Checking the Paths to Identify Mutant Application on Embedded Systems

Ahmadou A. SERE

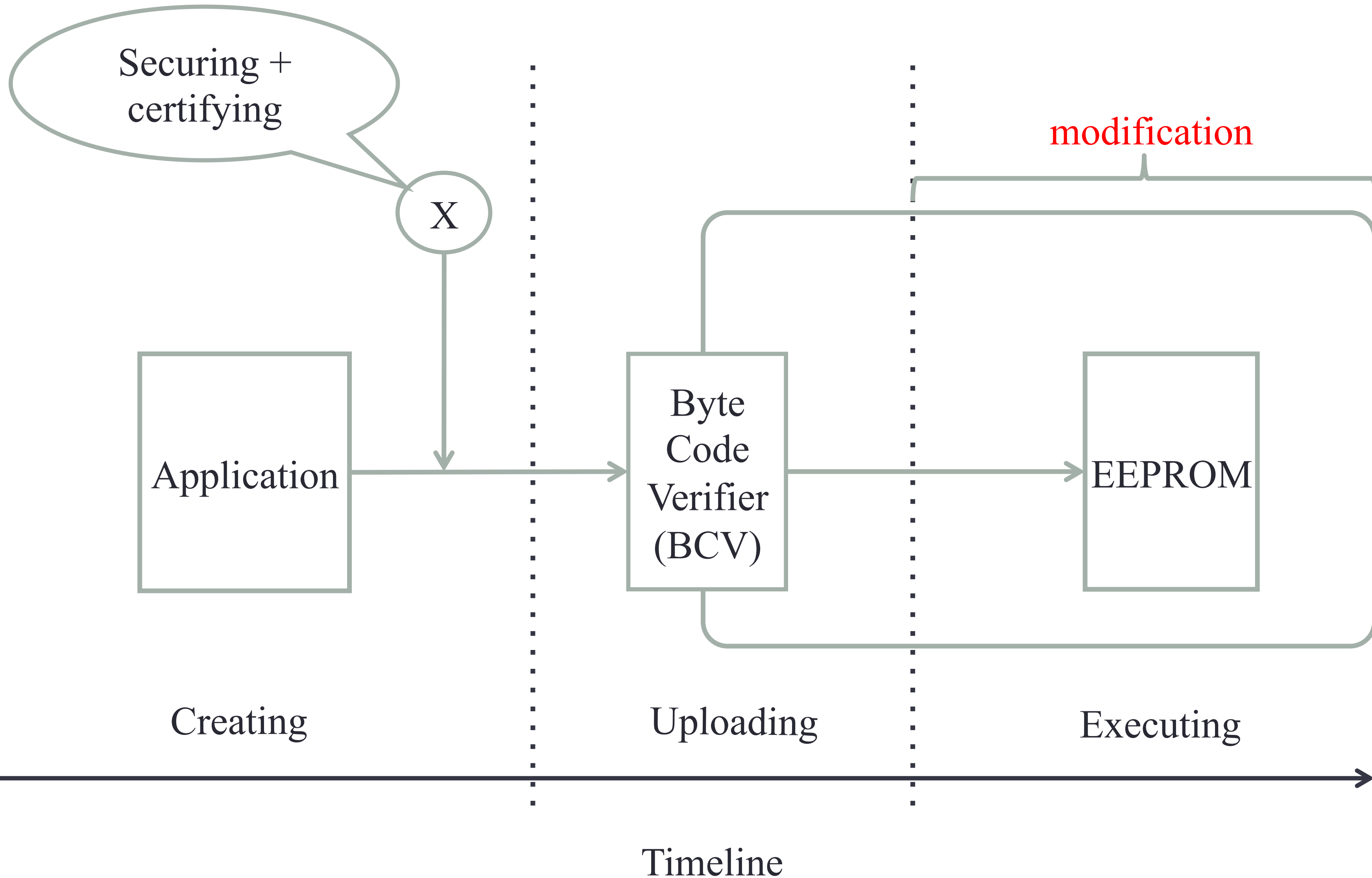
Speaker:
Jean-Louis LANET

SecTech 2010

Outline

I. Work context
II. Mutant application
III. Path check detection
IV. Evaluation process
V. Conclusion

Java Card worflow



Xlim
institut de recherche

Mutant

- Definition
  - A piece of code that passed the BC verification during the loading phase or any certification or any static analysis, and has been loaded into the EEPROM area,
  - This code is modified by a fault attack,
  - It becomes hostile : illegal cast to parse the memory, access to other pieces of code, unwanted call to the Java Card API (getKey).

Attack models

Non-encrypted memory

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 (0x00), reset (0xFF) or random	+
Unkonwn byte error	loose control	byte	no control	set (0x00)or reset (0xFF) or random	-
Unkonwn error	no control	variable	no control	set (0x00), reset (0xFF) or random	--

Encrypted memory

Example of mutant

Bytecode

```
00 : aload_0
01 : getfield #4
04 : invokevirtual #61
07 : ifeq 59
10 :  …
…
57 : goto 66
59 : sipush 25345
63 : invokestatic #13
66 : return
```

Octets

```
00 : 18
01 : 83 00 04
04 : 8B 00 3D
07 : 60 00 3B
10 : …
…
57 : 70 00 3B
59 : 13 63 01
63 : 8D 00 0D
66 : 7A
```

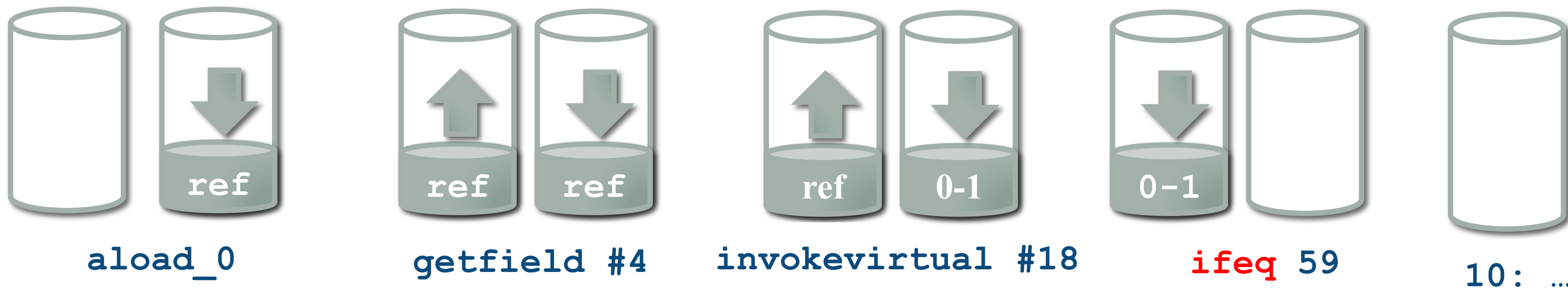
Java code

```
private void debit(APDU apdu) {


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

SW_PIN_VERIFICATION_REQUIRED);
}
```

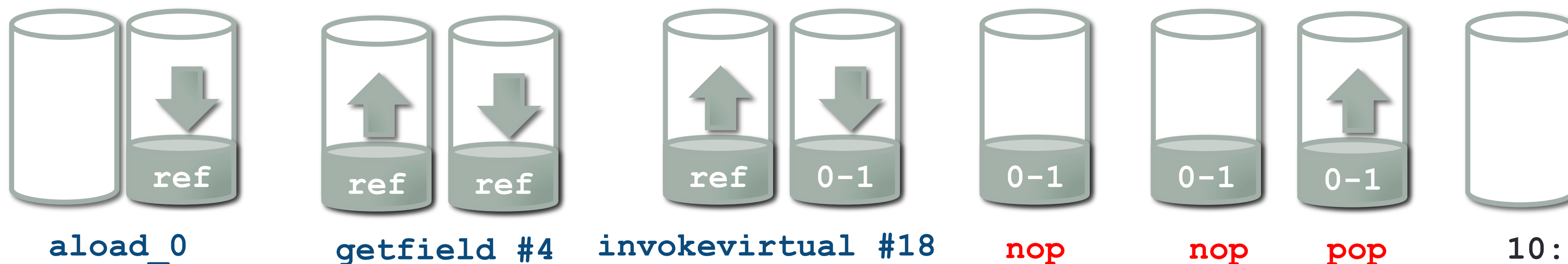
Stack



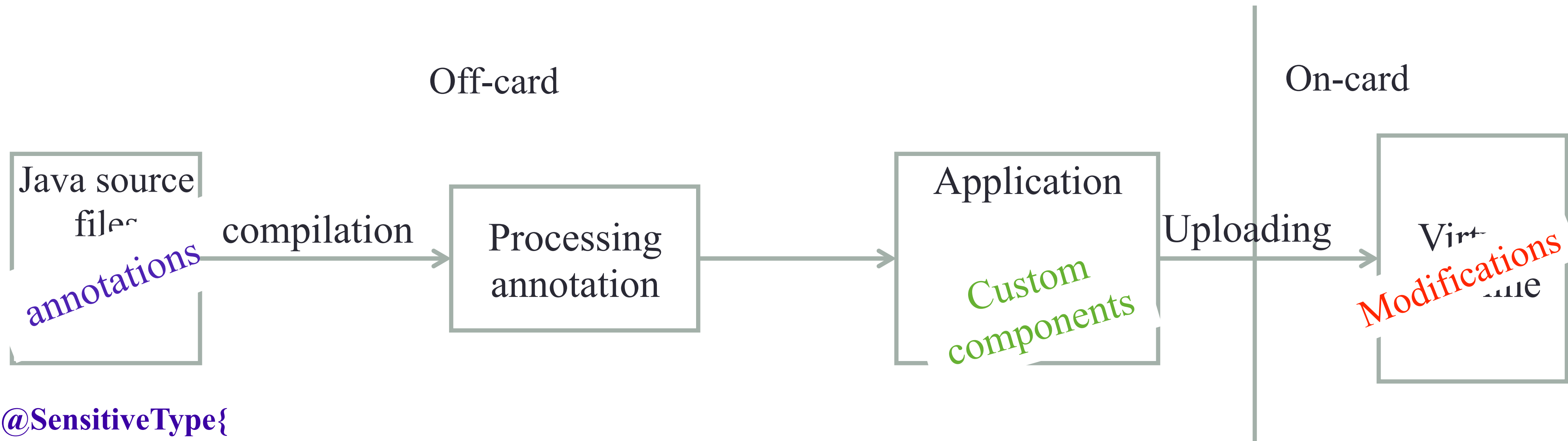
Example of mutant

Bytecode	Octets	Java code
00 : aload_0	00 : 18	private void debit(APDU apdu) {
01 : getfield #4	01 : 83 00 04	
04 : invokevirtual #61	04 : 8B 00 3D	
07 : nop	07 : 00	~~if (pin.isValidated())~~ {
08 : nop	08 : 00	
09 : pop	09 : 3B	
10 : …	10 : …	//make the debit operation
…	…	
57 : goto 66	57 : 70 00 39	~~} else {~~
59 : sipush 25345	59 : 13 63 01	ISOException.throwIt (
63 : invokestatic #13	63 : 8D 00 0D	
66 : return	66 : 7A	SW_PIN_VERIFICATION_REQUIRED);
		~~}~~
		}

Stack



Used approach



@SensitiveType{
sensitivity= SensitiveValue.INTEGRITY,
proprietaryValue="FoB"
}

```
private void debit(APDU apdu) {

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

X
X R R
X R R
X R R
…
…
X R R
X R R
X

Path Check (PCh) : principle

- Goal:
  - Detect control flow deviation
- Principle
  - Off-card :
    - Compute all the path from a control flow graph
  - On-card :
    - Compute runtime path and compare with previously computed paths

Path Check (PCh) : example

```
  0 aload_0
  1 getfield #4
  4 invokevirtual #18
  7 ifeq 98 (+91)
 10 aload_1
 11 invokevirtual #11
 14 astore_2
 15 aload_2
 16 iconst_4
 17 baload
 18 istore_3
 19 aload_1
 20 invokevirtual #19
 23 i2b
 24 istore 4
 26 iload_3
 27 iconst_1
 28 if_icmpne 37 (+9)
 31 iload 4
 33 iconst_1
 34 if_icmpeq 43 (+9)
 37 sipush 26368
 40 invokestatic #13
 43 aload_2
 44 iconst_5
 45 baload
 46 istore 5
 48 iload 5
 50 bipush 127
 52 if_icmpgt 60 (+8)
 55 iload 5
 57 ifge 66 (+9)
 60 sipush 27267
 63 invokestatic #13
 66 aload_0
 67 getfield #20
 70 iload 5
 72 isub
 73 i2s
 74 ifge 83 (+9)
 77 sipush 27269
 80 invokestatic #13
 83 aload_0
 84 aload_0
 85 getfield #20
 88 iload 5
 90 isub
 91 i2s
 92 putfield #20
 95 goto 104 (+9)
 98 sipush 25345
101 invokestatic #13
104 return
```

Xlim institut de recherche

Path Check (PCh) : example

0
```
0 aload_0;
1 getfield 4;
4 invokevirtual 18;
7 ifeq 98 (+91);
```

1
```
10 aload_1;
11 invokevirtual 11;
14 astore_2;
15 aload_2;
16 iconst_4;
17 baload;
18 istore_3;
19 aload_1;
20 invokevirtual 19;
23 i2b;
24 istore 4;
26 iload_3;
27 iconst_1;
28 if_icmpne 37 (+9);
```

2
```
31 iload 4;
33 iconst_1;
34 if_icmpeq 43 (+9);
```

3
```
37 sipush 26368;
40 invokestatic 13;
```

4
```
43 aload_2;
44 iconst_5;
45 baload;
46 istore 5;
48 iload 5;
50 bipush 127;
52 if_icmpgt 60 (+8);
```

5
```
55 iload 5;
57 ifge 66 (+9);
```

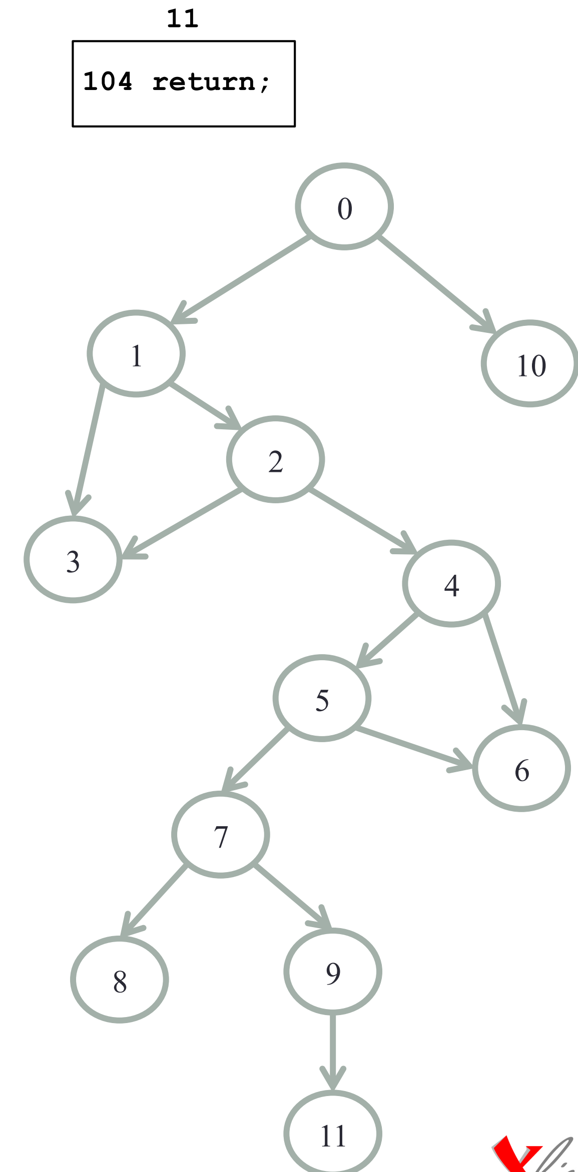
6
```
60 sipush 27267;
63 invokestatic 13;
```

7
```
66 aload_0;
67 getfield 20;
70 iload 5;
72 isub;
73 i2s;
74 ifge 83 (+9);
```

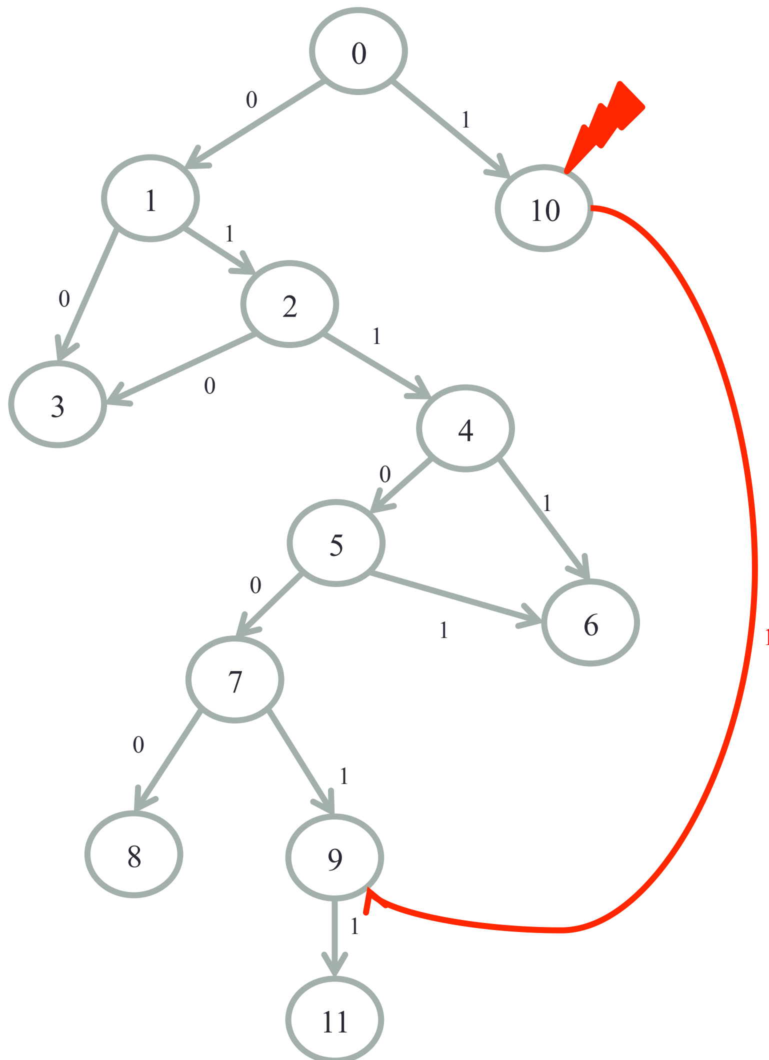
8
```
77 sipush 27269;
80 invokestatic 13;
```

9
```
83 aload_0;
84 aload_0;
85 getfield 20;
88 iload 5;
90 isub;
91 i2s;
92 putfield 20;
95 goto 104 (+9);
```

10
```
98 sipush 25345;
101 invokestatic 13;
```



Path Check (PCh) : example



Path leading to node 9 computed off-card:

0	1	0	1	1	0	0	1

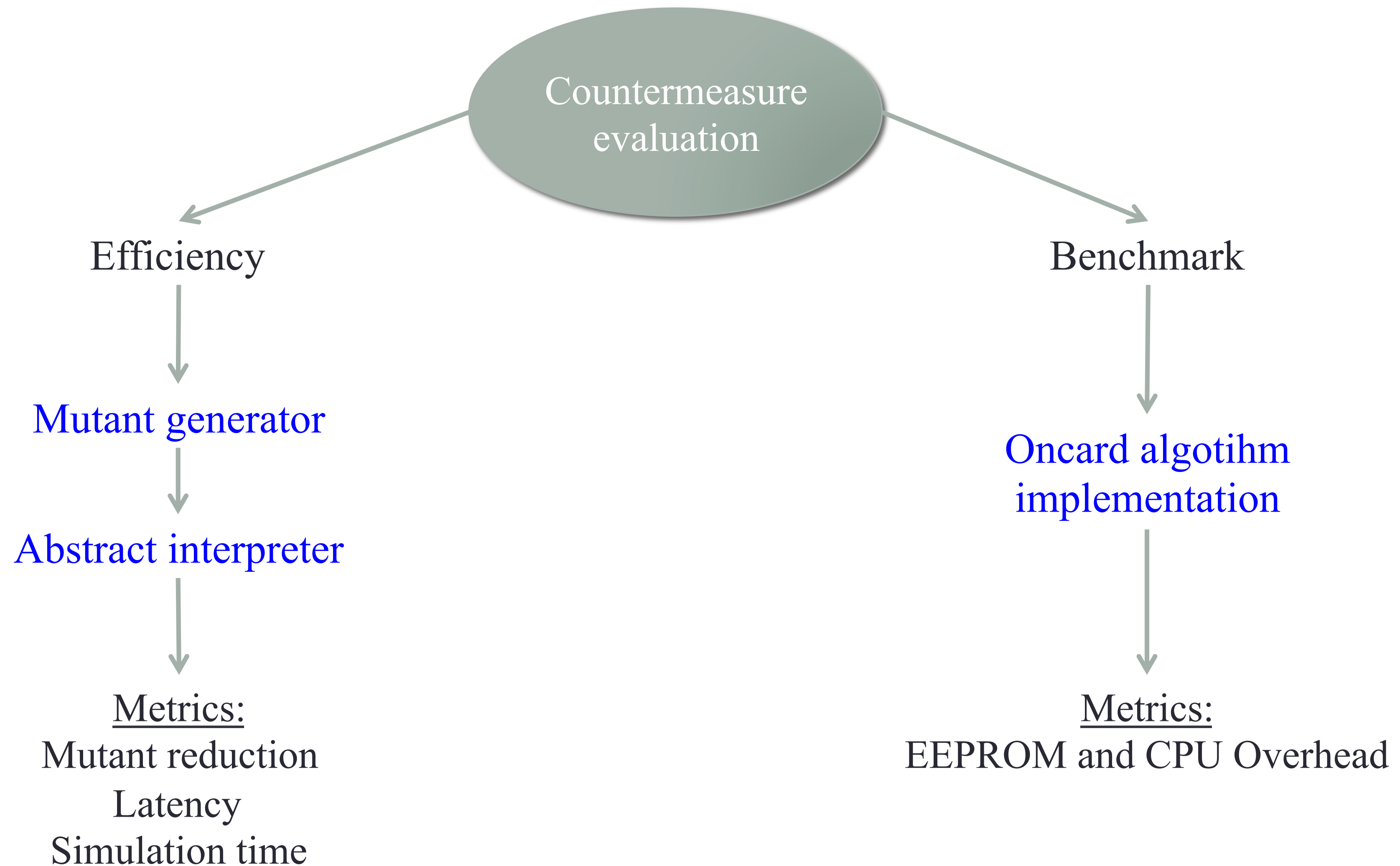
Path leading to node 9 computed on-card

0	1	1	1

Path Check (PCh) : conclusion

- Advantage
  - Allow to detect modifications that influence control flow graph and thus to fight against bypassing crucial tests.
- Drawback
  - Can’t detect a modification that doesn’t influence control flow graph .

Evaluation



Evaluation tools

- Abstract Interpreter
  - Simulate VM behavior during applet interpretation
  - Simulate fault attacks
  - Implement some BCV checks, and the developed countermeasure(s)
- Mutant generator
  - Application vulnerability analysis tool that allows the generation of a sensitivity report
- Evaluation board
  - AT91 EB40A arm7 with SimpleRTJ virtual machine

Efficiancy: mutants reduction

* Path Check
** Field of bit
*** Basic block

Reference model
440
-
5 s

	Partial BCV	PCh*	FoB**	BB***
Mutant reduction	81%	87%	93%	100%
Average latency	1,51	3	2,42	2,82
Simulation time	5 s	2 s	4 s	1 s

Wallet non encrypted memory - 472 attacks on 236 instructions

Xlim institut de recherche

Efficiancy: mutants reduction

Reference model
3743
-
810 s

	Partial BCV	PCh	FoB	BB
Mutant reduction	54%	56%	73%	100%
Average latency	1,01	13,06	8,61	10,53
Simulation time	605 s	218 s	121 s	95 s

Wallet encrypted memory - 60892 attacks on 236 instructions

Benchmark: maximum resources consumption

	CPU overhead	EEPROM	ROM
Field of bit	$< 5\%$	$\approx 3\ \%$	$\approx 1\%$
Basic block	$< 5\%$	$\leq 5\%$	$\approx 1\%$
Path check	$< 8\%$	$\leq 10\%$	$\approx 1\%$

Metrics obtained with all methods tagged

Conclusions

- The exposed countermeasure
  - Respectful of the Java Card specification
  - Brings security interoperability
- It is affordable for the card
  - Memory consumption
  - CPU overhead
- Less work for developers
  - Only need to use an annotation
- Lightweight change of the VM interpreter

Thank you for your attention
Any questions ?

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