Automatic detection of fault attacks and countermeasures

Presenter:
Ahmadou Al Khary SERE

Co-authors:
Julien Iguchi-Cartigny
Jean-Louis Lanet

Team:
Smart Secure Device

Institute:
XLIM Laboratory

WESS Workshop'09
Grenoble

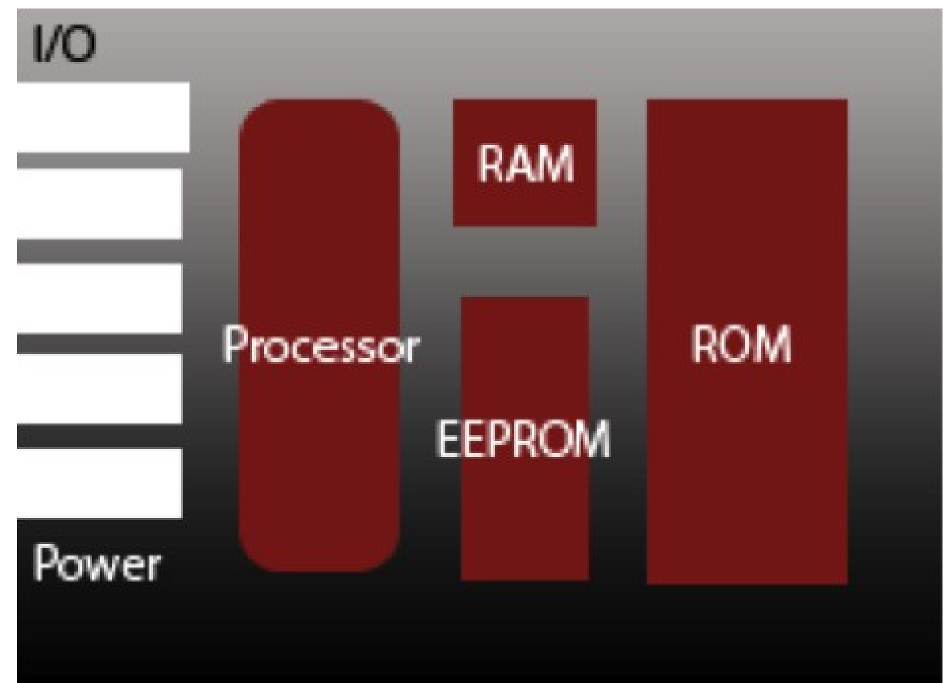
Outline

- Introduction
- Fault Model
- Contribution
- Conclusion

Introduction

Introduction – smart card

- Smart card
  - Small computer
  - Limited capacities
- Application
  - Telephony
  - Credit card
  - Pay TV
  - And More



Introduction – Java Card

- Java Card
  - Designed to fit in a smart card
  - Based on Java
  - Implement a subset of Java
- Java Card specification
  - API
  - Java Virtual Machine (JVM)
  - Java Runtime Environment (JRE)

Introduction – Java Card

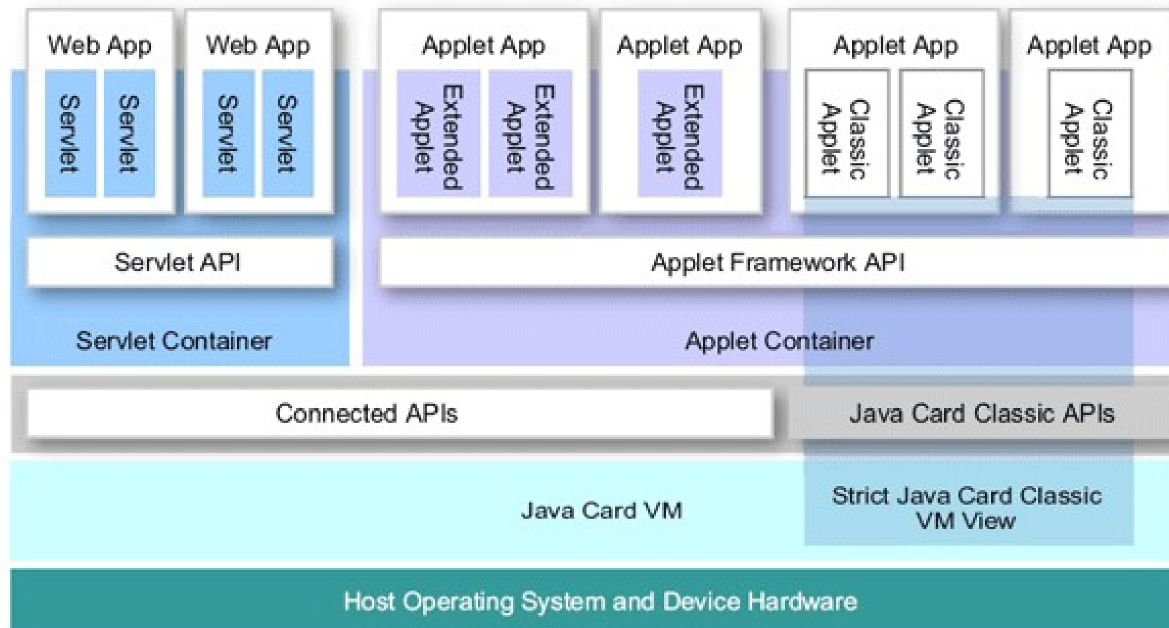
- Java Card 3.0.1
  - The last version of the specifications
  - Some real novelty
  - 2 releases of the specifications
  - Modern high-end smart card
- Java Card classic edition
  - No major evolution since Java Card 2.2.2

Introduction – Java Card

- Java Card Connected edition
  - Classic Applets
    - APDU + Backward compatibility
  - Extended Applets
    - APDU + String + Thread + GCF
  - Servlet Applications
    - HTTP/HTTPS + Servlet
  - Classfile compliant

Introduction – Java Card

Java Card 3 Architecture



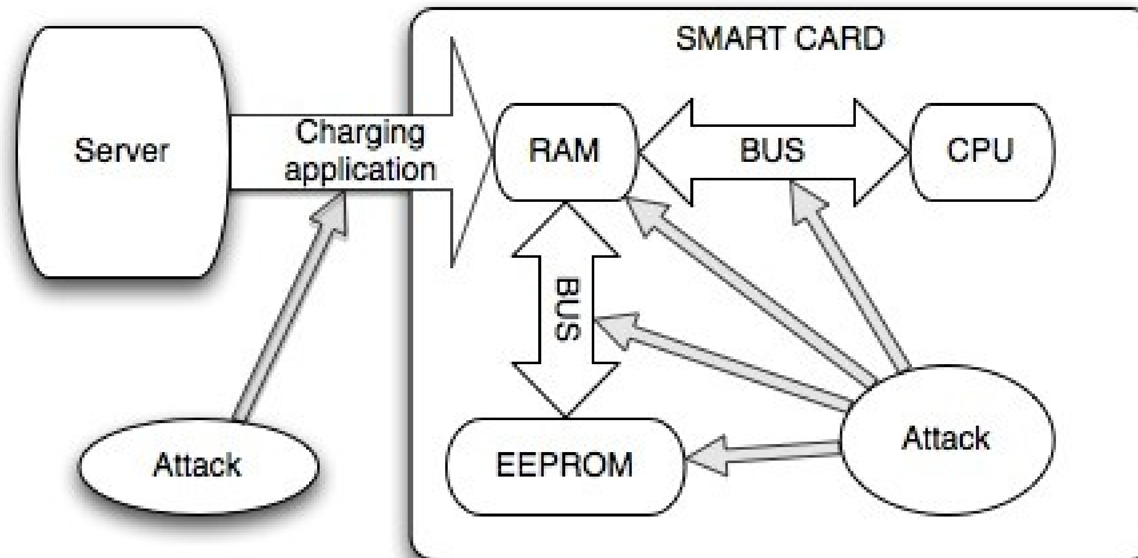
Introduction - Fault Attacks

- Physical attacks
- Disturb the environment of the card
  - Electromagnetic filed
  - Optical
  - Electrical
  - Heat
  - Clock glitch
  - Laser

Introduction - Fault Attacks

- Many consequences on the chip
  - Register disturbance
    - Program counter
  - Code modification
    - Changing bytecode
    - Changing operand
  - Control flow perturbation
    - Jump some logical verification
      - PIN code verification
      - cryptographic key verification, etc

Introduction - Problematic



Fault model

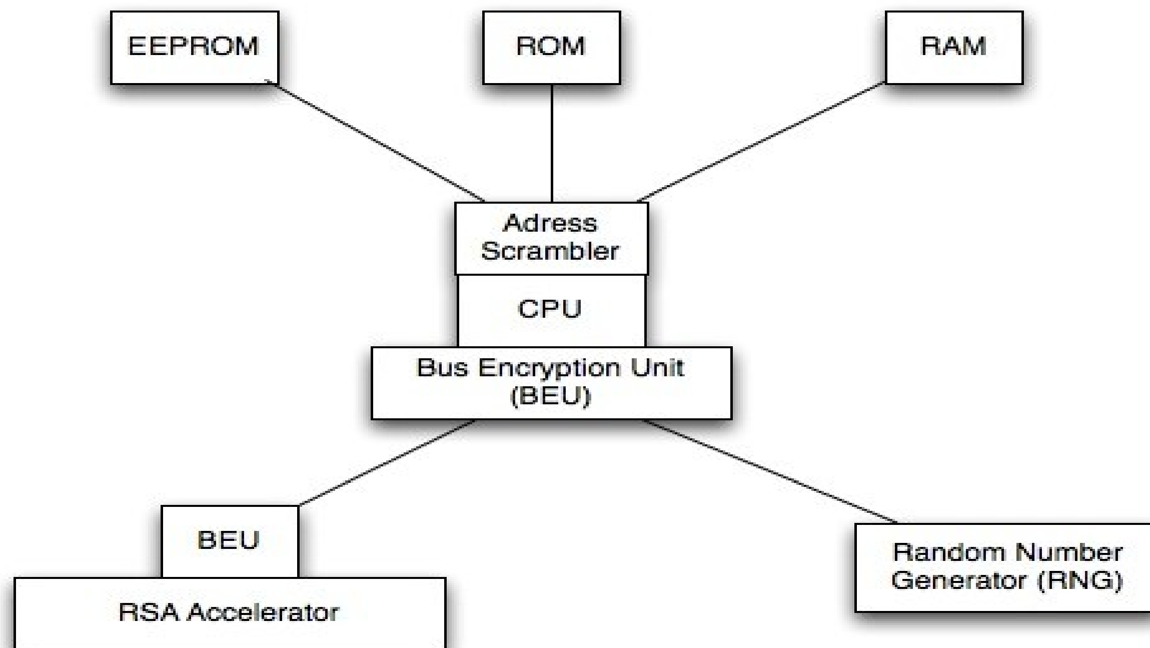
Fault Model

- Fault Model
  - Classification of fault attack
- Different criteria
  - Location
  - Timing
  - Precision
  - Fault type

Fault Model

- The opponent
  - Precise control over location
  - Precise control over timing
  - Capacity to set the store bit to any value
- Smart Card
  - Modern high-end smart card
  - Implement some hardware countermeasures

Fault Model



Architectural sketch of the considered smart card

Fault Model

- Existing fault model

Fault models	Location	Timing	Precision	Fault type
Precise bit error	Total control	Total control	bit	Bit set or reset (bsr)
Precise byte error	Total control	Total control	byte	Bsr, random fault (rf)
Unknown byte error	Loose control	Loose control	byte	Rf, bsr
Unknown error	No control	No control	variable	rf

Fault Model

- Reaction of the card
  - Unprotected card
    - Fail to detect attack
  - Protected card
    - Detect attack and take measures
- Chosen fault model
  - Precise byte error
  - Attacker can make one fault at a time
  - Byte set to 0x00, 0xFF or a random value

Fault Model

Example of fault attack

Before attack

bytecode	Byte representation	Java code

0 : aload_0
1 : getfield #4
4 : invokevirtual #18
7 : ifne 16
10 : sipush 25345
13 : invokestatic #13
16 : ...
	0 : 15
	1 : 83 00 04
	4 : 8B 00 23
	7 : 97 00 10
	10 : 19 63 01
	13 : 8D 00 0D
	16 : ...
		private void debit(APDU apdu) {
		// access authentication
		if (! pin.isValidated())
		ISOException.throwIt (SW_PIN_VERIFICATION_REQUIRED);
		// make the debit operation
		....
		}

After attack

bytecode	Byte representation	Java code

0 : aload_0
1 : getfield #4
4 : invokevirtual #18
7 : ifne 16
10 : nop
11 : nop
12 : nop
13 : nop
14 : nop
15 : nop
16 : ...
	0 : 15
	1 : 83 00 04
	4 : 8B 00 23
	7 : 97 00 10
	10 : 00
	11 : 00
	12 : 00
	13 : 00
	14 : 00
	15 : 00
	16 : ...
		private void debit(APDU apdu) {
		// access authentication
		if (! pin.isValidated())
		
		// make the debit operation
		....
		}

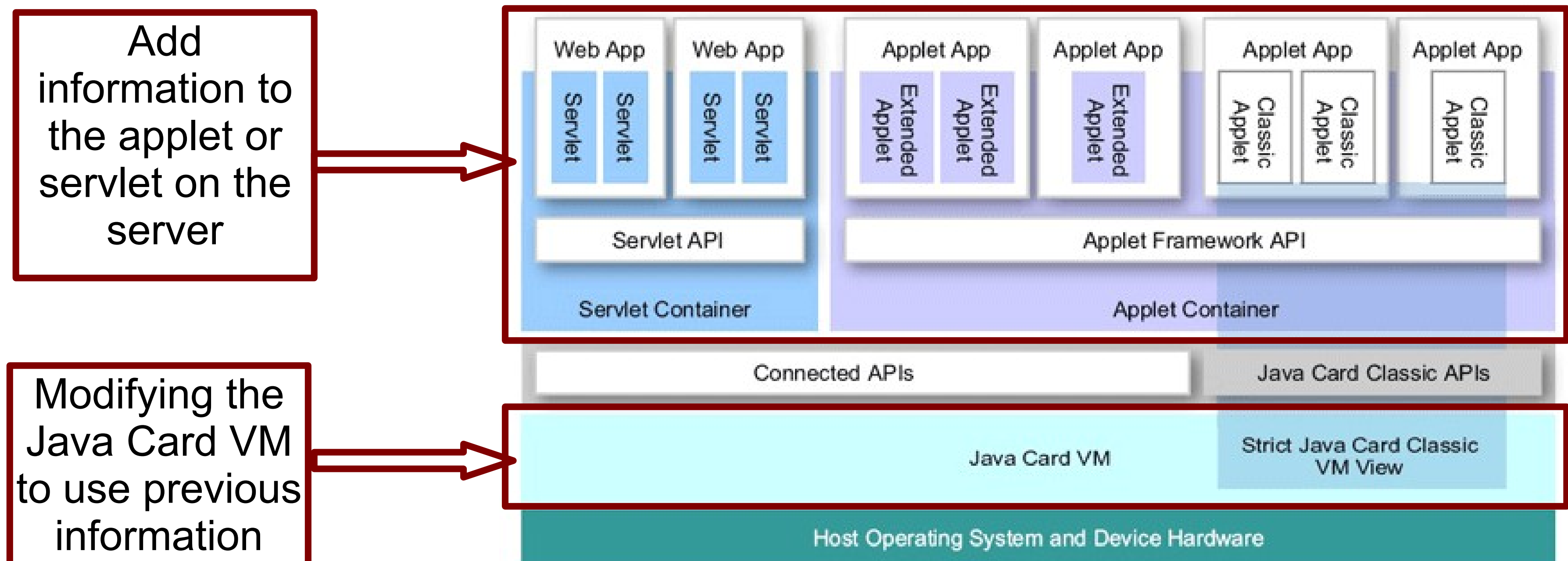
Contributions

Contributions - Countermeasures

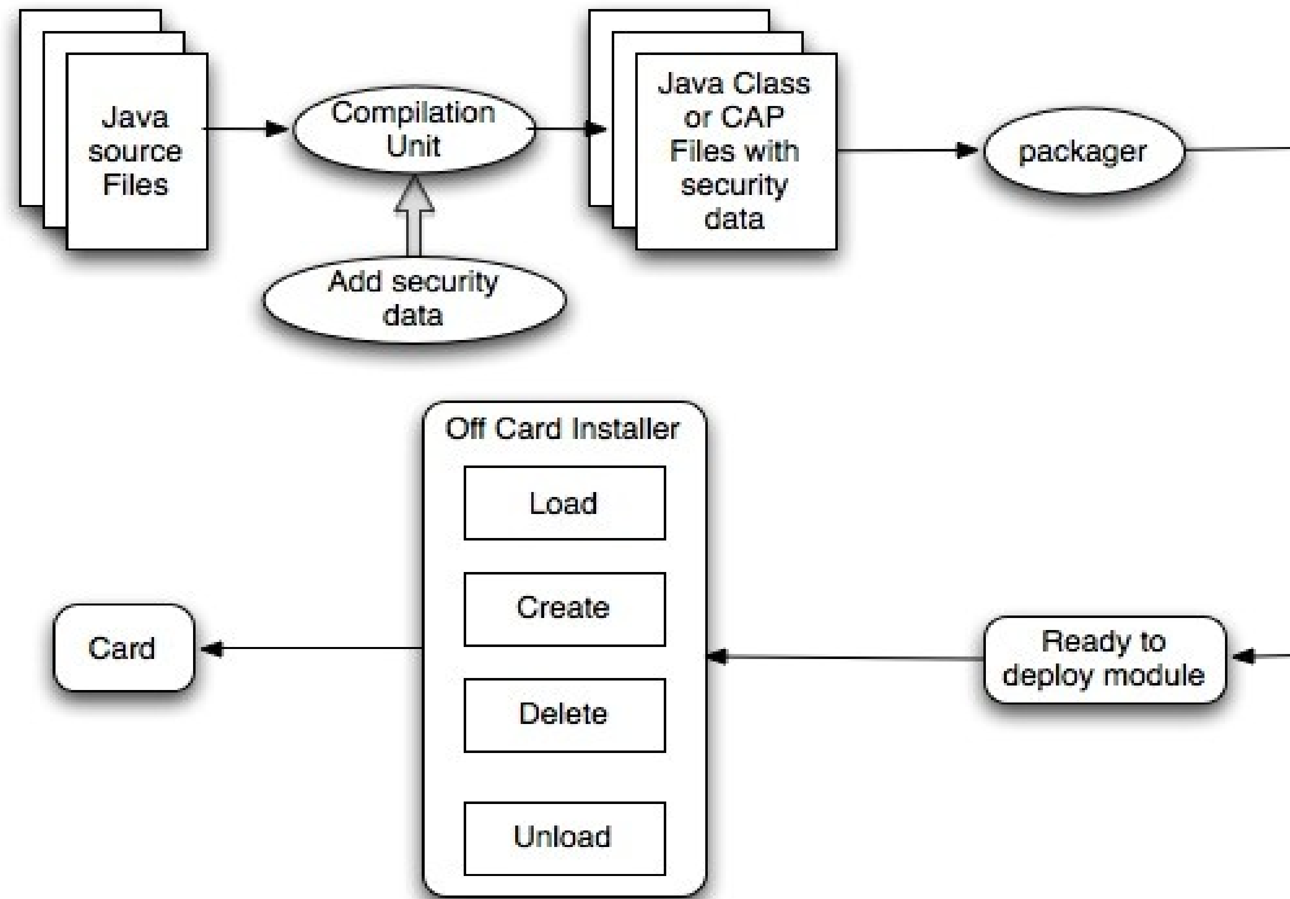
- Hardware Countermeasures
  - Detectors (light, supply voltage, frequency detector)
  - Hardware redundancy, AS, RNG, BEU, etc.
- Software Countermeasures
  - Cryptographic countermeasures
  - Applicative countermeasures
  - System countermeasures
  - Hybrid countermeasures

Contributions - Countermeasures

- Hybrid Countermeasures (our approach)



Contribution - Propositions



Contribution - Propositions

- Programmer point of view
  - Mark important method to protect
    - Java annotation
    - Reducing the size of information add to the code
    - Reducing the execution time

@Retention { RetentionPolicy.CLASS }
public @ interface secure {
    type security ();
    public static enum type (FIELDOFBIT, XOR, ...)
}
	private void debit(APDU apdu) {
	    // access authentication
	    if (! pin.isValidated())
	        ISOException.throwIt (SW_PIN_VERIFICATION_REQUIRED);
	    // make the debit operation
	    ....
	}

Contribution – Field of bit

- Field of bit
  - Goal:
    - Verify the control flow
    - Detect code change
    - Detect number of operand change
  - Principle
    - Offcard:
      - Tagging the code
    - Oncard:
      - During execution
      - Check concordance between the tag and the interpret code

Contribution – Field of bit

- The tagging process

Java Code	Byte representation	Associate Tag
0 : aload_0 1 : getfield #4 4 : invokevirtual #18 7 : ifne 16 10 : sipush 25345 13 : invokestatic #13 16 : ...	0 : 15 1 : 83 00 04 4 : 8B 00 23 7 : 97 00 10 10 : 19 63 01 13 : 8D 00 0D 16 : ...	X X R R X R R X R R X R R X R R ...

- Add to the classfile the table:

  X X R R X R R X R R X R R X R R

- Charge classfile to the card

Contribution – Field of bit

- On card:
  - Foreach PC
    - Check the byte corresponding to this pc (VM [PC])
    - For an opcode
      - if VM [PC] == X then nothing happened, the VM can continue
      - If VM [PC] != X then a fault occured, the VM will stop his execution
    - For an operand
      - If VM [PC] == R then nothing happened, the VM can continue
      - If VM [PC] != X then a fault occured, the VM will stop his
  - On the previous example

Contribution – Field of bit

Before attack

bytecode	Byte representation	Field of bit	Java code

0 : aload_0
1 : getfield #4
4 : invokevirtual #18
7 : ifne 16
10 : sipush 25345
13 : invokestatic #13
16 : ...
	0 : 15
	1 : 83 00 04
	4 : 8B 00 23
	7 : 97 00 10
	10 : 19 63 01
	13 : 8D 00 0D
	16 : ...
		X
		X R R
		X R R
		X R R
		X R R
		X R R
		...
			private void debit(APDU apdu) {
			// access authentication
			if (! pin.isValidated())
			ISOException.throwIt (SW_PIN_VERIFICATION_REQUIRED);
			// make the debit operation
			....
			}

After attack

bytecode	Byte representation	Field of bit	Java code

0 : aload_0
1 : getfield #4
4 : invokevirtual #18
7 : ifne 16
10 : nop
11 : nop
12 : nop
13 : nop
14 : nop
15 : nop
16 : ...
	0 : 15
	1 : 83 00 04
	4 : 8B 00 23
	7 : 97 00 10
	10 : 00
	11 : 00
	12 : 00
	13 : 00
	14 : 00
	15 : 00
	16 : ...
		X
		X R R
		X R R
		X R R
		X
		X
		X
		X
		X
		X
		...
			private void debit(APDU apdu) {
			// access authentication
			if (! pin.isValidated())
			
			// make the debit operation
			....
			}

Contribution – Field of bit

- Advantages
  - Immediate detection of code modification
  - Size of application increase about 3%
  - JVM interpreter files increase about 1%
  - Increase of time need for execution under 5%
- Drawbacks
  - Can only detect when the replacement is indistinguishable
    - The substitute instruction have
      - Same number of operand
      - Same effect on the operand stack

Contribution - XOR

- XOR detection mechanism
  - Goal
    - Verify the integrity of the code
    - Detect changes that can occur in the code
  - Principle
    - Cut the code in basic bloc
    - Off card:
      - Compute check values for each block of a mark method
      - Save these values into classfile
    - On card:
      - Compute check value again (during runtime)
      - Compare check value with computed value offcard

Contribution - XOR

- A basic bloc
  - A set of instructions
  - One entry point
  - One exit point
  - No jump inside
- The check value
  - XOR instruction between all bytes of the bloc

Contribution - XOR

- Some checks are made by the interpreter
  - At the beginning of a basic bloc
    - Check if the instruction can really begin a basic bloc
  - At the end of the basic bloc
    - Check if the instruction can really end a basic bloc
    - Check if the check values match with themselves
  - If one of the previous test fail
    - Then an attack occurred

Contribution - XOR

- In the previous example
  - Before attack
    - Only one block (0, 13)
    - Check value for this block: 0x77
  - After attack
    - Only One block (0, …)
    - Check value for PC 0 to 13: 0x0F
  - First detection
    - The pc that end the block are not the same => attack occurred
  - Second
    - Check values are different => attack occurred

Contribution - XOR

- Advantages
  - Detection of any modification (opcode and operand)
  - Size of application increase about 5 %
  - JVM interpreter files increase about 1%
  - Increase of time need for execution under 5%
- Drawbacks
  - More operation than the previous method
  - Problem with loop execution
  - Attacker can obtain information before the end of a block

Conclusion

Conclusion - Perspectives

- Solve the problem of indistinguishable opcodes
  - Find in applet associate opcodes
  - Compress them
    - Reduce the execution time
    - Reduce the number of requisite opcodes

  => Reduce the number of indistinguishable opcodes
- New countermeasure
  - Use of error detection and correction code
    - At least detect changes
    - At most correct them

Conclusion - closing comments

- Countermeasures exposed
  - Not too heavy for the card
    - Increase of classfile size and execution time < 5%
  - Less work for the developer to protect is application
    - Only need to tag the code
  - No major shaking of the applet conception chain

Conclusion

Thank you for your attention
Any questions ?