

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 **S**ystèmes et **A**rchitectures **S**écurisés





Cryptarchi June 2010

Plan

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

The project

- SOS  : **Smart On Smart** is a project funded by the "Agence Nationale pour la Recherche" : ANR-07-SESU-014  and supported by SCS cluster

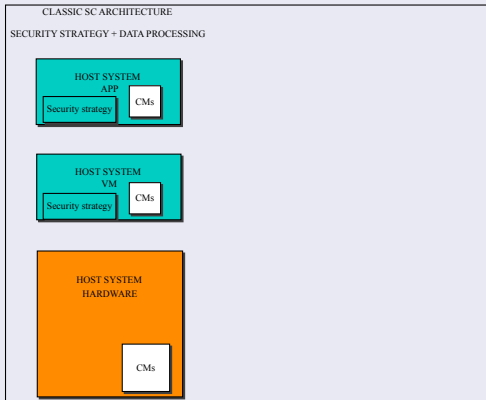


Purpose

- 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

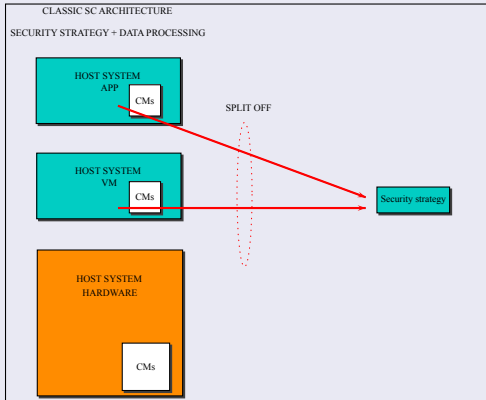
The concept

Split off



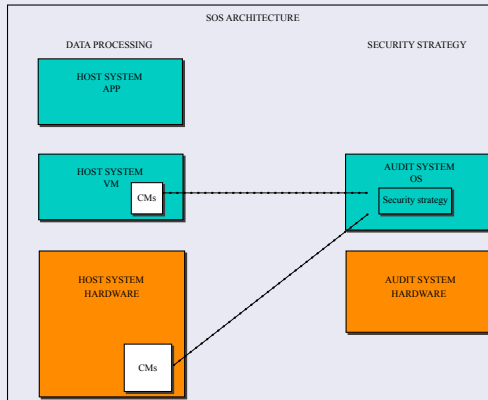
The concept

Split off



The concept

Split off



Split off

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

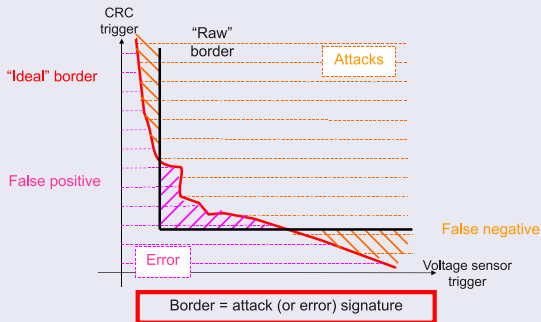
How ?

- 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

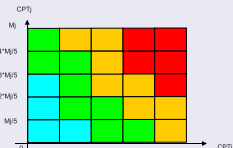


security policy

Graduated reaction



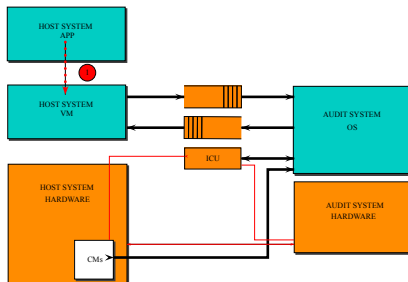
	Redundancy	Scrambling	Reset	Kill
Safe	No	No	No	No
Unsafe	*2	L1	No	No
Critical	*3	L2	Yes	No
Final	-	-	-	Yes



Plan

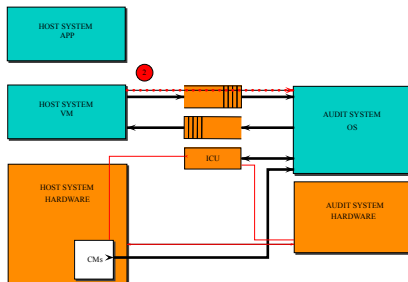
- 1 Introduction
- 2 Hardware architecture
- 3 Hardware CMs
- 4 Design method
- 5 Conclusion

HS ↔ AS communication : normal exec



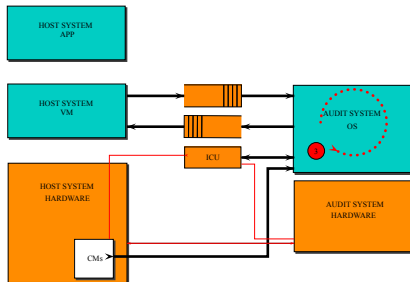
- 1 App requests to the VM a security level

HS ↔ AS communication : normal exec



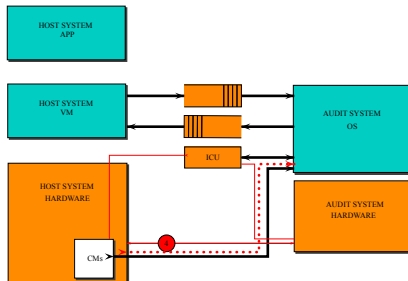
- 1 App requests to the VM a security level
- 2 VM forwards the request to AS

HS ↔ AS communication : normal exec



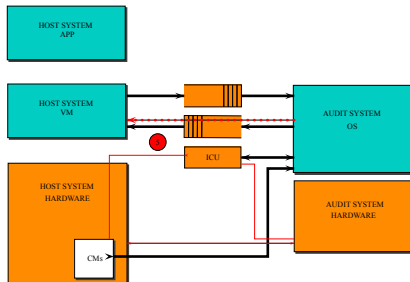
- 1 App requests to the VM a security level
- 2 VM forwards the request to AS
- 3 Depending on current context & request AS computes the actions to perform.

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

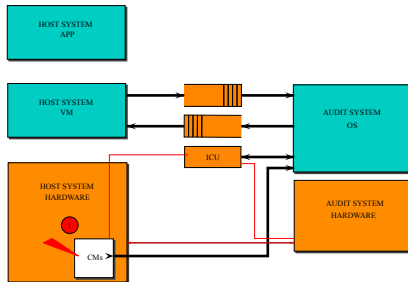
HS ↔ AS communication : normal exec



- 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.

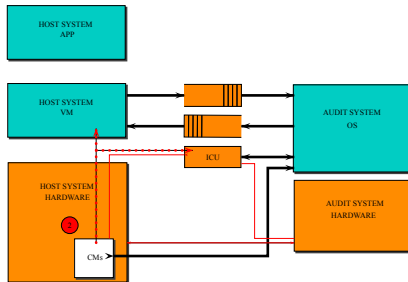
- 4 AS could decide to parameter some hardware CMs
- 5 AS could also ask to VM to apply software CMs.

HS ↔ AS communication : sensor event



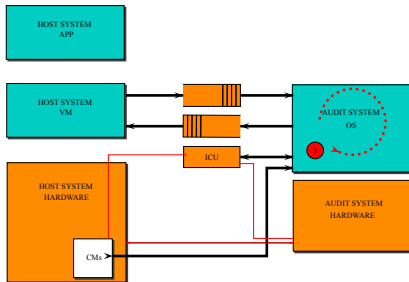
1 A sensor event occurs

HS ↔ AS communication : sensor event



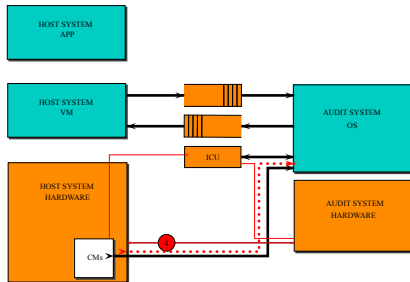
- 1 A sensor event occurs
- 2 An interruption is raised on HS and AS through the ICU

HS ↔ AS communication : sensor event



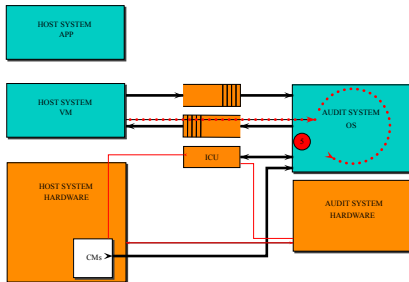
- 1 A sensor event occurs
- 2 An interruption is raised on HS and AS through the ICU
- 3 Depending on current context AS computes the actions to perform.

HS ↔ AS communication : sensor event



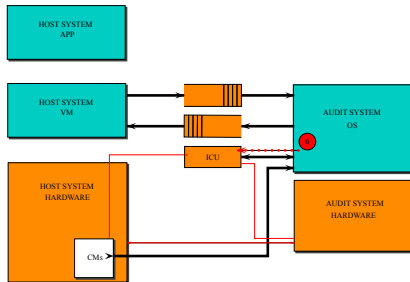
- 1 A sensor event occurs
- 2 An interruption is raised on HS and AS through the ICU
- 3 Depending on current context AS computes the actions to perform.
- 4 AS could decide to parameter some hardware CMs

HS ↔ AS communication : sensor event



- 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

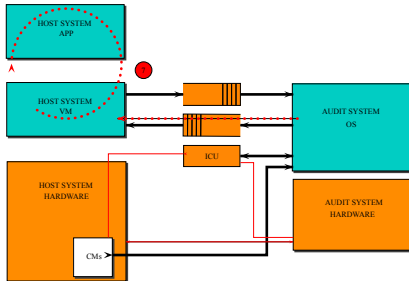
HS ↔ AS communication : sensor event



- 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

HS ↔ AS communication : sensor event



- 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
- 7 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

Interface

Follows the basic rule of SOS : “Split off”

- Sensitive data are processed by the Host System \Rightarrow **HS embeds CMs**
- Security is under the control of the Audit System \Rightarrow **AS controls CMs**
- Rem : To demonstrate SOS concept \Rightarrow 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 **Functional Units** (Adder, multiplier, shifter) \Rightarrow Time redundancy

if optypeA \neq optypeB then

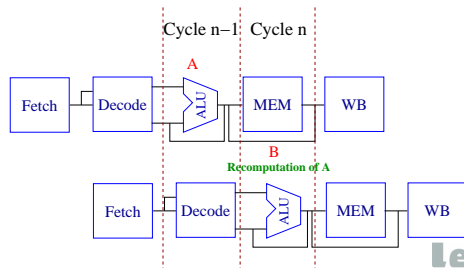
Recompute **A** and in parallel, Execute **B**

else

Stall the processor and Recompute **A**

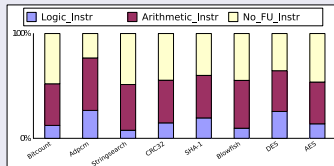
end if

Compare the recomputed result of **A** with the previously stored

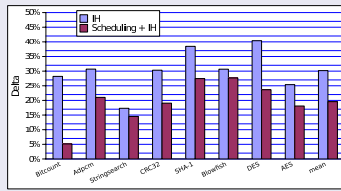


ALU protection against fault attacks : Results

% instructions using ALU



Scheduling improvement



Hardware overhead

	Number of Slices	Max Clock Period (ns)
Base ALU	2146	1.050
Fault tolerant ALU	2584 (+20.4%)	1.056 (+0.57%)

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

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 a sensitive data →
↗ leakage information

Solution

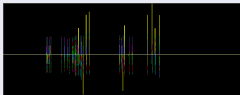
- Use 2 register files : the main register file **M** and the register file **R** with random values
- Inject a random instruction using operands of **R**

example

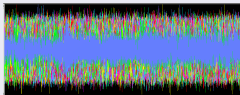
- ADD R5,R3,R0 → normal instruction
- ADD R5,R7,R8 → random instruction

Random Instruction Injection : Results

Trivial DPA on VCDs without CM



Trivial DPA on VCDs with CM



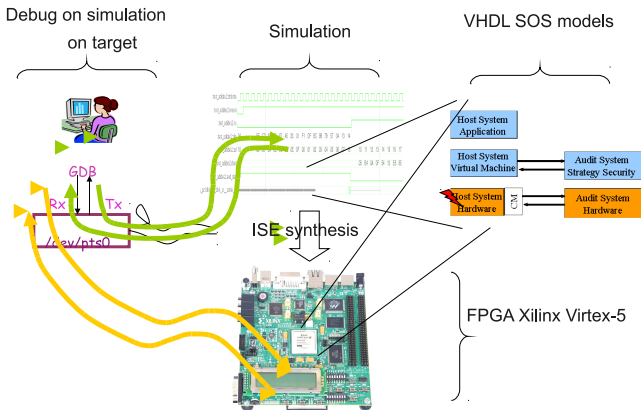
Hardware overhead

	Slices reg	Slices LUT	used as logic	used as ram	Max Clock Period (ns)
R#	2367	4568	4546	22	5.6ns
RII	3625	5413	5391	22	7 ns
	53%	18%	18%	0%	

Plan

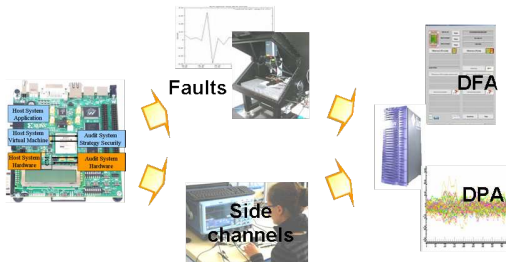
- 1 Introduction
- 2 Hardware architecture
- 3 Hardware CMs
- 4 Design method**
- 5 Conclusion

The design flow



Testing SOS

- The security policy is tested using hardware emulation for fault injection
- \Rightarrow rules improvement



Plan

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

SOS to be continued...

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