Checking the Path to Identify Control Flow Modification

Ahmadou A. SERE, Julien IGUCHI-CARTIGNY,
Jean-Louis LANET

PASTIS 2010
16th-17th June

Smart Secure Device

Xlim institut de recherche

Université de Limoges

Outline

I. Introduction

II. Fault model

III. Path check countermeasure

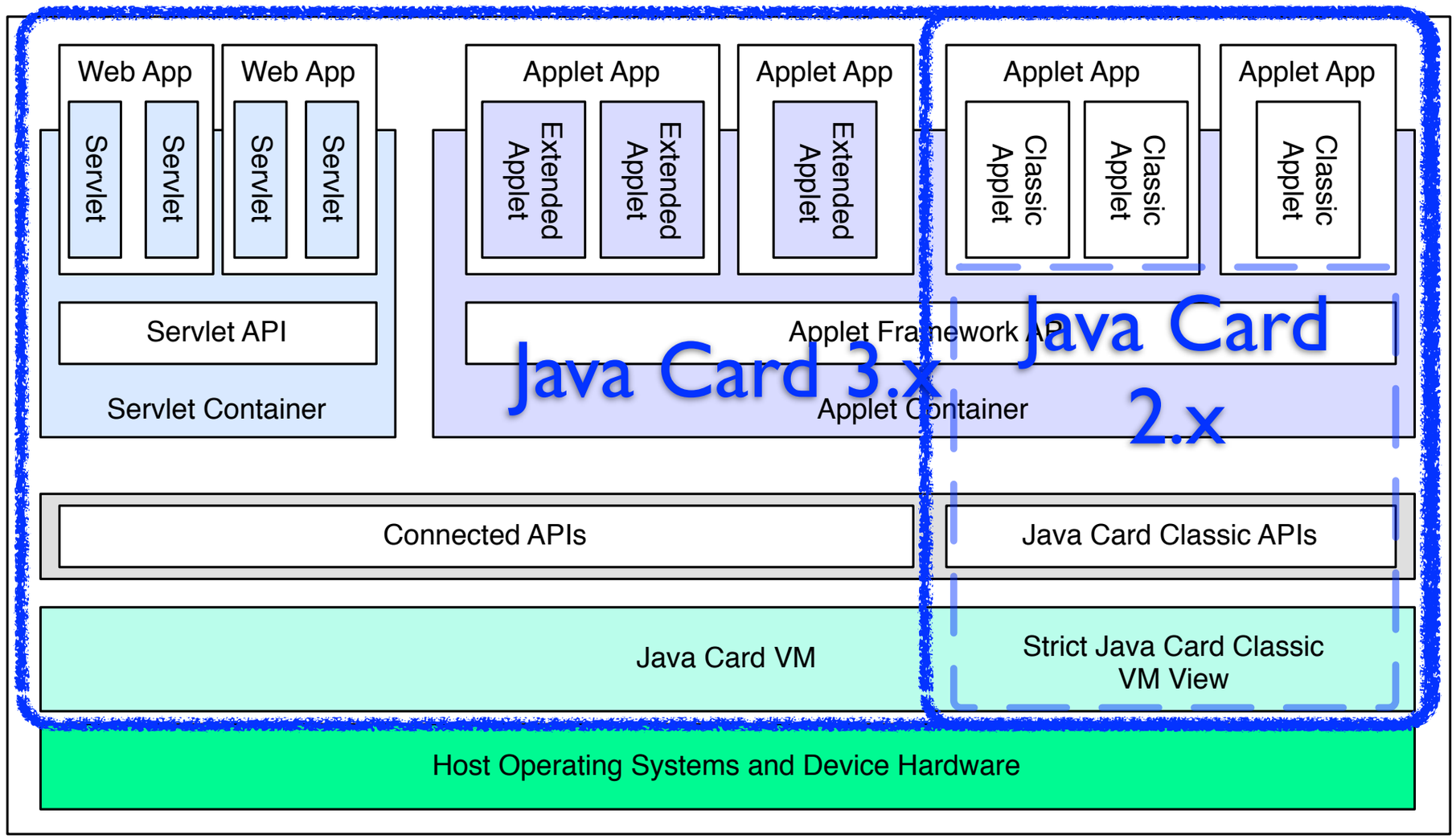
IV. Evaluation

V. Conclusion

Introduction

- Work on smart card
- Work on countermesures against attacks
  - Fault attacks
- Java Card
  - Java Implementation for smart card
- Application protection
  - Code modification
    - Control flow modification
- Already tested against
  - EMAN attacks
  - Mutating application

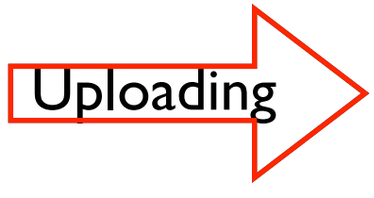
Java Card



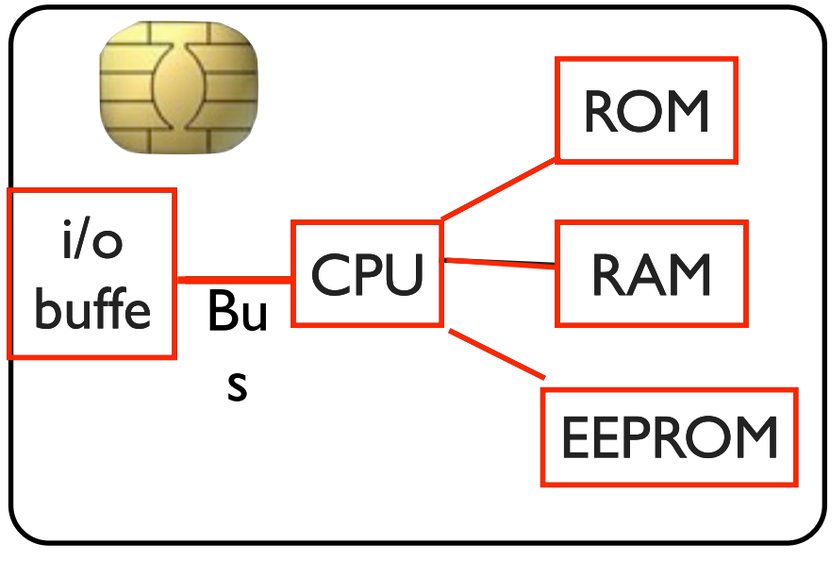
Java Card architecture

Fault Attacks

- Physical attacks
  - Laser
  - EM Field
- Two major categories
  - ~~Invasive~~
  - Non invasive

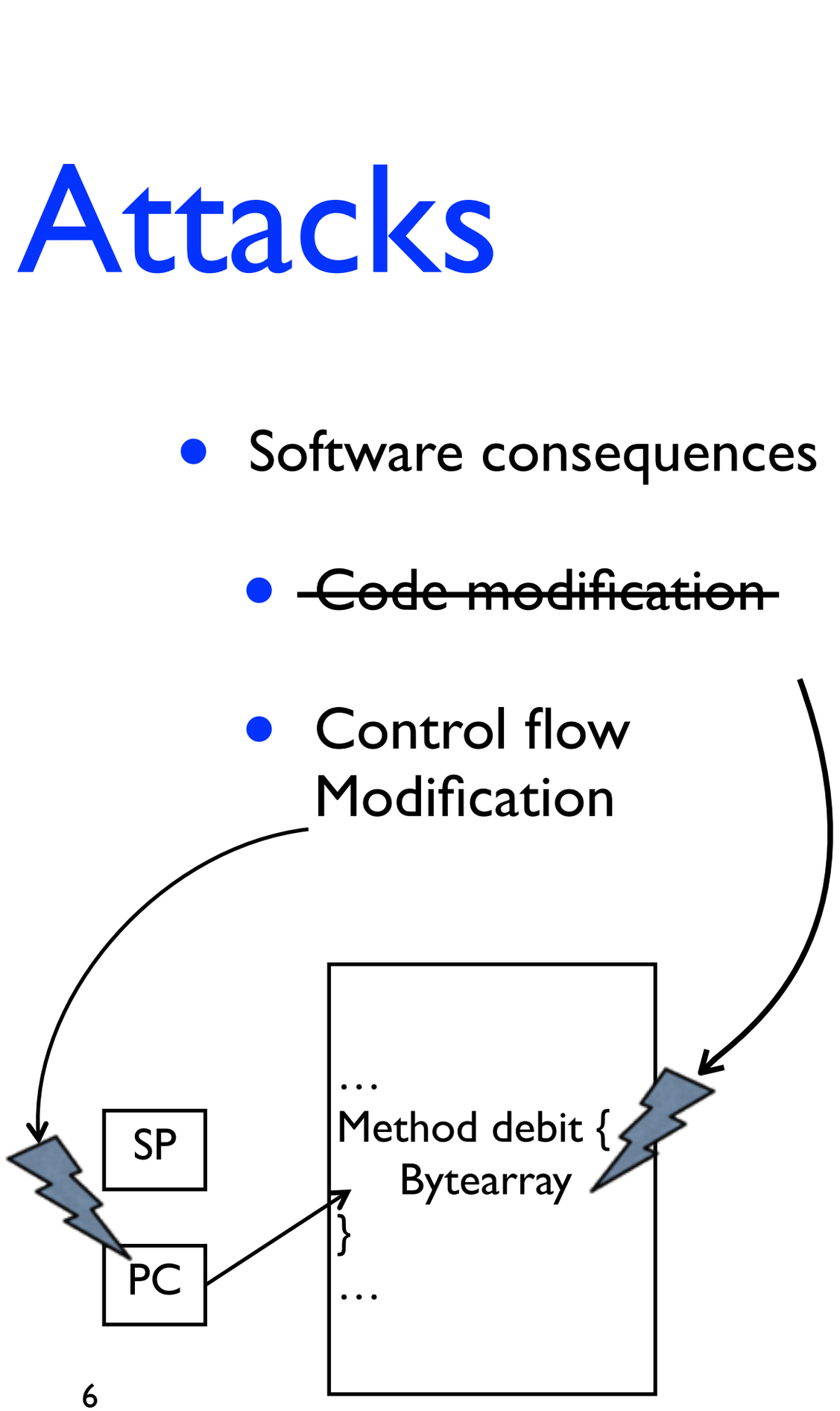


- Software consequences
  - Code modification
  - Control flow Modification



Fault Attacks

- Physical attacks
  - Laser
  - EM Field
- Two major categories
  - ~~Invasive~~
  - Non invasive
- Software consequences
  - ~~Code modification~~
  - Control flow Modification



Fault Model

- Fault Model
  - Location
  - Timing
  - Precision
  - Fault type

Fault Model

Fault model	Timing	precision	location	fault type	Difficulty
Precise bit error	total control	bit	total control	set (1) or reset (0)	++
Precise byte error	total control	byte	total control	set (0XFF), reset (0x00) or random	+
Unknown byte error	loose control	byte	no control	set (0XFF), reset (0x00) or random	-
Unknown error	no control	variable	no control	set (0XFF), reset (0x00) or random	--

Example of fault attack

Bytecode

```
00 : aload_0
01 : getfield #4
04 : invokevirtual #18
07 : ifeq 59
10 :  …
…
59 : sipush 25345
63 : invokestatic #13
66 : return
```

Octets

```
00 : 18
01 : 83 00 04
04 : 8B 00 23
07 : 60 00 3B
10 : …
…
59 : 13 63 01
63 : 8D 00 0D
66 : 7A
```

Java code

```
private void debit(APDU apdu) {


  if ( pin.isValidated() ) {
     // make the debit operation
  } else {
     ISOException.throwIt (
       SW_PIN_VERIFICATION_REQUIRED);
}
```

Bytecode

```
00 : aload_0
01 : getfield #4
04 : invokevirtual #18
07 : nop
08 : nop
09 : pop
10 :  …
…
59 : sipush 25345
63 : invokestatic #13
66 : return
```

Octets

```
00 : 18
01 : 83 00 04
04 : 8B 00 23
07 : 60
08 : 00
09 : 3B
10 : …
…
59 : 13 63 01
63 : 8D 00 0D
66 : 7A
```

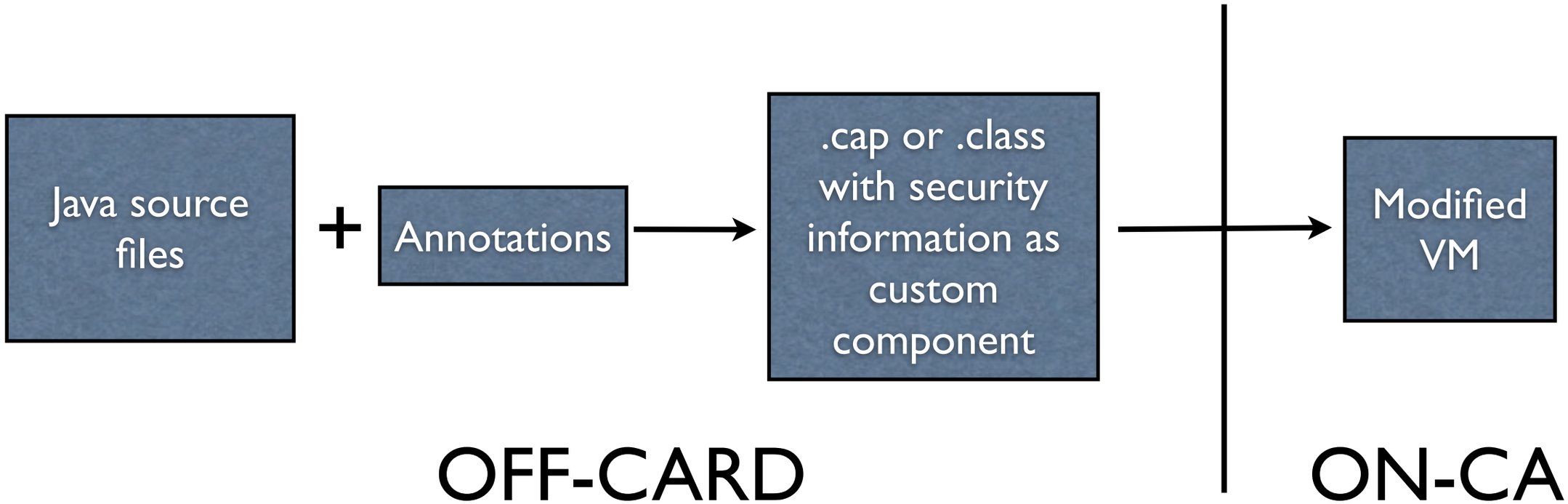
Java code

```
private void debit(APDU apdu) {

  if ( pin.isValidated() ) {
     // make the debit operation

} else {
     ISOException.throwIt (
       SW_PIN_VERIFICATION_REQUIRED
}
```

A Secure VM

- VM
  - Compliant with the *Sun* specification
  - Run application safely
    - Detect code modification
    - Detect control flow modification
- Ressources constraints
  - Memory consumption
  - CPU overhead

Approach



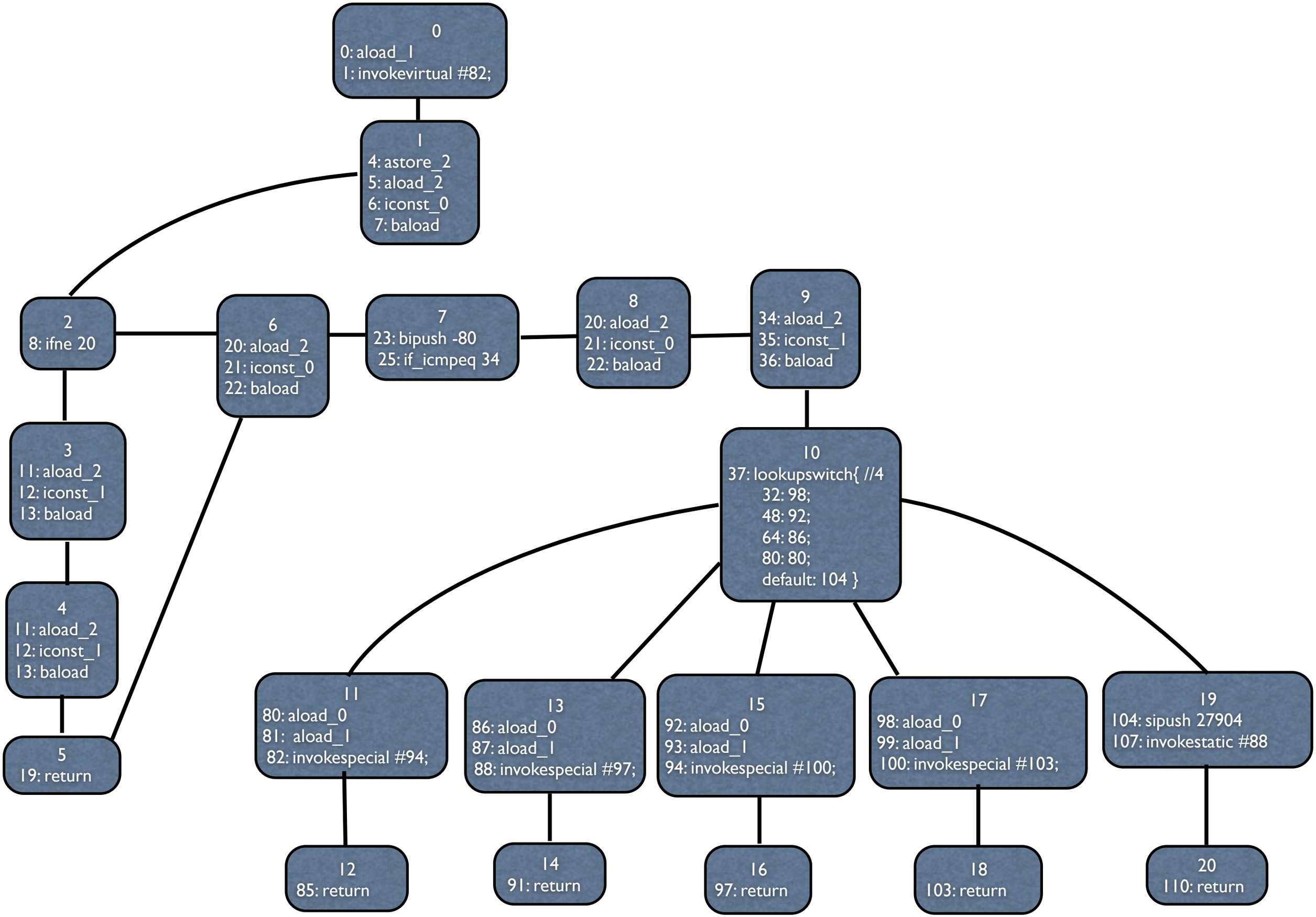

```
@SensitiveType{
 sensitivity= SensitiveValue.INTEGRITY,
 proprietaryValue="PathCheck"
}
private void debit(APDU apdu) {
if ( pin.isValidated() ) {
   //make the debit operation
} else {
   ISOException.throwIt
   (SW_PIN_VERIFICATION_REQUIRED);
}                          ||
```

Path check mechanism

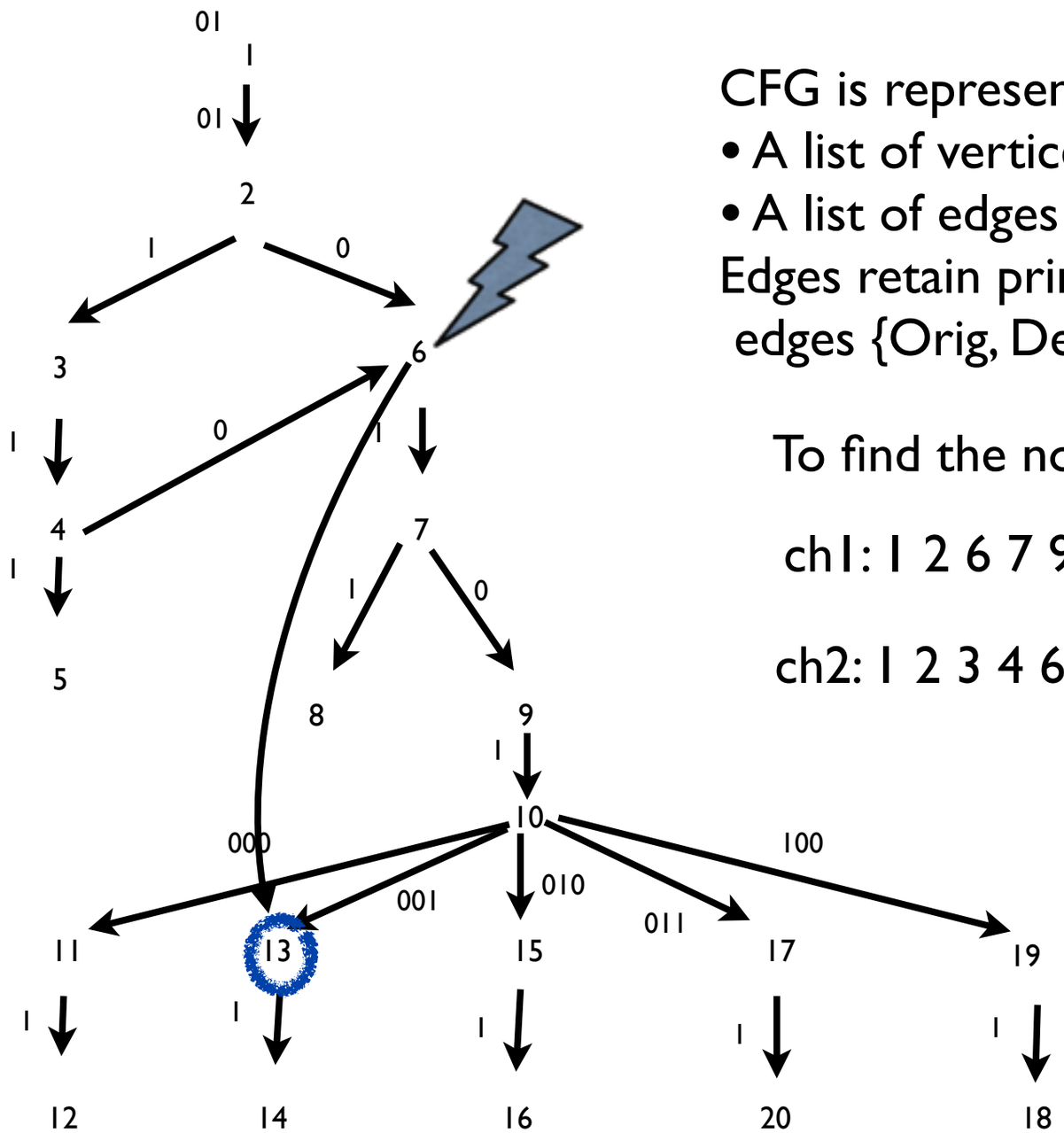
- Goal:
  - Detect PC modification during control flow transfer
- Principle
  - Off-card
    - Create a Control Flow Graph (CFG)
    - Find all potential paths in this graph and memorise them
  - On-card
    - Compute path during runtime
    - Check path concordance between the previously saved one

Path check mechanism

```
public void process(javacard.framework.APDU);
  Code:
   Stack=2, Locals=3, Args_size=2
   0: aload_1
   1: invokevirtual        #82; //Method javacard/framework/APDU.getBuffer:()[B
   4: astore_2
   5: aload_2
   6: iconst_0
   7: baload
   8: ifne 20
   11: aload_2
   12: iconst_1
   13: baload
   14: bipush -92
   16: if_icmpne 20
   19: return
   20: aload_2
   21: iconst_0
   22: baload
   23: bipush -80
   25: if_icmpeq 34
   28: sipush 28160
   31: invokestatic #88; //Method javacard/framework/ISOException.throwIt:(S)V
   34: aload_2
   35: iconst_1
   36: baload
   37: lookupswitch{ //4
            32: 98;
            48: 92;
            64: 86;
            80: 80;
            default: 104 }
   80: aload_0
   81: aload_1
   82: invokespecial #94; //Method getBalance:(Ljavacard/framework/APDU;)V
   85: return
   86: aload_0
   87: aload_1
   88: invokespecial #97; //Method debit:(Ljavacard/framework/APDU;)V
   91: return
   92: aload_0
   93: aload_1
   94: invokespecial #100; //Method credit:(Ljavacard/framework/APDU;)V
   97: return
   98: aload_0
   99: aload_1
   100: invokespecial      #103; //Method verify:(Ljavacard/framework/APDU;)V
   103: return
   104: sipush      27904
   107: invokestatic #88; //Method javacard/framework/ISOException.throwIt:(S)V
   110: return
```


0
0: aload_1
1: invokevirtual #82;
1
4: astore_2
5: aload_2
6: iconst_0
7: baload
2
8: ifne 20
6
20: aload_2
21: iconst_0
22: baload
7
23: bipush -80
25: if_icmpeq 34
8
20: aload_2
21: iconst_0
22: baload
9
34: aload_2
35: iconst_1
36: baload
3
11: aload_2
12: iconst_1
13: baload
4
11: aload_2
12: iconst_1
13: baload
5
19: return
10
37: lookupswitch{ //4
32: 98;
48: 92;
64: 86;
80: 80;
default: 104 }
11
80: aload_0
81: aload_1
82: invokespecial #94;
13
86: aload_0
87: aload_1
88: invokespecial #97;
15
92: aload_0
93: aload_1
94: invokespecial #100;
17
98: aload_0
99: aload_1
100: invokespecial #103;
19
104: sipush 27904
107: invokestatic #88
12
85: return
14
91: return
16
97: return
18
103: return
20
110: return

Path check mechanism



CFG is represented by
- A list of vertices
- A list of edges

Edges retain principally three information:
edges {Orig, Dest, Code}

To find the node that has the number 13:

ch1: 1 2 6 7 9 10 13 ⇔ 01 0 1 0 1 001

ch2: 1 2 3 4 6 7 9 10 17 ⇔ 01 1 1 0 1 0 1 001

We make a comparison of runtime computed path
chr1: 01 0 or
chr2: 01 1 1 0 with
ch1: 01 0 | 0 1 001 or
ch2: 01 1 1 0 | 0 1 001
⇒ fail

Evaluation

- At91 board
  - Arm7 CPU
  - Same ressources as regular smart card
- Simple RTJ
  - Tiny Java Virtual machine
  - Highly restricted constraint devices
- Abstract Byte Code Interpreter
  - Simulation of fault attack

Results

Overhead	CPU	EEPROM	ROM
Path check	< 8%	Variable (0 to 7%)	≈ 2%

Conclusion

- Countermeasure proposed
  - Affordable for the card
  - Respectful of Java Card specification
  - Don’t need for a developer to be aware of the underlying security mechanisms
  - Need small changes on VM interpreter
  - Portability of application

Ressources

http://secinfo.msi.unilim.fr

http://msi.unilim.fr/~sere

ANY QUESTIONS ?

Thank you for your attention