PIN digit*  
*r – random digit*  
*x – arbitrary, all the same*  
*F – 0xF digit*  
*Z – 0x0 digit*  
*l – PIN length*  
*f – either “p” of “F”*  
*a – PAN digit*

# ANSI X9.8 Attacks I

---

- Attacking PAN with translation & verification functions – input parameters (key K, EPB, PAN)
  - Functions decrypt EPB & extract PIN  
 $CPB \text{ xor } P_2 = 04ppppFFFFFFFFFFFF \Rightarrow PIN = pppp$
  - Extraction tests PIN digits to be 0–9!
  - If a digit of PAN is modified by  $x$ 
    - $P_2' = P_2 \text{ xor } 0000x000000000000$
    - $CPB \text{ xor } P_2' = \begin{array}{l} 04ppppFFFFFFFFFFFF \text{ xor} \\ \text{xor} \\ 0000x000000000000 \end{array}$   
it means that  $PIN = pppp \text{ xor } 00x0$
    - If  $p \text{ xor } x < 10$  function ends successfully, otherwise function fails

# ANSI X9.8 Attacks II

---

- The sequence of (un)successful function calls can be used by attacker to identify  $p$  as a digit from set  $\{p, p \text{ xor } 1\}$
  - For example if PIN digit is 8 or 9, then this sequence will be PFFFFFFFFPPPPPPP, where P is PASS, F is FAIL and  $x$  is incremented from 0 to 15
- 
- Only last two PIN digits can be attacked
  - PIN space is reduced from  $10^4$  to 400
  - This attack can be extended to all PIN digits

# ANSI X9.8 Attacks

## Collision Attack (Basic Idea)

---

- Assuming well designed API (e.g., DT is fixed)
- Attack allows to partially identify last two PIN digits
  - Basic idea (simple example with one-digit PIN&PAN)

PAN	PIN	xor	EPB	PAN	PIN	xor	EPB
0	0	0	21A0	7	0	7	2F2C
0	1	1	73D2	7	1	6	345A
0	2	2	536A	7	2	5	0321
0	3	3	FA2A	7	3	4	FF3A
0	4	4	FF3A	7	4	3	FA2A
0	5	5	0321	7	5	2	536A
0	6	6	345A	7	6	1	73D2
0	7	7	2F2C	7	7	0	21A0
0	8	8	<b>4D0D</b>	7	8	F	<b>AC42</b>
0	9	9	<b>21CC</b>	7	9	E	<b>9A91</b>

- Attacker knows for each PAN only the set of EPBs

# EMV Secure Messaging Attacks I

---

- CCA functionality extended to support EMV
  - Secure messaging is EMV form of key export
    - Related control vector (CV) is SECMSG
  - Various message formats (due different manufacturers)
    - Message template is block aligned plaintext value
    - Offset pointer to template (points to the place where store key data) can be non-block aligned
- Encryption oracle for keys with SECMSG CV
  - ECB and CBC modes (without padding) allows ciphertext truncation (entire last block can be removed)
  - Extending template by zero block and setting offset to this block => the key will be in the last block
    - After its removing we get only encrypted template

# EMV Secure Messaging Attacks II

---

- Cracking exportable keys by using non-aligned offsets and encryption oracle
  - Create 256 plaintext messages/templates  
`0x0000000000000000xx` with `xx` from `00` to `ff`
  - Encrypt this 256 templates by encryption oracle
  - Perform API call with template of zeros, offset value 7, and exporting key KE
  - Comparing first encrypted block with 256 ciphertexts yields the first byte of key KE (denoted `uu`)
  - Create new 256 plaintext messages/templates  
`0x0000000000000000uuxx` with `xx` from `00` to `ff`
  - Repeating process with offset 6 yields second byte, etc.
- Entire `k`-byte key found after `k · 256` queries

# PKCS #11 Attacks I

---

## □ Symmetric Key Attacks

### ■ 3DES Key Binding Attack

- 3DES-2 key  $K$  with halves  $K1$  and  $K2$
- Export as  $E_{KEK}(K) = (E_{KEK}(K1), E_{KEK}(K2))$

### ■ Key Separation Attack

- Conflict setting of key properties
  - Key encrypt vs. data decrypt

### ■ Weaker Key/Algorithm Attack

- Encrypting by short keys RC2 (40 bits) or DES (56 bits)

### ■ Related Key Attack

- 3DES-3 key  $K1 = (KA, KB, KC)$  and  $K2 = (KA \text{ xor } \text{DELTA}, KB, KC)$
- $$P' = D_{KA \text{ xor } \text{DELTA}}(E_{KB}(D_{KC}(E_{KC}(D_{KB}(E_{KA}(P))))) =$$
$$= D_{KA \text{ xor } \text{DELTA}}(E_{KA}(P))$$

# PKCS #11 Attacks II

---

- Reduced Key Space Attack
  - Function `C_DeriveKey()` create new key from existing key by using successive series of its bits
  - This “feature” can be misused to reduce keyspace
  - Attacker create from 56bit single DES key 40bit RC2 key and by brute-force find its value
  - With this knowledge find rest of single DES key bits
  
- Public Key API Attacks
  - Small Public Exponent with No Padding Attack
  - Trojan Public Key Attack
  - Trojan Wrapped Key Attack
  - Private Key Modification Attack



# Trusted platform modules

---

- ❑ TPM chip typically based on similar technology as secure microcontrollers for smart cards
- ❑ Fundamental features and functions of TPM
  - Trusted measurement, storage, and reporting
    - ❑ Complete integrity snapshot of HW&FW&SW components necessary for performing secure boot sequence
    - ❑ Three roots of trust serves to anchor a certificate verification chain that is unique to a given system
  - Identity/attestation (by external entities)
    - ❑ Shielded locations, protected capabilities, and roots of trust
- ❑ Shielded locations (memory, register, ...) for sensitive data
  - Storage of crypto keys to authenticate reported measurement
  - Platform configuration registers (PCRs) to protect integrity measurements

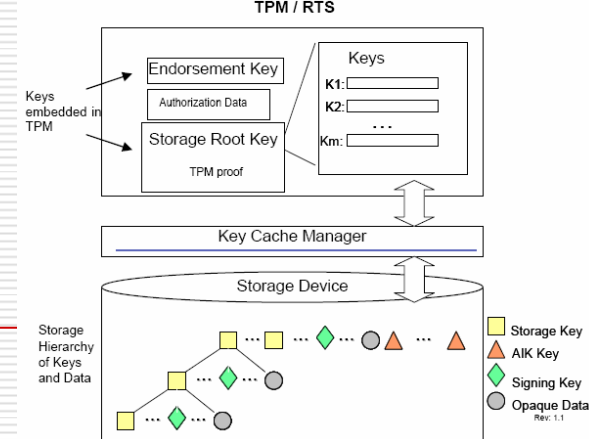
# Roots of trust

## □ Three basic roots of trust

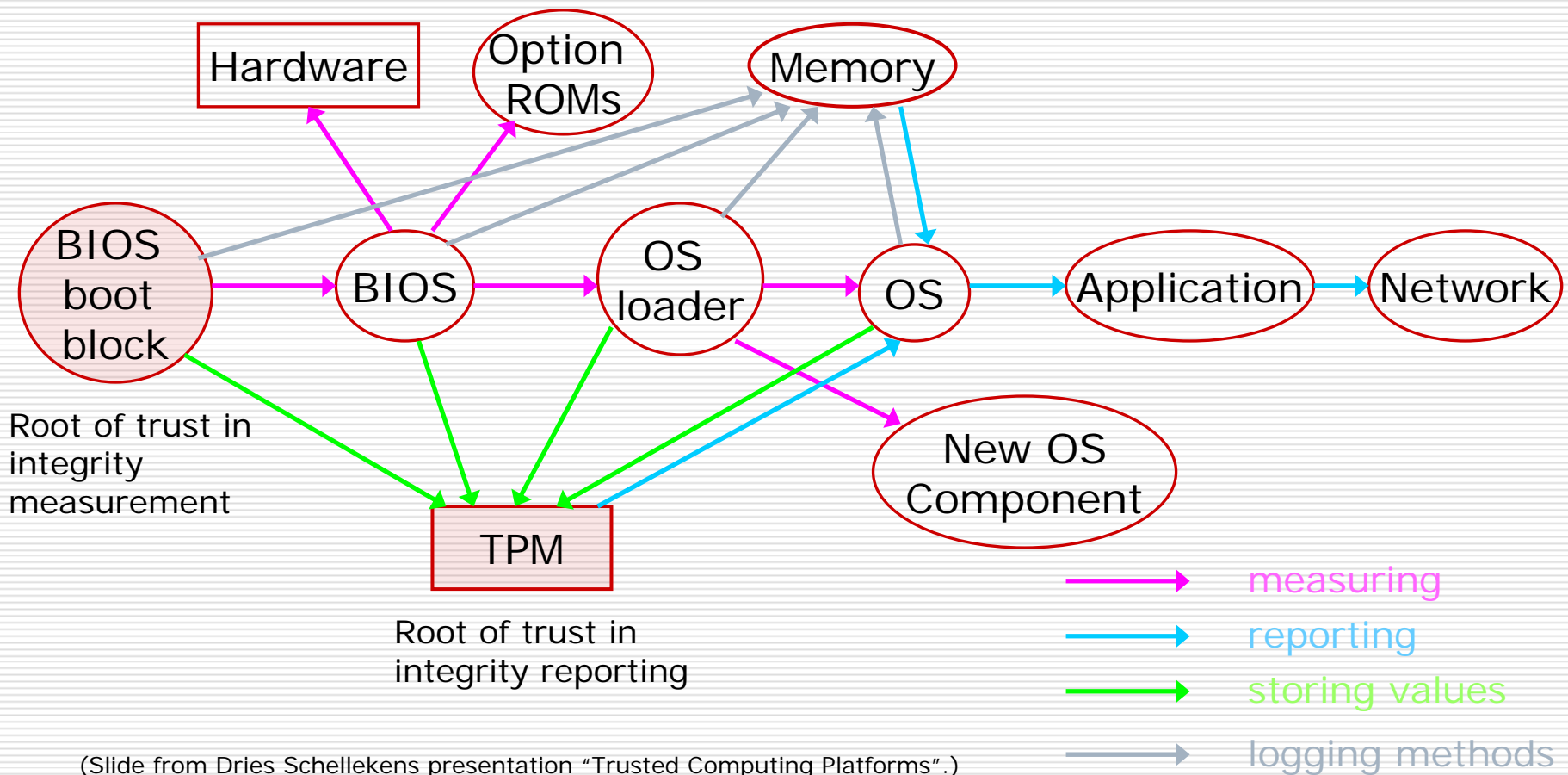
- RTS: root of trust of storage (for external objects)
  - Endorsement key (EK) unique for TPM (= > platform)
- RTR: Root of trust for reporting (and attestation)
- RTM: Root of trust measurements

## □ Core root of trust measurement (CRTM)

- First executed code is initialization code
- Correctness and integrity is critical
- Two basic variants
  - CRTM = trusted BIOS
  - CRTM = BIOS Boot Block (without BIOS POST)
- CTRM called static S-CTRM (spec TPM 1.1),  
new independent dynamic D-CTRM (spec TPM 1.2)



# Secure bootstrap



(Slide from Dries Schellekens presentation "Trusted Computing Platforms".)

# TPM authorization protocols

---

- Commands accessing protected storage must be authorized to protect stored sensitive data
  - Authorization data based on SHA-1 hashed pass-phrase
  - Prevention of dictionary attacks (from version TPM 1.2)
- Five basic challenge-response protocols
  - Three to create and manage authorization information
    - Contained in objects under the control of TPM
  - Two to establish authorized session contexts
    - Object-Independent Authorization Protocol (OIAP)
      - Establishes an authorized clear-text session between the TPM and an external entity
      - Message integrity ensured by HMAC
    - Object-Specific Authorization Protocol (OSAP)
      - Authorized session is bound to a TPM object
      - Computes ephemeral secret

# Conclusions

---

- Secure hardware (HSMs, TPMs)
  - Limited functionality – easier to verify – better security (than multipurpose hardware)
  - Dedicated circuits – faster than software implementation
- Secure hardware doesn't guarantee absolute security
  - Any secure hardware can be reengineered
  - Main reason of its usage is increased cost of attack
- Bad design and integration imply attacks
  - The security of current generation banking APIs is really bad with respect to insider attacks
  - Number of (banking) standards implemented ensures interoperability but also causes errors
- Issues of smartcards will be discussed in 2 weeks...