Some Security Problems Raised by Open Multiapplication Smart Cards

Serge Chaumette

serge.chaumette@labri.fr

&

Damien Sauveron

damien.sauveron@labri.fr

http://damien.sauveron.free.fr/

Tartu, Estonia
21/10/2005

Outline

- Context of our work
- Closed smart cards
- Common attacks
- Towards open multiapplication smart cards
- New threats
- Physical characterization methods
- Physical attacks
- Matching attack
- Experiments
- Conclusion and perspectives

Context of our work

Java Card Security project (2001-2003):

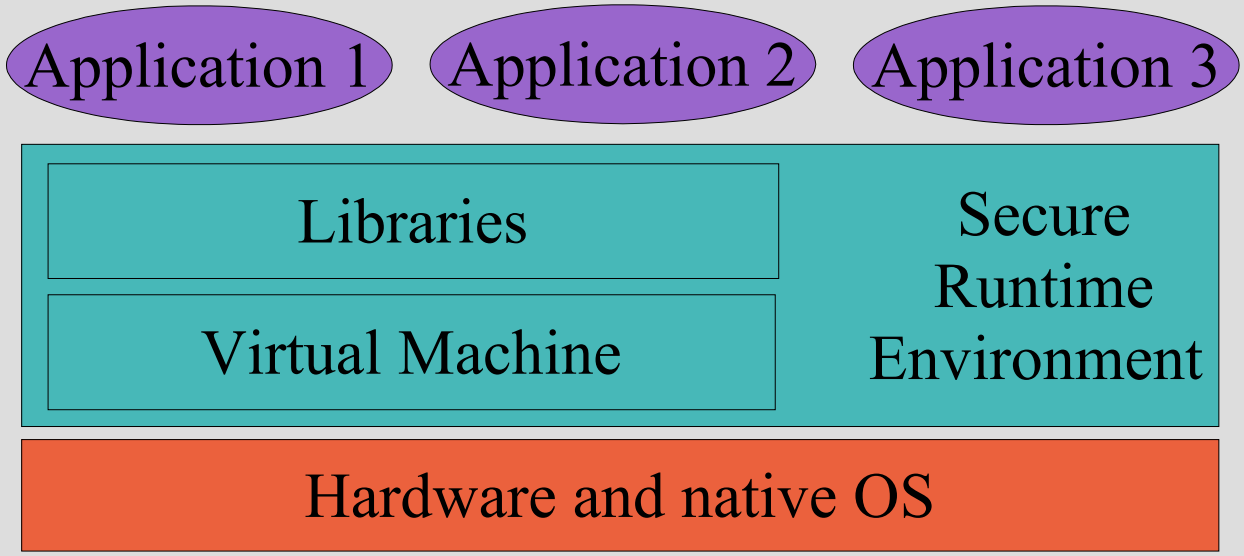
- funded by the French Ministry of Industry
- LaBRI (Bordeaux, France )
  - security of Java Card technology
- Information Technology Security Evaluation Facilities of SERMA Technologies (Pessac, France)
  - specialized in the Common Criteria security evaluation of smart card chips and Java Card products.

Question: How future open multiapplication smart cards will be threated at physical level?

Closed smart cards

- Monoapplication
(proprietary OS)

- Multiapplication
(Java Card, Smartcard.NET, Multos)



Closed = only authorized people can upload applications

- Need authentification keys

Common attacks (1/2)

Software attacks:

- APDU fuzzing, ...

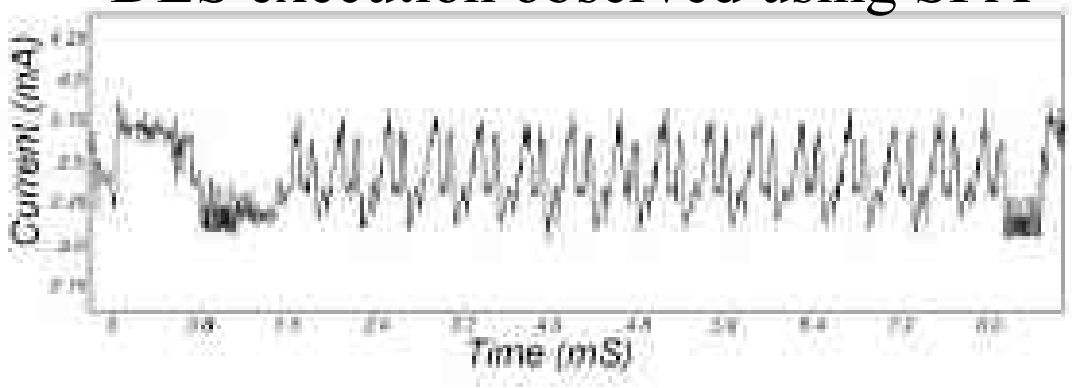
```
00000000...
01000000...
...
FF000000...
00010000...
01010000...
...
FF010000...
...
```

Generally it does not work!

Hardware attacks:

- Side channel attacks
  - Timing attack
  - SPA (Simple Power Analysis) / SEMA (Simple ElectroMagnetic Analysis)
  - DPA (Differential Power Analysis) / DEMA (Differential ElectroMagnetic Analysis)

DES execution observed using SPA

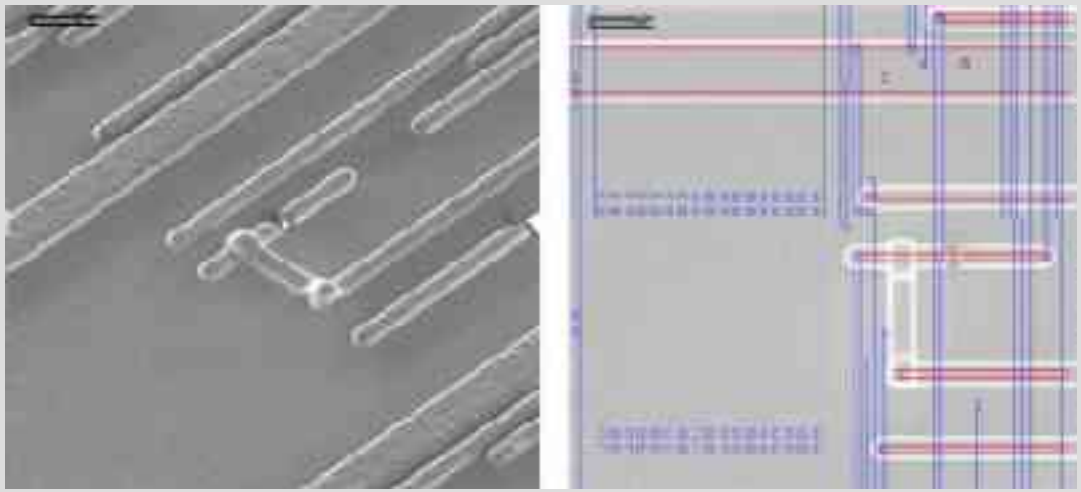


Common attacks (2/2)

Hardware attacks:

- Fault injection attacks
  - Laser injection
  - Glitches, ...

- Microprobing
- Modifications of the chip with a FIB (Focused Ion Beam)



Towards open multiapplication SC

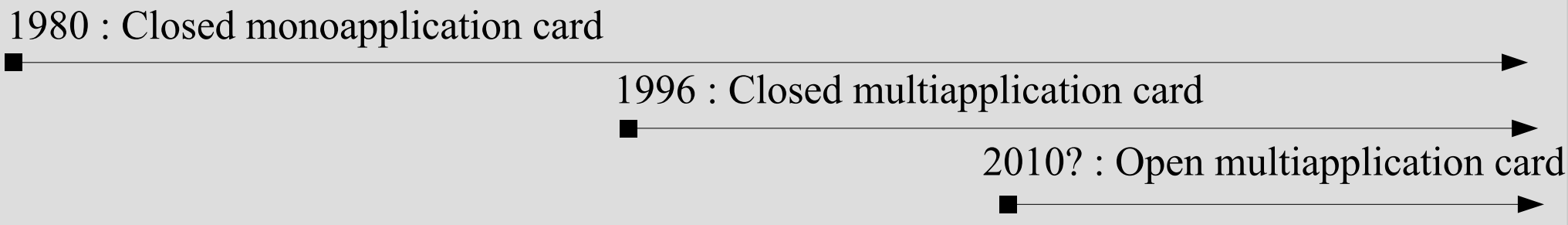
Open multiapplication card = everybody can upload his own application (even malicious applications!)

Same architecture of the closed multiapplication SC

Additionnal security mechanisms:

- Bytecode verifier and an offensive VM
- A defensive VM

Should be available in few years on the market.



New threats

Internal attacks:

- Identification of services and collection of information
- Attacks against the VM and the firewall
- Mixed hardware and software attacks

Attacks used by ITSEF and IC manufacturers to test their **closed** products since they own keys

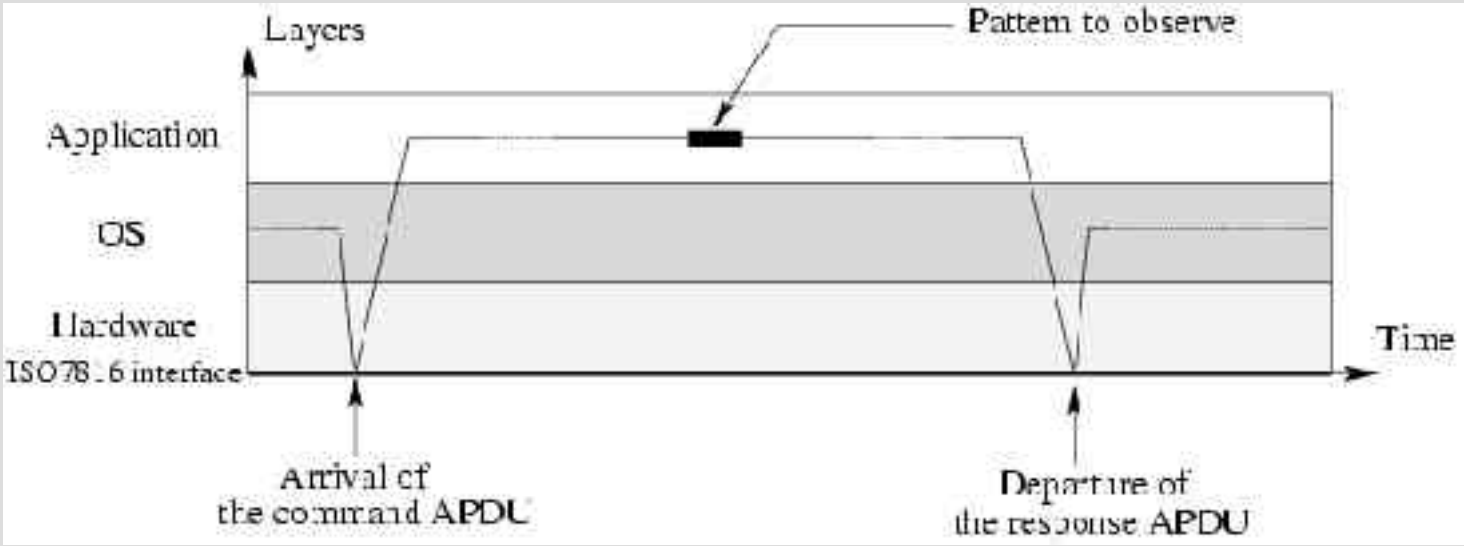
Our contribution: we exibit the possibilities to characterize the open smart cards from inside:

- to improve the efficiency of mixed attacks
- to propose new kinds of attacks

Physical characterization methods (1/5)

Goal: To identify a pattern in an execution trace to perform attacks afterwards.

- Normal execution which contains the pattern to observe



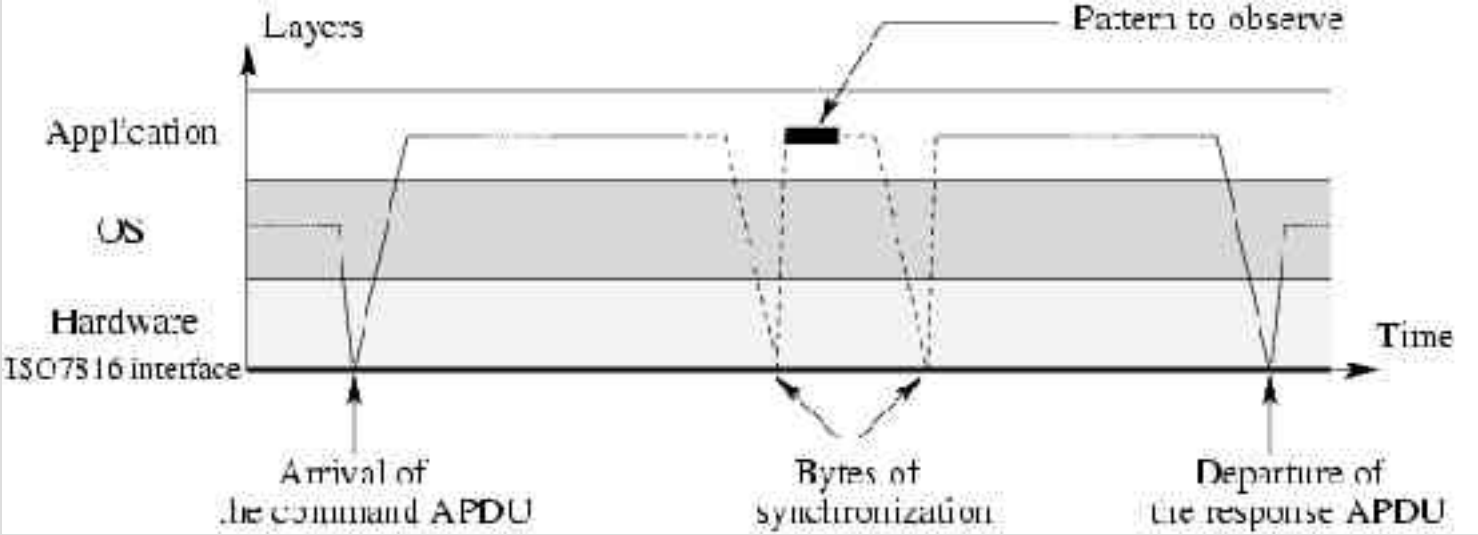
- Example: DES computation, bytecode or method execution, ...

Identification is really difficult!

Physical characterization methods (2/5)

Goal: To provide mechanisms to help in the process of identification

- Additionnal glitches on the I/O in the program using a workaround in the communication protocol (Miami, FL, USA -- WOSIS 2005)



- We have proposed solutions to block these attacks using randomization and bufferization methods before the smart card sends the APDU response

Physical characterization methods (3/5)

Classical application code

```
public void process(APDU apdu) {
...

   // Pattern to observe
   cipherLength = cipher.doFinal(plaintextData, (short) 0,
               (short) plaintextData.length,
               cipherData, (short) 0);
...
}
```

Application code using glitches

```
public void process(APDU apdu) {
...

   // Send the synchro ... means the beginning of the process (glitch 1)
   apdu.sendBytes((short) 0, (short) 1);

   // Pattern to observe
   cipherLength = cipher.doFinal(plaintextData, (short) 0,
               (short) plaintextData.length,
               cipherData, (short) 0);

   // Send the synchro ... means the end of the process (glitch 2)
   apdu.sendBytes((short) 0, (short) 1);
...
}
```

Classical application bytecodes

```
getfield_a_this 0x0
getfield_a_this 0x1
sconst_0
getfield_a_this 0x1
arraylength
getfield_a_this 0x2
sconst_0
invokevirtual   0x0 0xa // pattern to observe (real encryption)
sstore_2
```

Application bytecodes using glitches

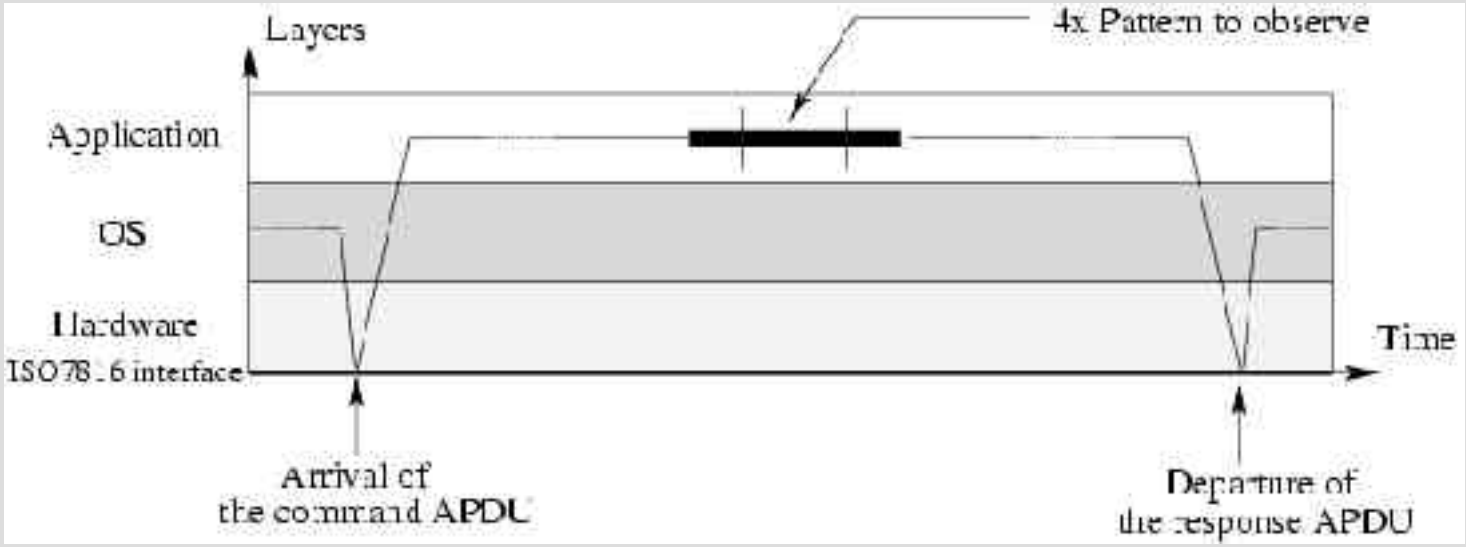
```
aload_1
sconst_0
sconst_1
invokevirtual   0x0 0x6 // glitch 1 (real glitch)
getfield_a_this 0x0
getfield_a_this 0x1
sconst_0
getfield_a_this 0x1
arraylength
getfield_a_this 0x2
sconst_0
invokevirtual   0x0 0xa // pattern to observe (real encryption)
sstore_2
aload_1
sconst_0
sconst_1
invokevirtual   0x0 0x6 // glitch 2 (real glitch)
```

Application bytecodes using optimized glitches

```
aload_1
sconst_0
sconst_1
getfield_a_this 0x0
getfield_a_this 0x1
sconst_0
getfield_a_this 0x1
arraylength
getfield_a_this 0x2
sconst_0
aload_1
sconst_0
sconst_1
invokevirtual   0x0 0x6 // glitch 1 (real glitch)
invokevirtual   0x0 0xa // pattern to observe (real encryption)
sstore_2
invokevirtual   0x0 0x6 // glitch 2 (real glitch)
```

Physical characterization methods (4/5)

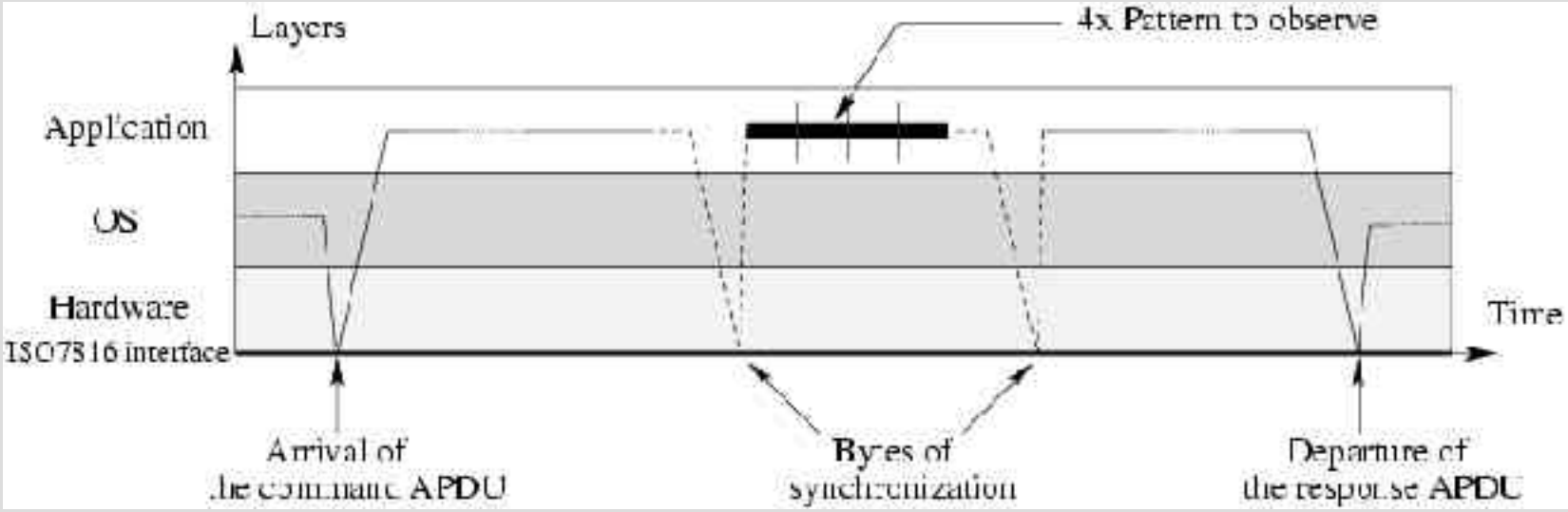
- Repeating the pattern to observe



- A solution may be to use a random clock or asynchronous processors

Physical characterization methods (5/5)

- Hybrid method combines the two advantages of previous methods for a better efficiency



Now we can identify the pattern that we want.

Physical attacks

Goal: To quickly evaluate the feasibility of an attack against an official application

- First identify a pattern used by an official application
- Develop an application that uses it and apply one of the previous characterization methods
- Upload the application on card
- Attack the pattern (e.g. with fault injection)
- If it is a success, try to attack the official application else try to find an other interesting pattern to attack

Best experimental position:

- Implementation is known
- Location of the pattern to attack is also known

Advantage: save many time

Matching attack

Goal: To reverse-engineer an official application

- Build a dictionnary that contains a lot of patterns for the targeted smart card using the characterization methods
- Save an execution trace of the official application
- Match the patterns identified in the trace with the dictionnary
- Now you should get the code of the official application
- Try to find a bug that you can exploit
  - Having the code does not implies that there is a security problem
  - Often in security evaluation of an IT product the code is an asset that should be protected

Experiments

Done at the ITSEF:

- On closed multiapplication smart cards using the keys we had
- JCatools is used to build optimized tests suite

Unsucceded to perform a matching attack on a certified Java Card available on the market

Succeded to achieve physical attacks on multiapplication smart cards!

Conclusion and perspectives

Open multiapplication smart cards open the way to new threats

New features of the software lead to threats at hardware level

In 2006 at Limoges the new laboratory called XLIM will be created with people specialized:

- in antennas (old IRCOM laboratory), in cryptography (old LACO laboratory), in information security (old LMSI laboratory)

The work initiated at the LaBRI will continue, and there also will be a collaboration with the XLIM laboratory in a project to securize smart cards.

Disclaimer & Copyright

Some pictures of this document come from Internet sources.