Security in modern CPU

Guillaume Bouffard (guillaume.bouffard@ssi.gouv.fr)

Hardware Security Labs – National Cybersecurity Agency of France (ANSSI)

DIENS, ENS, CNRS, PSL University

Workshop SILM — 21 November 2019

Who am I?

Me

- Expert in Embedded System Security (Hardware Security Labs — ANSSI)
- Associate Researcher in the Information Security Group at ENS

Research subjects

- Embedded software security against hardware and software attacks
- Java Card, IC (secure component, micro-controller and SoC).

Aim of this Tutorial

This tutorial aims at introducing an overview of root of trust hardware and software security.

During this tutorial:

- I will focus on security from secure element to system-on-chip
- *No cryptographic implementations will be mistreated during this presentation*

1. Introduction

The Root of Trust

Several features must be executed in a trust environment where is able to:

- host sensitive applications:
  - where sensitive and cryptographic data protection are ensured;
- compute sensitive (as cryptographic) operations:
  - without any leak.

The Root of Trust (cont.)

- The root of trust is a secure environment.

The Root of Trust (cont.)

- The root of trust is a secure environment.
- **Mainly**, it's a secure component.

The Root of Trust (cont.)

- The root of trust is a secure environment.
- **Mainly**, it's a secure component.
- The most populate secure component is the smart card.

The Root of Trust (cont.)

Several software implementations of a secure component exist:

- Hardware secure component emulation:
  - Changing TPMs by secure enclaves, (as ARM TrustZone)
  - **this is not a secure component**.
- Whitebox cryptographic:
  - It's **basically** less secure.
  - How to ensure the security level of those implementations?
  - How and under which condition make those evaluations?

Attacks against Root of Trust

Physical attacks

- Side Channel attacks (timing attacks, power analysis attack, etc.);
- Fault attacks (electromagnetic injection, laser beam injection, etc.).

Software attacks

- Execution of malicious instructions.

Combined attacks

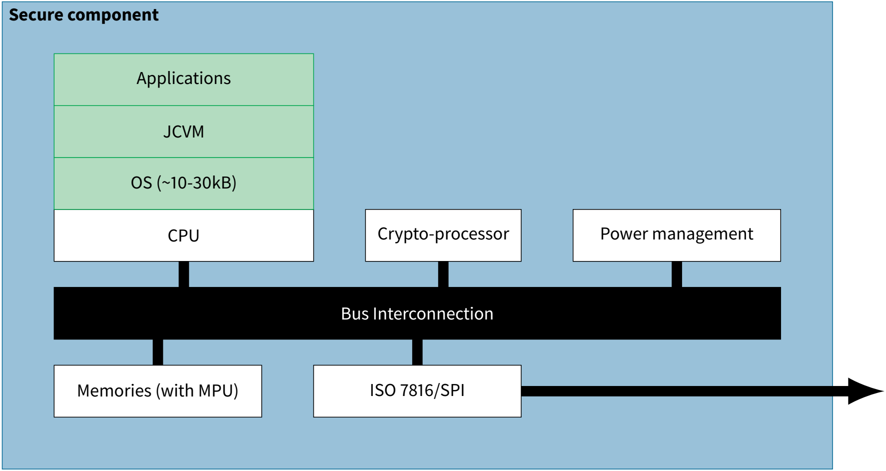
- Mix of physical and software attacks.

The Secure Component?

A secure component is a component with securities features:

- A micro-controller with 1-core CPU and limited-resources;
- Confidentiality and integrity of the flash memory data;
- Random number generator;
- Cryptographic accelerators;
- Detect probing attacks or signal corruption;
- Side channel attacks protection;
- Hardened software.

The Secure Component? (cont.)



How to ensure security level of Secure Component?

- Customers specify the security requirements.
- Developers implement security requirements in the product.
- ITSEFs evaluate the product security level.
- Certification Body certify products and checks each step of the evaluation process.

How to ensure security level of Secure Component?

- Customers specify the security requirements.
- Developers implement security requirements in the product.
- ITSEFs evaluate the product security level.
- Certification Body certify products and checks each step of the evaluation process.

A scheme: the Common Criteria

- Common Criteria is an international standard (ISO/IEC 15408) for certification of secure products.
- International recognition

How to ensure security level of Secure Component?

- Customers specify the security requirements.
- Developers implement security requirements in the product.
- ITSEFs evaluate the product security level.
- Certification Body certify products and checks each step of the evaluation process.

A scheme: the Common Criteria

- Common Criteria is an international standard (ISO/IEC 15408) for certification of secure products.
- International recognition
- Evaluation area:
  - Smartcards & similar devices
  - Hardware Devices with Security Boxes
  - Software

Common Criteria Evaluation Level

- Several certification classes exist:

Level	Description
EAL1	Functionally Tested
EAL2	Structurally Tested
EAL3	Methodically Tested and Checked
EAL4	Methodically Designed, Tested and Reviewed
EAL5	Semiformally Designed and Tested
EAL6	Semiformally Verified Design and Tested
EAL7	Formally Verified Design and Tested

- For each class may be *augmented*:
  - For instance: a smartcard can be evaluated as: `EAL4 + ALC_DVS.2 + AVA_VAN.5`
- Each evaluation is not time constraint.

CC	CSPN
EAL 1 to 7	Only one level
Grey/white box	Black box
International certification recognition	No recognition
No time constraint	25md (+10 for crypto)
Product update during the evaluation	Fixed product version
Developer must provide compliant docs	No specific knowledge
Very expensive (60 to 200k€)	Relatively low cost (25 to 35k€)

CC	CSPN
EAL 1 to 7	Only one level
Grey/white box	Black box
International certification recognition	No recognition
No time constraint	25md (+10 for crypto)
Product update during the evaluation	Fixed product version
Developer must provide compliant docs	No specific knowledge
Very expensive (60 to 200k€)	Relatively low cost (25 to 35k€)

- CPSN-like scheme available in Germany (BSZ — Accelerated Security Certification) and Spain (LINCE).

From the Secure Component to the System of Chip

- Sensitive assets are in and computed on the secure component.
- Secure component are designed (and evaluated) to be tamper-resistant against physical and software attacks.
- System on Chips (SoC) are everywhere:
  - Automotive
  - Smartphone
  - IoT
- Secure component are limited resources devices.
- For sensitive operations where more resources are required, SoCs are used.

Secure Component vs SoC

Smartcard

Mobile device

Same services, different securities

Secure Component vs SoC

Based on a secure component

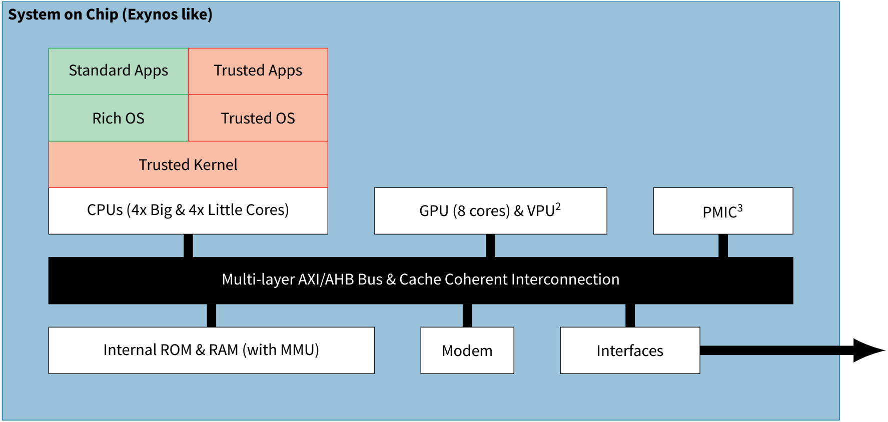
- Simple CPU
- Designed for security
- Certified

Based on a full System on Chip

- Complex CPU
- Designed for performance
- Adding TEE[1] for software security

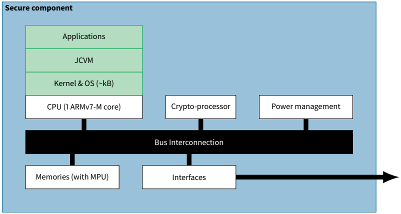
[1] **Trusted Environment Execution**

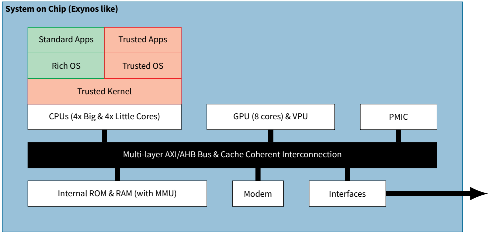
What is a System on Chip?



[2] **Video Processing Unit**
[3] **Power Management Integrated Circuit**

Secure Component vs System on Chip



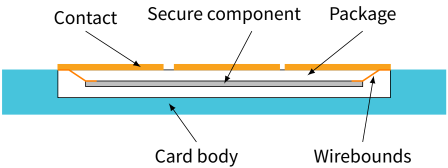


- Run at 4 to 60 MHz
- Not multi-threaded
- Fine engraving > 40 nm
- Constant Voltage & Frequency
- Trusted hardware & apps only
- Hardware mitigation

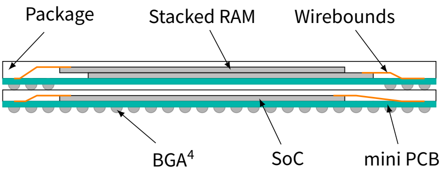
- Run at 300 MHz to 3 GHz
- Multi-threaded
- Fine engraving < 20 nm
- Dynamic Voltage & Frequency management
- Trusted Environment Execution
- No hardware mitigation

The Packaging

Smart card package with secure component



SoC with package on package



[4] **Ball Grid Array**

2. Security of SoC

An overview of state-of-the-art SoC attacks

Injection medium	Physical target	Software target	Software security
Software	RAM	Virtual to physical translation table	Memory partitioning
Glitch voltage	Clock	Key	Cryptography
Laser	Register	Instruction	Secure boot
EM	Bus	Return value	Execution flow integrity
	Cache	Program counter	Confidentiality
	MMU	User rights	
	Pipeline	Data	

An overview of state-of-the-art SoC attacks

Injection medium	Physical target	Software target	Software security
Software →	RAM →	Virtual to physical translation table →	Memory partitioning
Glitch voltage	Clock	Key	Cryptography
Laser	Register	Instruction	Secure boot
EM	Bus	Return value	Execution flow integrity
	Cache	Program counter	Confidentiality
	MMU	User rights	
	Pipeline	Data	

Project Zero attack/Drammer (2015 - 2016) [vdVFL+16]

An overview of state-of-the-art SoC attacks

Injection medium	Physical target	Software target	Software security
Software	RAM	Virtual to physical translation table	Memory partitioning
Glitch voltage	Clock	Key	Cryptography
Laser	Register	Instruction	Secure boot
EM	Bus	Return value	Execution flow integrity
	Cache	Program counter	Confidentiality
	MMU	User rights	
	Pipeline	Data	

Project Zero NaCl/Rowhammer on TrustZone (2015) [Car17]

An overview of state-of-the-art SoC attacks

Injection medium	Physical target	Software target	Software security
Software	RAM	Virtual to physical translation table	Memory partitioning
Glitch voltage	**Clock**	Key	**Cryptography**
Laser	Register	**Instruction**	Secure boot
EM	Bus	Return value	Execution flow integrity
	Cache	Program counter	Confidentiality
	MMU	User rights	
	Pipeline	Data	

ClkScrew (2017) [TSS17]

An overview of state-of-the-art SoC attacks

Injection medium	Physical target	Software target	Software security
Software	RAM	Virtual to physical translation table	Memory partitioning
Glitch voltage	Clock	Key	Cryptography
Laser	Register	Instruction	Secure boot
EM	Bus	Return value	Execution flow integrity
	Cache	Program counter	Confidentiality
	MMU	User rights	
	Pipeline	Data	

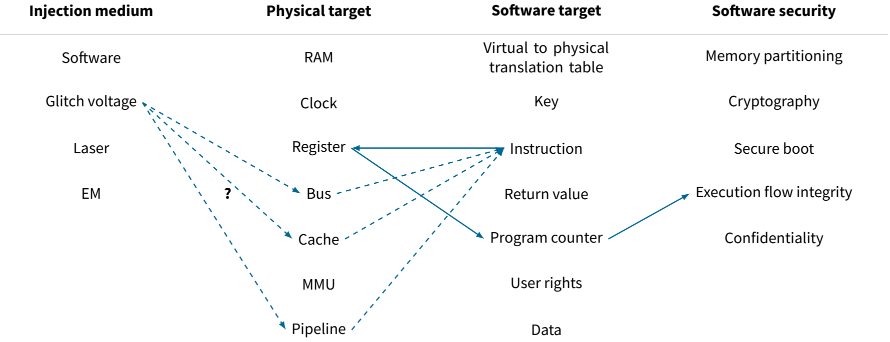
Meltdown attack [LSG+18]

An overview of state-of-the-art SoC attacks

Injection medium	Physical target	Software target	Software security
Software	RAM	Virtual to physical translation table	Memory partitioning
Glitch voltage	Clock	Key	Cryptography
Laser	Register	Instruction	Secure boot
EM	Bus	Return value	Execution flow integrity
	Cache	Program counter	Confidentiality
	MMU	User rights	
	Pipeline	Data	

Spectre attack [KHF+19]

An overview of state-of-the-art SoC attacks



Controlling PC on ARM (2016) [TSW16]

An overview of state-of-the-art SoC attacks

Injection medium	Physical target	Software target	Software security
Software	RAM	Virtual to physical translation table	Memory partitioning
Glitch voltage	Clock	Key	Cryptography
Laser	Register	Instruction	Secure boot
EM	Bus	Return value	Execution flow integrity
	Cache	Program counter	Confidentiality
	MMU	User rights	
	Pipeline	Data	

Attack on PS3

An overview of state-of-the-art SoC attacks

Injection medium	Physical target	Software target	Software security
Software	RAM	Virtual to physical translation table	Memory partitioning
Glitch voltage	Clock	Key	Cryptography
Laser	Register	Instruction	Secure boot
EM	Bus	Return value	Execution flow integrity
	Cache	Program counter	Confidentiality
	MMU	User rights	
	Pipeline	Data	

Attack on Xbox 360 (2015) [Bla15]

An overview of state-of-the-art SoC attacks

Injection medium	Physical target	Software target	Software security
Software	RAM	Virtual to physical translation table	Memory partitioning
Glitch voltage	Clock	Key	Cryptography
Laser →	Register	Instruction	Secure boot
EM	Bus	Return value	Execution flow integrity
	Cache	Program counter	Confidentiality
	MMU	User rights	
	Pipeline	Data	

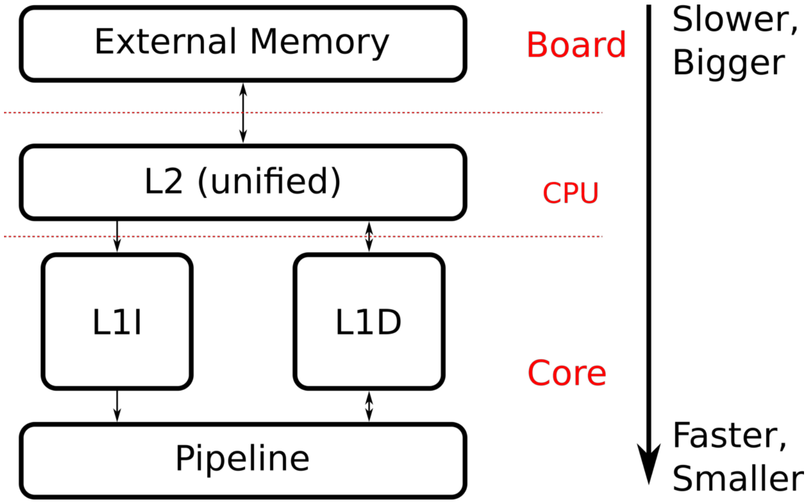
Laser induced fault on smartphone (2017) [VTM+17]

3. Fault Effect Forensic on complex CPU

Fault Effect Forensic on complex CPU

- Fault on complex CPU is possible
- How to analyse a fault effect?
- Fault effect analysis on MPU and L1 instruction cache dysfunction
- This work is a co-joint ANSSI/INRIA [TBE+19]

Reminder on memory hierarchy



Targeted software (single-core)

```
trigger_up();
//wait to compensate bench latency
wait_us(2);
for(i = 0;i<50; i++) {
  for(j = 0;j<50;j++) {
    cnt++;
  }
}
trigger_down();
```

Forensic

Just after a fault, we set the Program Counter to the start of the loop. Then we execute step-by-step and check the side effects.

```
_0x48a04: ldr   w0, [x29,#20]
_0x48a08: add   w0, w0, #0x1
_0x48a0c: str   w0, [x29,#20]
_0x48a10: ldr   w0, [x29,#24]
_0x48a14: add   w0, w0, #0x1
_0x48a18: str   w0, [x29,#24]
_0x48a1c: ldr   w0, [x29,#24]
_0x48a20: cmp   w0, #0x31
_0x48a24: b.le 48a04
```

Forensic

Just after a fault, we set the Program Counter to the start of the loop. Then we execute step-by-step and check the side effects.

```
→ _0x48a04: ldr  w0, [x29,#20]
  _0x48a08: add  w0, w0, #0x1
  _0x48a0c: str  w0, [x29,#20]
  _0x48a10: ldr  w0, [x29,#24]
  _0x48a14: add  w0, w0, #0x1
  _0x48a18: str  w0, [x29,#24]
  _0x48a1c: ldr  w0, [x29,#24]
  _0x48a20: cmp  w0, #0x31
  _0x48a24: b.le 48a04
```

```
pc: 0x48a04
> reg x0
x0 (/64): 0x1
```

JTAG session

Forensic

Just after a fault, we set the Program Counter to the start of the loop. Then we execute step-by-step and check the side effects.

```
  _0x48a04: ldr   w0, [x29,#20]
→ _0x48a08: add   w0, w0, #0x1
  _0x48a0c: str   w0, [x29,#20]
  _0x48a10: ldr   w0, [x29,#24]
  _0x48a14: add   w0, w0, #0x1
  _0x48a18: str   w0, [x29,#24]
  _0x48a1c: ldr   w0, [x29,#24]
  _0x48a20: cmp   w0, #0x31
  _0x48a24: b.le 48a04
```

```
pc: 0x48a04
> reg x0
x0 (/64): 0x1
> step
pc: 0x48a08
```

JTAG session

Forensic

Just after a fault, we set the Program Counter to the start of the loop. Then we execute step-by-step and check the side effects.

```
  _0x48a04: ldr  w0, [x29,#20]
→ _0x48a08: add  w0, w0, #0x1
  _0x48a0c: str  w0, [x29,#20]
  _0x48a10: ldr  w0, [x29,#24]
  _0x48a14: add  w0, w0, #0x1
  _0x48a18: str  w0, [x29,#24]
  _0x48a1c: ldr  w0, [x29,#24]
  _0x48a20: cmp  w0, #0x31
  _0x48a24: b.le 48a04
```

```
pc: 0x48a04
> reg x0
x0 (/64): 0x1
> step
pc: 0x48a08
> reg x0
x0 (/64): 0x2
```

JTAG session

Forensic

Just after a fault, we set the Program Counter to the start of the loop. Then we execute step-by-step and check the side effects.

```
  _0x48a04: ldr   w0, [x29,#20]
  _0x48a08: add   w0, w0, #0x1
→ _0x48a0c: str   w0, [x29,#20]
  _0x48a10: ldr   w0, [x29,#24]
  _0x48a14: add   w0, w0, #0x1
  _0x48a18: str   w0, [x29,#24]
  _0x48a1c: ldr   w0, [x29,#24]
  _0x48a20: cmp   w0, #0x31
  _0x48a24: b.le  48a04
```

```
pc: 0x48a04
> reg x0
x0 (/64): 0x1
> step
pc: 0x48a08
> reg x0
x0 (/64): 0x2
> step
pc: 0x48a0c
```

JTAG session

Forensic

Just after a fault, we set the Program Counter to the start of the loop. Then we execute step-by-step and check the side effects.

```
  _0x48a04: ldr   w0, [x29,#20]
  _0x48a08: add   w0, w0, #0x1
→ _0x48a0c: str   w0, [x29,#20]
  _0x48a10: ldr   w0, [x29,#24]
  _0x48a14: add   w0, w0, #0x1
  _0x48a18: str   w0, [x29,#24]
  _0x48a1c: ldr   w0, [x29,#24]
  _0x48a20: cmp   w0, #0x31
  _0x48a24: b.le  48a04
```

```
pc: 0x48a04
> reg x0
x0 (/64): 0x1
> step
pc: 0x48a08
> reg x0
x0 (/64): 0x2
> step
pc: 0x48a0c
> reg x0
x0 (/64): 0x2
```

JTAG session

Forensic

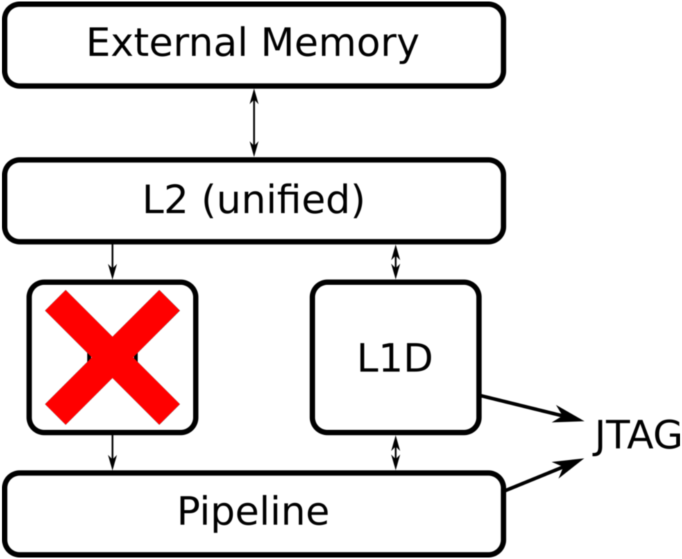
Just after a fault, we set the Program Counter to the start of the loop. Then we execute step-by-step and check the side effects.

```
  _0x48a04: ldr   w0, [x29,#20]
  _0x48a08: add   w0, w0, #0x1
→ _0x48a0c: str   w0, [x29,#20]
  _0x48a10: ldr   w0, [x29,#24]
  _0x48a14: add   w0, w0, #0x1
  _0x48a18: str   w0, [x29,#24]
  _0x48a1c: ldr   w0, [x29,#24]
  _0x48a20: cmp   w0, #0x31
  _0x48a24: b.le 48a04
```

```
pc: 0x48a04
> reg x0
x0 (/64): 0x1
> step
pc: 0x48a08
> reg x0
x0 (/64): 0x2
> step
pc: 0x48a0c
> reg x0
x0 (/64): 0x2
> mdw 0x48a08 1
0x00048a08: add  w0, w0, #0x1
```

JTAG session

Confirming micro-architectural model



Confirming micro-architectural model

How to confirm?

Invalidate L1I cache by executing corresponding instruction.

```
> reg pc 0x6a784
pc (/64): 0x000000000006A784
> step => IC IALLU
pc: 0x6a788
> step => ISB
pc: 0x6a78c
> reg pc 0x48a08
pc (/64): 0x0000000000048A08
> reg x0
x0 (/64): 0x0000000000000002
> step
pc: 0x48a0c
> reg x0
x0 (/64): 0x0000000000000003
```

JTAG session

Failure cause

Hypothesis

- Fault is only on first execution,
- and fault has an impact on L1I.

The fault occurs on a memory transfer when writing instructions to L1I.

Failure cause

Hypothesis

- Fault is only on first execution,
- and fault has an impact on L1I.

The fault occurs on a memory transfer when writing instructions to L1I.

```
trigger_up();
wait_us(2);
/* + */invalidate_icache();
for(i = 0;i<50; i++) {
  for(j = 0;j<50;j++) {
    cnt++;
  }
}
trigger_down();
```

Observations

Now, we can reproduce the previous fault, if we inject during the cache reload (lasts $2\mu s$).

How to improve security of Complex CPU

Several attacks were published without knowledge of the targeted element or the fault model:

- Unable to reproduce attacks.
- Problem to design efficient countermeasure.
- Problem to evaluate sensitive functions.

How to improve security of Complex CPU

Several attacks were published without knowledge of the targeted element or the fault model:

- Unable to reproduce attacks.
- Problem to design efficient countermeasure.
- Problem to evaluate sensitive functions.

Characterisation of fault effect on complex CPU is a work in progress.

- How to characterizing?
- Which approach?

4. Characterizing Fault Model on Complex CPU

State-of-the-art characterizing the fault effect

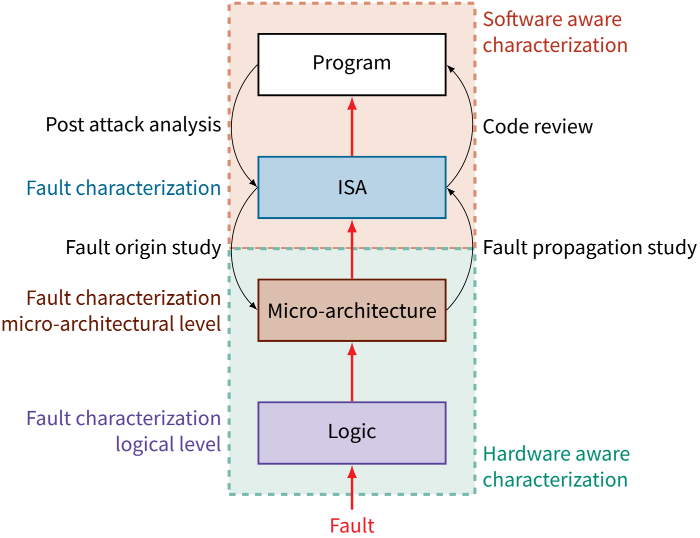
Micro-controller CPU characterisation

- Balasch *et al.* [BGV11] (Clock)
- Moro *et al.* [MDH$^+$13] (EM Perturbation)
- Korak *et al.* [KH14] (Clock & et tension)
- Riviere *et al.* [RNR$^+$15] (Instruction cache)
- Yuce *et al.* [YSW18]

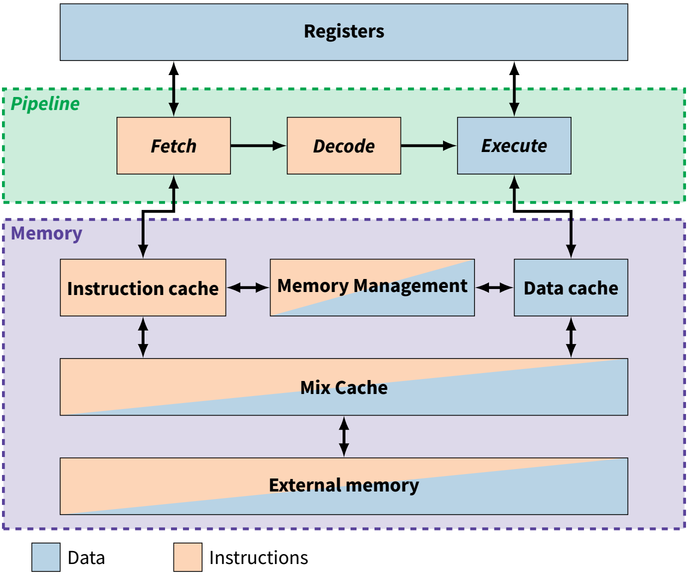
Complex CPU characterisation

- Dumont *et al.* [DLM19] (low level characterisation)
- Proy *et al.* [PHB$^+$19] (EM perturbation to characterize their countermeasures)

Which is the methodology to use?



General Complex CPU architecture



Characterizing the fault model from ISA to Micro-Architectural Block (MAB)

Based on a part of Thomas Trouchkine's thesis, published in [TBC19]

Hypotheses

- Non-changing state instructions are executed
- Instructions manipulate registers only

Data perturbation

$$r_f = f(r)$$

Instruction perturbation

$$r_f = i_f(s)$$

$$i_f = f(i)$$

Data processing test code

Listing 1: ARM semantic `nop` instruction

```
mov r0, r0
  # Several times
mov r0, r0
```

Listing 2: x86 semantic `nop` instruction

```
mov rax, rax
  # Several times
mov rax, rax
```

Memory access test code

Listing 3: ARM read/write in memory instructions

```
str r0, [r1]
ldr r0, [r1]

    # Several times

str r0, [r1]
ldr r0, [r1]
```

Listing 4: x86 read/write in memory instructions

```
mov rax, [rbx]
mov [rbx], rax

    # Several times

mov rax, [rbx]
mov [rbx], rax
```

Corruption effects analysis

Faulted element	Data				
Fault type	Register corruption	Memory corruption		Bad fetch	
Faulted MAB	Registers	Cache	Data bus	Cache	Memory Management

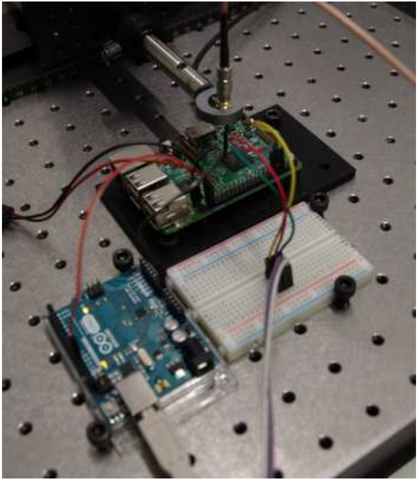
Corruption effects analysis

Faulted element	Data				
Fault type	Register corrup- tion	Memory corruption		Bad fetch	
Faulted MAB	Registers	Cache	Data bus	Cache	Memory Manage- ment

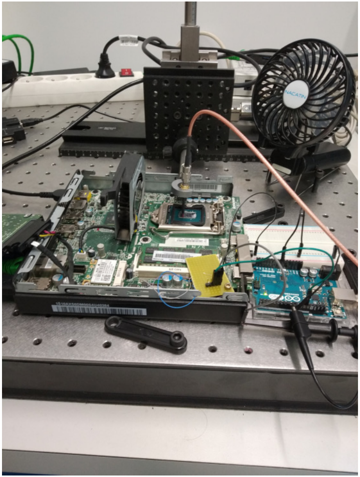
Faulted element	Instruction				
Fault type	Corruption			Bad fetch	
Faulted MAB	Pipeline	Cache	Bus	Cache	Memory Manage- ment

Experiences

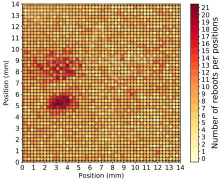
BCM2837 (ARM)



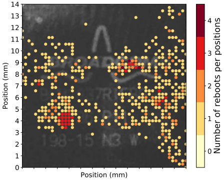
Intel Core i3 (x86)



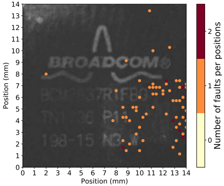
EM sensibility of SoC of Raspberry pi 3 board (BCM2837)



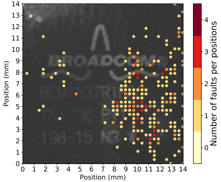
Reboot on bare metal



Reboot on Linux



Faults on code on bare metal

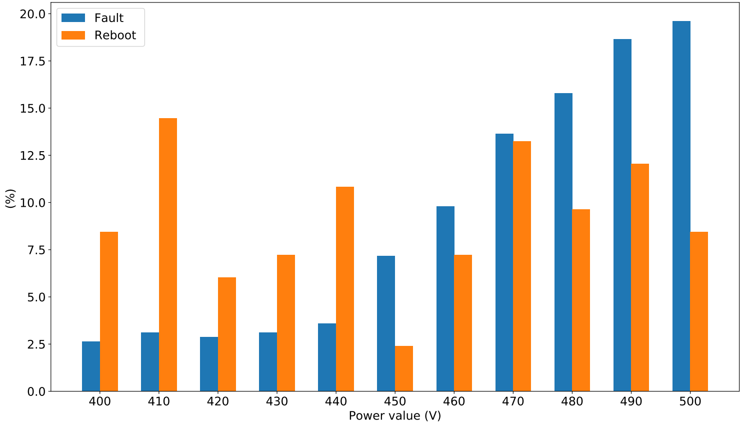


Faults on code on Linux

Bare-metal code was developed by the INRIA-LHS [TBE+19]

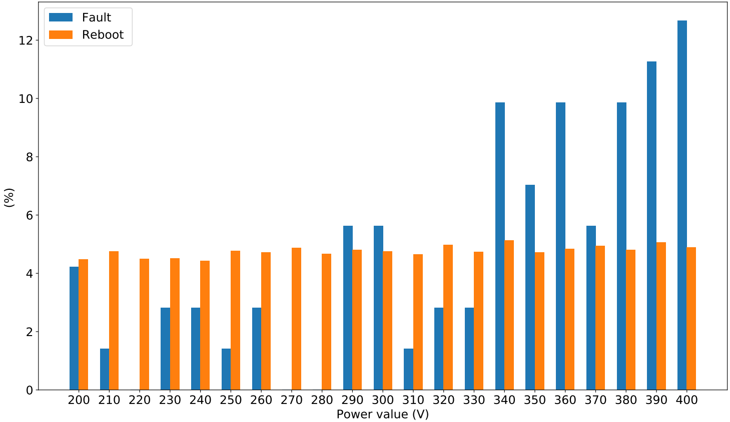
Faults/Reboots depend on EM power

- Probe is placed on “fault” position
- Tested on **Linux**



Faults/Reboots depend on EM power (cont.)

- Probe is placed on “fault” position
- Tested on **bare-metal**

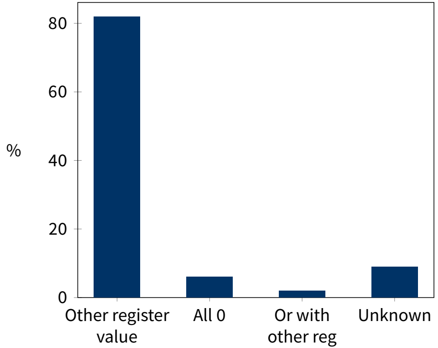


EM sensibility of SoC of Raspberry pi 3 board (BCM2837) (cont.)

`mov r0, r0` test code

`r0 <= r0`

Pattern of the faulted value



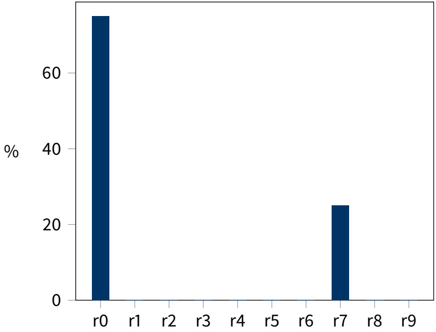
- check on `r0` to `r9`
- the operand doesn’t change (80%)
- `rX <= rY`

Experiments on Raspberry Pi 3 - Results

`mov r0, r0` test code

`r0 <= r0`

Number of faults per register

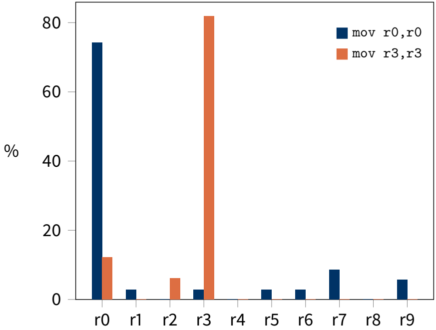


- destination register doesn’t change (75%)
- `r0 <= rX`

Destination analysis

```
mov r0, r0
mov r3, r3
```

Number of faults per register



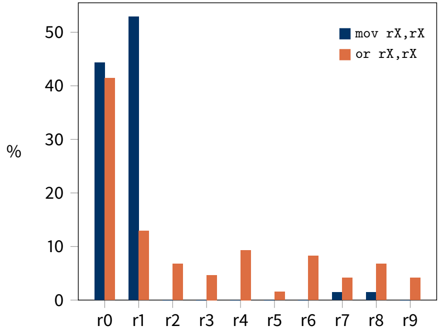
- destination register doesn't change (75%)
- `r0 <= rX`

Operands analysis

```
mov rX, rX
or rX, rX
```

$X \in [0, 9]$

Value in the faulted register

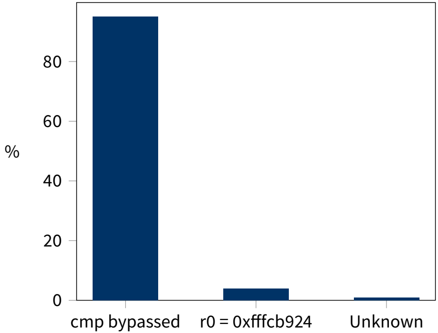


- all registers faulted with same probability
- `rX <= r{0,1}`
- second operand set to 0 or 1

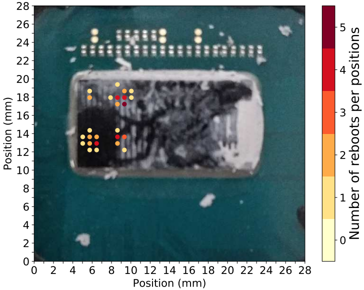
Example of exploitation

Targeting `cmp` instruction

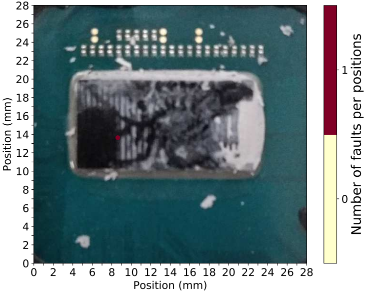
```
init:    r3 <= 0xff

         cmp r3, #255
         bne fault
         b nofault
fault:   mov r9, #170
         b end
nofault: mov r9, #85
end:     nop
```



EM sensibility of Intel i3 CPU



Reboot on Linux



Fault on Linux

We obtained the same fault model as Raspberry pi 3 SoC.

To Conclude

- Secure Components have been designed to be tamper-resistant against hardware and software attacks
  - Their security evaluation is well-know and resistant over the time.
- Complex CPUs are more and more used for security features
  - Several attacks target modern CPU without knowledge of the fault model
  - Works starting to characterizing fault effect on complex CPUs.
    - Require to designed efficient countermeasures
- Recent SoCs embed secure component
  - It is a good way to improve security of sensitive assets
  - How to evaluate their security level?

Questions?

Guillaume Bouffard
<guillaume.bouffard@ssi.gouv.fr>

References

[BGV11] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede, *An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus*, 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011 (Luca Breveglieri, Sylvain Guilley, Israel Koren, David Naccache, and Junko Takahashi, eds.), IEEE Computer Society, 2011, pp. 105–114.

[Bla15] BlackHat, *Xbox 360 glitching on fault attack*, November 2015.

[Car17] Pierre Carru, *Attack trustzone with rowhammer*, eshard, 2017.

[DLM19] Mathieu Dumont, Mathieu Lisart, and Philippe Maurine, *Electromagnetic fault injection : How faults occur*, 2019 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2019, Atlanta, GA, USA, August 24, 2019, 2019, pp. 9–16.

References (cont.)

[KH14] Thomas Korak and Michael Hoefler, *On the effects of clock and power supply tampering on two microcontroller platforms*, 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014, Busan, South Korea, September 23, 2014 (Assia Tria and Dooho Choi, eds.), IEEE Computer Society, 2014, pp. 8–17.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom, *Spectre attacks: Exploiting speculative execution*, 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019, 2019, pp. 1–19.

References (cont.)

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg, *Meltdown: Reading kernel memory from user space*, 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018, 2018, pp. 973–990.

[MDH+13] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz, *Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller*, 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013 (Wieland Fischer and Jörn-Marc Schmidt, eds.), IEEE Computer Society, 2013, pp. 77–88.

References (cont.)

[PHB+19] Julien Proy, Karine Heydemann, Alexandre Berzati, Fabien Majéric, and Albert Cohen, *A first isa-level characterization of EM pulse effects on superscalar microarchitectures: A secure software perspective*, Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019., 2019, pp. 7:1–7:10.

[RNR+15] Lionel Rivière, Zakaria Najm, Pablo Rauzy, Jean-Luc Danger, Julien Bringer, and Laurent Sauvage, *High precision fault injections on the instruction cache of armv7-m architectures*, IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015, IEEE Computer Society, 2015, pp. 62–67.

References (cont.)

[TBC19] Thomas Trouchkine, Guillaume Bouffard, and Jessy Clediere, *Fault injection characterization on modern cpus – from the isa to the micro-architecture*, Information Security Theory and Practice - 13th IFIP WG 11.2 International Conference, WISTP 2019, Paris, France, December 10-11, 2019, 2019.

[TBE+19] Thomas Trouchkine, Sebanjila Kevin Bukasa, Mathieu Escouteloup, Ronan Lashermes, and Guillaume Bouffard, *Electromagnetic fault injection against a system-on-chip, toward new micro-architectural fault models*, CoRR **abs/1910.11566** (2019).

[TSS17] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo, *Clkscrew: Exposing the perils of security-oblivious energy management*, Tech. report, Columbia University, 2017.

References (cont.)

[TSW16] Niek Timmers, Albert Spruyt, and Marc Witteman, *Controlling PC on ARM using fault injection*, 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016, IEEE Computer Society, 2016, pp. 25–35.

[vdVFL+16] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida, *Drammer: Deterministic rowhammer attacks on mobile platforms*, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016 (Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, eds.), ACM, 2016, pp. 1675–1689.

[VTM+17] Aurélien Vasselle, Hugues Thiebeauld, Adèle Morisset, Quentin Maouhoub, and Sebastien Ermeneux, *Laser-induced fault injection on smartphone bypassing the secure boot*.

References (cont.)

[YSW18] Bilgiday Yuce, Patrick Schaumont, and Marc Witteman, *Fault attacks on secure embedded software: Threats, design, and evaluation*, J. Hardware and Systems Security **2** (2018), no. 2, 111–130.