



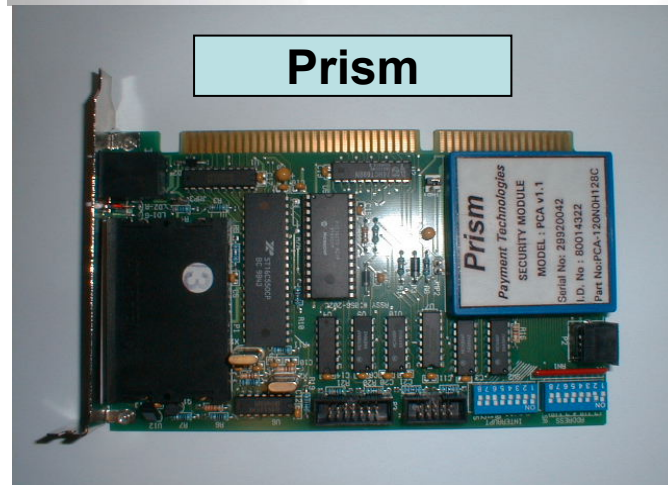
Security API analysis with the spi-calculus

Ta Vinh Thong

Contents

- HSM and security API
- A simplified API
- PIN attacks
- Challenges
- The Spi calculus
- My contribution
- Automatic analysis
- Conclusion
- Future work

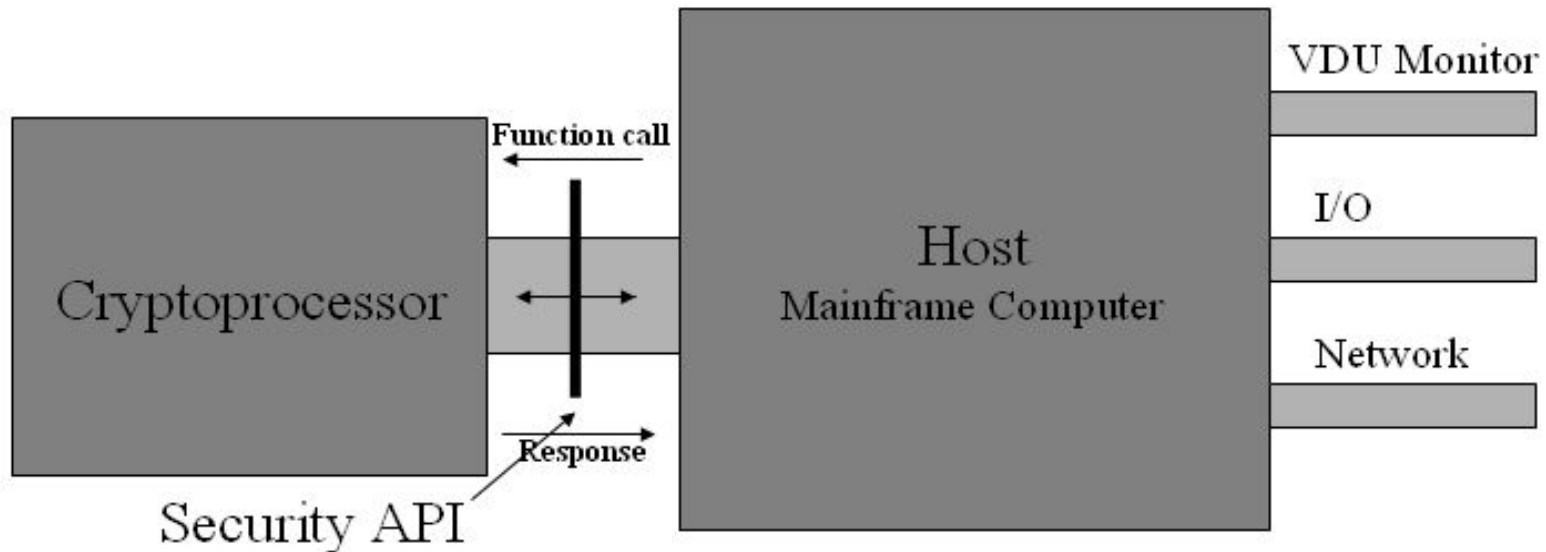
Hardware Security Modules



- Provide high-level physical protection
- Store sensitive data
- Perform cryptographic op.
- Very expensive
- Are applied in:
 - Banking(ATM) networks
 - Electronic commerce

What is a Security API ?

- to invoke the functions provided by the module.
- Use cryptography
- Enforce a security policy



A simplified API

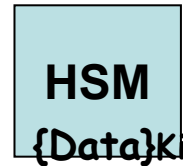
- Four functions:
 1. Data-encryption
 2. Data-decryption
 3. Key-export
 4. Key-import
- One MK master key (stored inside)
- Data encryption key - K_i (stored outside)
 $\{K_i\}_{MK}$
- Key encryption key - KEK_j (stored outside)
 $\{KEK_j\}_{MK}$

A simplified API (2)

- **data-encryption:**

- inputs: Data, $\{K_i\}_{MK}$

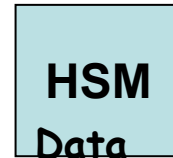
- output: $\{Data\}_{K_i}$



- **data-decryption:**

- inputs: $\{Data\}_{K_i}, \{K_i\}_{MK}$

- output: Data



- **key-export:**

- inputs: $\{K_i\}_{MK}, \{KEK_j\}_{MK}$

- output: $\{K_i\}_{KEK_j}$

- **key-import:**

- inputs: $\{K_i\}_{KEK_j}, \{KEK_j\}_{MK}$

- output: $\{K_i\}_{MK}$

Attacking the simplified API

1. key-export:

- inputs: $\{K_i\}_{MK}, \{KEK_j\}_{MK}$
- output: $\{K_i\}_{KEK_j}$

2. data-decryption

- inputs: ~~$\{D_i\}_{KEK_j}, \{KEK_j\}_{MK}$~~
- output: K_i

$K_i, KEK_j \leftarrow$ secured under the same MK master key!!!
(they have a same type)

PIN attacks

- **Key separation attack** against Visa Security Module. **Extract customer's PIN.**
- **Check value attacks** (Visa Module, IBM 4758)
- **Decimalization attack** (Financial APIs)
- **Type-casting attack** (IBM 4758)
- **Key-conjuring attack** (Financial APIs)
- **Related key attack** (Financial APIs)

Challenges

- In real life, security APIs may contain few hundred calls.
- APIs attacks are very subtle. Most of the commercially available APIs had flaws.
- Informal analysis is error-prone. More systematic approach is needed.

Idea: Using formal analysis!!!

My idea

- Use **the spi-calculus** (an extension of the pi-calculus) for this purpose

Why???

- Spi-calculus provides semantics to model **cryptography**
- Model and prove **secrecy properties**
- Spi-calculus provides **expressive semantics**, with large set of cryptography operations.
- Spi-calculus is a programming language thus it is **easy to apply** it for modeling security APIs
- Modeling security APIs with the spi-calculus **is not complicated** due to the functions of APIs are simple.

Syntax of Spi-Calculus

- Terms:

$L, M, N ::=$

n *name*

(M, N) *pair*

0 *zero*

$suc(M)$ *successor*

x *variable*

$\{M\}_N$ *shared – key encryption*

- Processes:

$P, Q, R ::=$	
$\bar{M} \langle N \rangle . P$	<i>output</i>
$M(x) . P$	<i>input</i>
$P Q$	<i>composition</i>
$(\nu n) P$	<i>restriction</i>
$!P$	<i>replication</i>
0	<i>nil</i>
$[M \text{ is } N] P$	<i>match</i>
$\text{let } (x, y) = M \text{ in } P$	<i>pair splitting</i>
$\text{case } M \text{ of } 0 : P \text{ suc}(x) : Q$	<i>integer case</i>
$\text{case } L \text{ of } \{x\}_N \text{ in } P$	<i>shared – key decryption</i>

Hashing, public key and digital signatures

- We need to add more primitives to the grammar:

- Hashing:

$$L, M, N ::= H(M) \quad \textit{hashing}$$

- Public key:

$$L, M, N ::= M^+ \quad \textit{public part}$$
$$M^- \quad \textit{private part}$$
$$\{[M]\}_N \quad \textit{public - key encryption}$$
$$P, Q, R ::= \textit{case } L \textit{ of } \{[x]\}_N \textit{ in } P \quad \textit{decryption}$$

- Digital signatures

$$L, M, N ::= [\{M\}]_N \quad \textit{private - key signature}$$
$$P, Q, R ::= \textit{case } L \textit{ of } [\{x\}]_M \textit{ in } P \quad \textit{signature check}$$

Semantic of Spi-calculus

- Reaction: (\rightarrow)

$$\bar{m}\langle N \rangle.P \mid m(x).Q \longrightarrow P \mid Q[N/x]$$

- Various process equivalences:

- Structural equivalence
- Strong Bisimilarity
- Testing Equivalence
- Barbed Equivalence
- Barbed Simulation
- Barbed Congruence

Modeling the API

$MODULE^{ENC}, MODULE^{DEC}, MODULE^{EXP}, MODULE^{IMP}$

$C_{enc}, C_{dec}, C_{exp}, C_{imp}, \bar{C}_{user}$

data-encryption(with type):

- inputs: Data, {DataKey, Ki}MK
- output: {TData, Data}ki

$MODULE^{ENC}(MK) \triangleq$

$C_{enc}(x_{data}, x_{token0}). \text{case } x_{token0} \text{ of } \{x_{typeK}, x_{K_i}\}_{MK} \text{ in } [x_{typeK} \text{ is } DataKey]$

$\bar{C}_{user} < \{TData, x_{Data}\}_{x_{K_i}} >.$

$SysAPI(K_i, KEK_j) \triangleq$

$(\nu MK) (\bar{C}_{user} < \{K_i\}_{MK} > . \bar{C}_{user} < \{KEK_j\}_{MK} > .$

$(!MODULE^{DEC}(MK) || !MODULE^{EXP}(MK) || !MODULE^{ENC}(MK) || !MODULE^{IMP}(MK)))$

Definition of security

The module provides the secrecy of K_i if :

Arbitrary K_i and K_j

$$\text{Sys}_{API}(K_i, KEK) | R \approx \text{Sys}_{API}(K_j, KEK) | R$$

Arbitrary process!!!

Define an binary, symmetric relation \mathfrak{R} between $\text{Sys}_{API}(K_i, KEK) | R$ and $\text{Sys}_{API}(K_j, KEK) | R$

1. if $\text{Sys}_{API}(K_i, KEK) | R \downarrow \beta$ then $\text{Sys}_{API}(K_j, KEK) | R \downarrow \beta$
2. if $\text{Sys}_{API}(K_i, KEK) | R \rightarrow P'$ then there exist Q' , such that $\text{Sys}_{API}(K_i, KEK) | R \rightarrow Q'$, and $P' \mathfrak{R} Q'$.

My contributions

- I adapted the proof technique of the spi-calculus to prove the security of the API.

$$Sys_{API}(K_i, KEK) \mid R \approx Sys_{API}(K_j, KEK) \mid R$$

(Complicated proof. I made some changes, created lemmas.)

- I fixed the example API shown before by introducing **types**.
- With the spi-calculus, I formally
 - Model the above attack.
 - Proved manually the security of the fixed simplified API.
- I also explore some ways of automating the proof: **The ProVerif tool**.

Automatic Analysis with the ProVerif tool

- Input = (applied) pi-calculus form \rightarrow expressive semantics (translate to Horn-clause, and use resolution solving alg.)
- Yields proof for an **unbound number** of sessions and **unbounded number** of message space.
- Allows generic definitions of crypt. primitives
 - Hash, symmetric and asymmetric crypto, digital signature, Exor operation.
- Finds shortest attacks

<pre>Goal of the attack : attacker:k[] out(attacker, encrypt(k,mk_20)) Completing... ok, secrecy assumption verified: fact unreachable attacker:mk[] Starting query not attacker:k[] RESULT not attacker:k[] is true. Starting query not attacker:kek[] RESULT not attacker:kek[] is true. c:\proverif1.14pl2></pre>	<pre>Goal of the attack : attacker:kek[] out(attacker, encrypt(k,mk_41))</pre>
<pre>out(attacker, encrypt(kek,mk_20)) out(attacker, encrypt(k,mk_20)) Completing... ok, secrecy assumption verified: fact unreachable attacker:mk[] RESULT Equivalence proof succeeded (bad not derivable). i c:\proverif1.14pl2>analyzer -in pi examples/pinoninterf/apistrongsec _</pre>	<pre>out(attacker, encrypt(k,mk_41)) out(attacker, encrypt(kek,mk_41))</pre>
<pre>out(attacker, encrypt(k,kek)) in(moduledec, (encrypt(k,kek),encrypt(kek,mk_20))) out(attacker, k) An attack has been found. RESULT not attacker:k[] is false.</pre>	<pre>out(attacker, kek) An attack has been found. RESULT not attacker:kek[] is false. c:\proverif1.14pl2>analyzer -in pi examples/piboth/apiattackTacas</pre>

Conclusion

- Attacks against security APIs can be very subtle. Therefor, informal analysis is not reliable.
- Formal technique can be useful here.
- I provide a survey of API attacks (the summary of many papers).
- I propose the application of the spi-calculus for security API analysis.
- I demonstrated how to model security APIs with spi-calculus, developed a technique to prove the security of APIs, and proved the security of a simplified API manually.
 - To the best of our knowledge I am the first who use them to model security APIs.
- I demonstrated how to use the ProVerif tool to find attacks, and prove the security of the simplified API.

Future Work

- Use the ProVerif tool to analyse more complex, commercially available security APIs
- Investigate other automatic tools.
- Improve the spi-calculus to be able to model more operations used in APIs.