SOS
An innovative secure system architecture

Michel Agoyan, Bruno Robisson, Minh Huu Nguyen CEA LETI /DSIS/LCS/SAS
Pirouz Bazargan-Sabe UPMC-LIP6
Guillaume Phan TRUSTED-Logic
Sébastien Le Henaff VIACCESS

CEA-LETI/DSIS/SCME/CCS
SAS Systèmes et Architectures Sécurisés

Cryptarchi June 2010
Plan

1. Introduction
2. Hardware architecture
3. Hardware CMs
4. Design method
5. Conclusion

michel.agoyan@cea.fr
SOS An innovative secure system architecture
**SOS** : **Smart On Smart** is a project funded by the "Agence Nationale pour la Recherche" : ANR-07-SESU-014 and supported by SCS cluster

**Purpose**

- Propose an innovative secure system architecture for embedded device such as smart card and prove it.
- **An innovative secure system architecture to:**
  - Help the design of the security strategy
  - Improve the smartness of the security strategy
  - Reconcile availability with security
  - Improve the performance
The concept

Split off

CLASSIC SC ARCHITECTURE
SECURITY STRATEGY + DATA PROCESSING

HOST SYSTEM
APP
Security strategy
CMs

HOST SYSTEM
VM
Security strategy
CMs

HOST SYSTEM
HARDWARE
CMs
The concept

Split off

CLASSIC SC ARCHITECTURE
SECURITY STRATEGY + DATA PROCESSING

HOST SYSTEM
APP
CMs

HOST SYSTEM
VM
CMs

HOST SYSTEM
HARDWARE
CMs

SPLIT OFF

Security strategy

michel.agoyan@cea.fr
SOS An innovative secure system architecture
The concept

Split off

SOS ARCHITECTURE

DATA PROCESSING

HOST SYSTEM
VM
CMs

HOST SYSTEM
APP

SECURITY STRATEGY

AUDIT SYSTEM
OS
Security strategy

AUDIT SYSTEM
HARDWARE

HOST SYSTEM
HARDWARE
CMs

michel.agoyan@cea.fr

SOS An innovative secure system architecture
Advantages

- Regroup all the security management under the responsibility of an unique entity
  - Security policy more flexible, adaptable
  - Opportunity to design a smarter security policy
- Fault attacks path more complex → increase the security level
- Improve the performance → 2 systems running in parallel

Fit the initial main objectives

Drawbacks

- Impact on the cost
- New concept → new paths for attack?
The model

Why?
- Practical approach to test the concept
- Prove it

How?
- Choose an application requiring different security levels and performance → pay tv
- Conditional Access System:
  - Principle: digital audio/video stream can be unscrambled if the right is owned by the smart-card.
  - 3 main classes of command are used by the CAS:
    - Subscription management (keys & rights writing): Very sensitive
    - Unscrambling (generating a control word): Sensitive
    - Subscriber operations (parental control): Not very sensitive
security policy

The challenge: availability versus security

“Naive approach”: Hardcoded “attack/error” border

- “Ideal” border
- False positive
- CRC trigger
- “Raw” border
- Attacks
- False negative
- Voltage sensor trigger
- Error

Border = attack (or error) signature
Plan

1. Introduction
2. Hardware architecture
3. Hardware CMs
4. Design method
5. Conclusion
App requests to the VM a security level
HS ↔ AS communication: normal exec

1. App requests to the VM a security level
2. VM forwards the request to AS
HS ↔ AS communication : normal exec

1. App requests to the VM a security level
2. VM forwards the request to AS
3. Depending on current context & request AS computes the actions to perform.

HOST SYSTEM
APP

HOST SYSTEM
VM

HOST SYSTEM
HARDWARE

CMs

ICU

AUDIT SYSTEM
OS

AUDIT SYSTEM
HARDWARE

HOST SYSTEM
VM

HOST SYSTEM
APP

HS ↔ AS communication : normal exec
HS ↔ AS communication: normal exec

1. App requests to the VM a security level
2. VM forwards the request to AS
3. Depending on current context & request AS computes the actions to perform.
4. AS could decide to parameter some hardware CMs
HS ↔ AS communication: normal exec

1. App requests to the VM a security level
2. VM forwards the request to AS
3. Depending on current context & request AS computes the actions to perform.
4. AS could decide to parameter some hardware CMs
5. AS could also ask to VM to apply software CMs.
HS ↔ AS communication : normal exec

1. App requests to the VM a security level
2. VM forwards the request to AS
3. Depending on current context & request AS computes the actions to perform.
4. AS could decide to parameter some hardware CMs
5. AS could also ask to VM to apply software CMs.
6. AS resumes the execution
A sensor event occurs

HS ↔ AS communication: sensor event
A sensor event occurs
An interruption is raised on HS and AS through the ICU
A sensor event occurs
2. An interruption is raised on HS and AS through the ICU
3. Depending on current context AS computes the actions to perform.
A sensor event occurs

2. An interruption is raised on HS and AS through the ICU

3. Depending on current context AS computes the actions to perform.

4. AS could decide to parameter some hardware CMs
A sensor event occurs
2. An interruption is raised on HS and AS through the ICU
3. Depending on current context AS computes the actions to perform.
4. AS could decide to parameter some hardware CMs
5. AS waits for HS interruption ack
1. A sensor event occurs
2. An interruption is raised on HS and AS through the ICU
3. Depending on current context AS computes the actions to perform.
4. AS could decide to parameter some hardware CMs
5. AS waits for HS interruption ack
6. AS clears the interruption
A sensor event occurs
An interruption is raised on HS and AS through the ICU
Depending on current context AS computes the actions to perform.

AS could decide to parameter some hardware CMs
AS waits for HS interruption ack
AS clears the interruption
AS could also asks to VM to apply software CMs and resumes the execution
Plan

1. Introduction
2. Hardware architecture
3. Hardware CMs
4. Design method
5. Conclusion
Follows the basic rule of SOS: “Split off”

- Sensitive data are processed by the Host System ⇒ HS embeds CMs
- Security is under the control of the Audit System ⇒ AS controls CMs
- Rem: To demonstrate SOS concept ⇒ chose hardware CMs that are parametrable, switchable, and have an impact on CPU performance

Hardware interface

- 4KB dedicated to map CMs registers into memory space of AS
  - A control register: on/off + parameter
  - A status register
- Interrupts from the sensors into the Host system are routed to the Interrupt Control Unit of the Audit system
ALU protection against fault attacks

**Principle**

- Based on Idle Hardware: \( ALU = \sum \text{Functional Units} \) (Adder, multiplier, shifter) \( \Rightarrow \) Time redundancy

```plaintext
if optypeA \neq optypeB then
    Recompute A and in parallel, Execute B
else
    Stall the processor and Recompute A
end if
```

Compare the recomputed result of A with the previously stored...
ALU protection against fault attacks: Results

% instructions using ALU

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Logic_Instr</th>
<th>Arithmetic_Instr</th>
<th>No FU_Instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base ALU</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Fault tolerant ALU</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Scheduling improvement

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Base ALU</th>
<th>Fault tolerant ALU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Clock Period (ns)</td>
<td>1.050</td>
<td>1.056 (+0.57%)</td>
</tr>
</tbody>
</table>

Hardware overhead

<table>
<thead>
<tr>
<th>ALU Type</th>
<th>Number of Slices</th>
<th>Max Clock Period (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base ALU</td>
<td>2146</td>
<td>1.050</td>
</tr>
<tr>
<td>Fault tolerant ALU</td>
<td>2584 (+20.4%)</td>
<td>1.056 (+0.57%)</td>
</tr>
</tbody>
</table>

michel.agoyan@cea.fr

SOS An innovative secure system architecture
Random Instruction Injection

A command register programmable by the Audit System

- **Based on**: Injecting random instructions at random places during the execution [Smart Random Code Injection to Mask Power Analysis Based Side Channel Attacks, Jude Angelo Ambrose, University of New South Wales Sydney, Australia]

- Setting and clearing a flag delimit the block frame to protect

- **SET_FLAG**: starts to generate random instructions
  - **N**: the maximum number of random instructions injected between two regular instructions
  - **D**: the maximum number of regular instructions skipped

- **RESET_FLAG**: stops the injection of random instructions

michel.agoyan@cea.fr  
SOS An innovative secure system architecture
Random Instruction Injection

A command register programmable by the Audit System

- Based on: Injecting random instructions at random places during the execution [Smart Random Code Injection to Mask Power Analysis Based Side Channel Attacks, Jude Angelo Ambrose, University of New South Wales Sydney, Australia]

- Setting and clearing a flag delimit the block frame to protect

- SET_FLAG: starts to generate random instructions
  - N: the maximum number of random instructions injected between two regular instructions
  - D: the maximum number of regular instructions skipped

- RESET_FLAG: stops the injection of random instructions

Limitation of random instruction set

- random register combined with zero register result written back to same random register: $ADD$ $3$, $3$, $0$
Random Instruction Injection: Implementation

Issue

- if the injected instruction uses a register containing a sensitive data → leakage information
**Random Instruction Injection: Implementation**

**Issue**
- If the injected instruction uses a register containing sensitive data → leakage information

**Solution**
- Use 2 register files: the main register file $M$ and the register file $R$ with random values
- Inject a random instruction using operands of $R$

**Example**
- $ADD R5, R3, R0$ → normal instruction
- $ADD R5, R7, R8$ → random instruction
Random Instruction Injection: Results

**Trivial DPA on VCDs without CM**

**Trivial DPA on VCDs with CM**

**Hardware overhead**

<table>
<thead>
<tr>
<th></th>
<th>Slices reg</th>
<th>Slices LUT</th>
<th>used as logic</th>
<th>used as ram</th>
<th>Max Clock Period (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RII</td>
<td>2367</td>
<td>4568</td>
<td>4546</td>
<td>22</td>
<td>5.6ns</td>
</tr>
<tr>
<td>RII</td>
<td>3625</td>
<td>5413</td>
<td>5391</td>
<td>22</td>
<td>7 ns</td>
</tr>
<tr>
<td></td>
<td>53%</td>
<td>18%</td>
<td>18%</td>
<td>0%</td>
<td></td>
</tr>
</tbody>
</table>
The design flow

- Debug on simulation on target
- Simulation
- VHDL SOS models

- Host System Application
- Host System Virtual Machine
- Audit System Strategy Security
- Host System Hardware
- Audit System Hardware

- FPGA Xilinx Virtex-5

SOS An innovative secure system architecture
The security policy is tested using hardware emulation for fault injection
⇒ rules improvement
Security Policy

- The project finishes at the end of 2010.
- The basic framework to build a smarter security is already available.
- Implementation for different applications is possible.
- These new implementations will help us to improve and validate the concept.

Performance

- The main concern of the first implementation of the communication protocol is the security policy in spite of the performance.
- So there are opportunities to exploit the parallelism of the architecture and to reveal the performance.