

DE LA RECHERCHE À L'INDUSTRIE



FAULT ATTACKS ON TWO SOFTWARE COUNTERMEASURES

Nicolas Moro^{1,3}, Karine Heydemann³, Amine Dehbaoui², Bruno Robisson¹, Emmanuelle Encrenaz³

¹ **CEA**

Commissariat à l'Énergie Atomique et aux Énergies Alternatives

² **ENSM.SE**

Ecole Nationale Supérieure des Mines de Saint-Etienne

³ **LIP6 - UPMC**

Laboratoire d'Informatique de Paris 6
Sorbonne Universités - Université Pierre et Marie Curie

Amine Dehbaoui is now with Serma Technologies



Experimental evaluation of two software countermeasures against fault attacks

N. Moro, K. Heydemann, A. Dehbaoui, B. Robisson, E. Encrenaz
IEEE HOST Symposium 2014, Arlington, VA, USA

TRUDEVICE 2014 – MAY 29-30, PADERBORN, GERMANY

Concern: Security of embedded programs against fault attacks

- Many **software** countermeasures
- Defined by respect to a **fault model**
- Often based on **redundancy principles**
- Some recent schemes propose to **add this redundancy at assembly level**

```

73 0800770 <fonctionTest>:
74 8000770: b570      push    {r4, r5, r6, lr}
75 8000772: 4604      mov     r4, r0
76 8000774: 460e      mov     r6, r1
77 8000776: 2201      movs   r2, #1
78 8000778: 0211      lsls   r1, r2, #8
79 800077a: 480a      ldr    r0, [pc, #40] ; (80007a4 <fonctionTest+0x34>)
80 800077c: f7ff fdf5 bl     800036a <GPIO_WriteBit>
81 8000780: 2500      movs   r5, #0
82 8000782: e005      b.n    8000790 <fonctionTest+0x20>
83 8000784: 7820      ldrb   r0, [r4, #0]
84 8000786: 5d71      ldrb   r1, [r6, r5]
85 8000788: 4408      add    r0, r1
86 800078a: 0240      sxtb   r0, r0
87 800078c: 7020      strb   r0, [r4, #0]
RR 8AAA78e: 1c6d      adde   r5, r5, #1
  
```



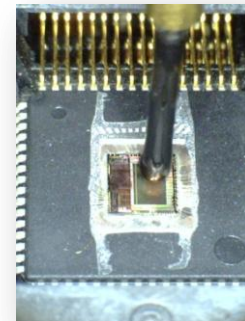
Can we **evaluate the practical effectiveness** of some assembly-level countermeasures against fault attacks ?

- 1 – Provide an experimental evaluation **on single isolated instructions**
- 2 – Provide an experimental evaluation **on complex codes**

- ➔ I. Experimental setup**
- II. Preliminaries about the fault model**
- III. Evaluation on simple codes**
- IV. Evaluation on a FreeRTOS implementation**
- V. Conclusion**

Pulsed electromagnetic fault injection

- **Transient and local** effect of the fault injection
- **Standard circuits not protected** against this technique
- **Solenoid** used as an injection antenna
- Up to **210V** sent on the injection antenna, pulses width longer than **10ns**



Microcontroller based on an ARM Cortex-M3

- 130nm CMOS technology, ARMv7-M architecture
- Frequency 56 MHz, clock period 17.8 ns
- **16/32 bits Thumb-2** RISC instruction set
- Keil ULINKpro JTAG probe to **debug the microcontroller**
- **3-stage pipeline** (Fetch – Decode – Execute), no prefetch

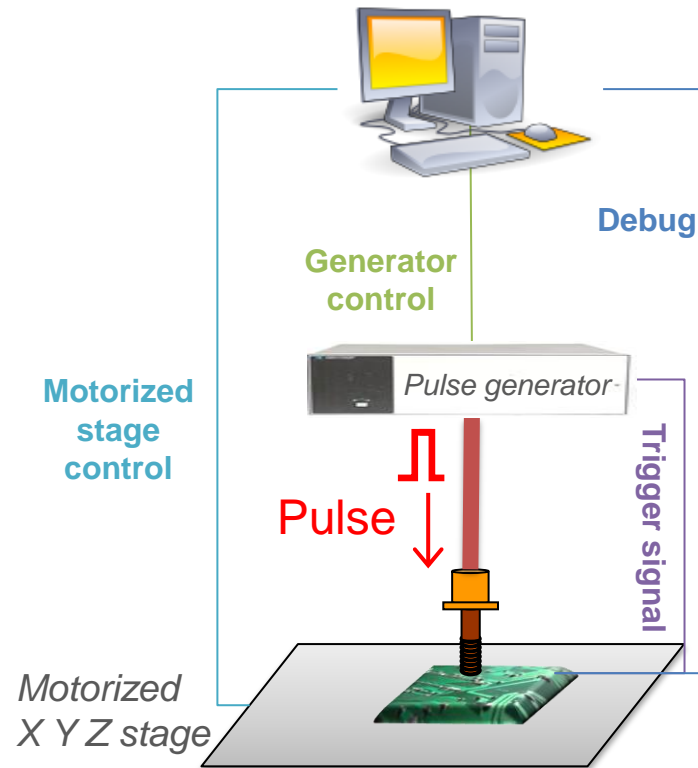


The Definitive Guide to the ARM Cortex-M3 – Joseph Yiu, Newnes, 2009

- The experiment is **driven by the computer**
- The target code is **runned on the microcontroller**
- The pulse generator **sends a voltage pulse**
- The microcontroller is stopped
- The microcontroller's internal data is harvested

Main experimental parameters

- **Position** of the injection antenna (fixed for this work)
- **Electric parameters** of the pulse (fixed for this work)
- **Injection time** of the pulse
- **Executed code** on the microcontroller



Hardware exceptions

UsageFault exceptions for illegal instructions are triggered

→ Used to **identify the impacted instruction** for a given injection time

I. Experimental setup

➔ II. Preliminaries about the fault model

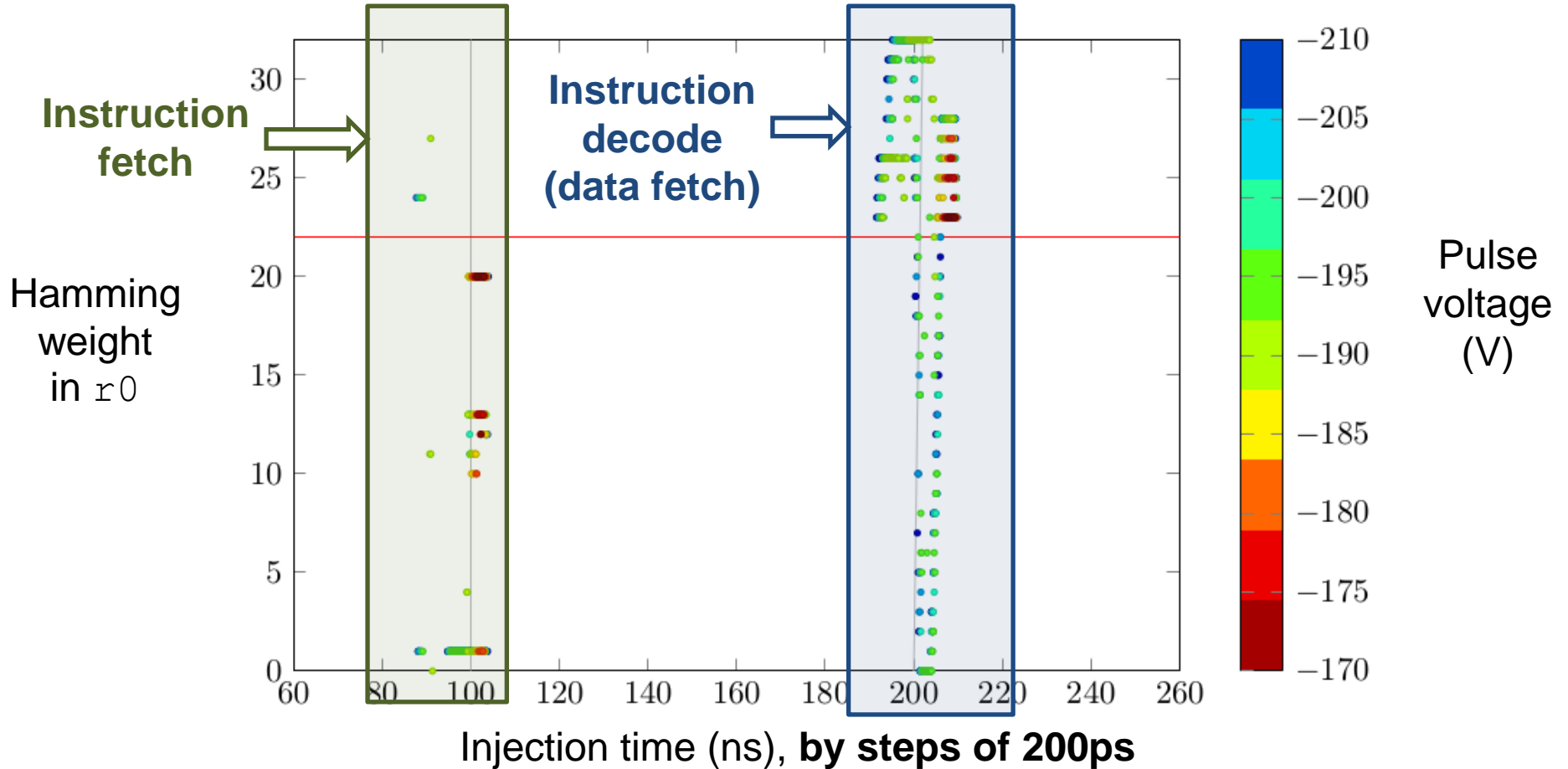
III. Evaluation on simple assembly codes

IV. Evaluation on a FreeRTOS implementation

V. Conclusion

cea FAULT INJECTION ON A SINGLE 16-BIT LDR INSTRUCTION

`ldr r0, [pc, #40]` → loads a 32-bit word into a register from the Flash memory



cea FAULT INJECTION ON A SINGLE 16-BIT LDR INSTRUCTION

`ldr r0, [pc, #40]` → loads a 32-bit word into a register from the Flash memory

Consequences regarding the **instruction flow** (instruction fetch)

- Instructions **replacements**
- Instruction **skips under certain conditions** (~ 20-30% of time)

Consequences regarding the **data flow** (instruction decode)

- Corruption of the `ldr` instructions from the Flash memory



Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller

N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, E. Encrenaz - FDTC Workshop, Santa-Barbara, 2013

I. Experimental setup

II. Preliminaries about the fault model

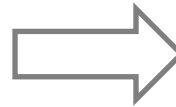
➔ III. Evaluation on simple assembly codes

IV. Evaluation on a FreeRTOS implementation

V. Conclusion

- Two fault injection attempts, every 200 ps
- During a time interval defined by hardware instructions

```
ldr    r0, [pc, #34]
```



```
ldr    r0, [pc, #40]
ldr    r1, [pc, #38]
cmp    r0, r1
bne    <error>
```



150 ns

1500 fault injection attempts
180 faulty outputs



300 ns

3000 fault injection attempts
210 faulty outputs / 50 faulty o.

Relevant metric to evaluate the countermeasures ?

Replacement sequences add some instructions → longer execution time
→ more fault injections to do → **different number** of results to compare

From a security point of view, **effectiveness = reduction of faulty outputs**

- Fault tolerance against one **instruction skip**
- Formally verified using model-checking tools
- A **replacement sequence** for every instruction
- No protection for the data flow
- Experiment performed on the `b1` instruction
- In the tested code, the subroutine modifies `r0`

`b1` `<function>`



```

adr    r1, <return_label>
adr    r1, <return_label>
add    lr, r1, #1
add    lr, r1, #1
b      <function>
b      <function>

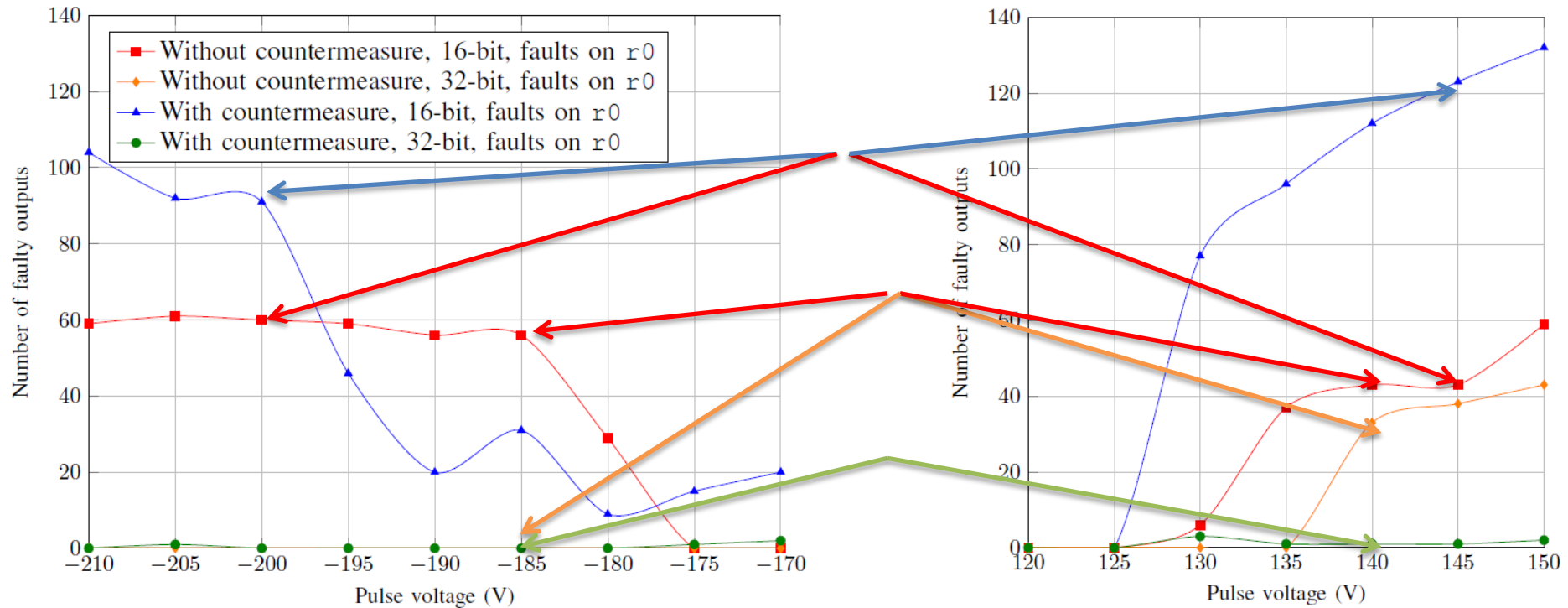
```

`return_label`

Formal verification of a software countermeasure against instruction skip fault attacks

N. Moro, K. Heydemann, E. Encrenaz, B. Robisson - Journal of Cryptographic Engineering, Springer, 2014

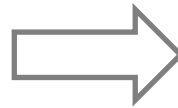
cea FAULT INJECTION RESULTS



- Fewer faults by **forcing the 32-bit encoding of instructions** (orange curve)
- The countermeasure is **not effective with 16-bit instructions** (blue curve)
- The combination **32-bit inst + countermeasure** is very effective (green curve)

- Detects **any single fault** (instruction skips, replacements, data flow)
- Proposed for a **restricted set of instructions** (ALU, load-store)
- Tested for a `ldr` instruction from the Flash memory

```
ldr    r0, [pc, #34]
```



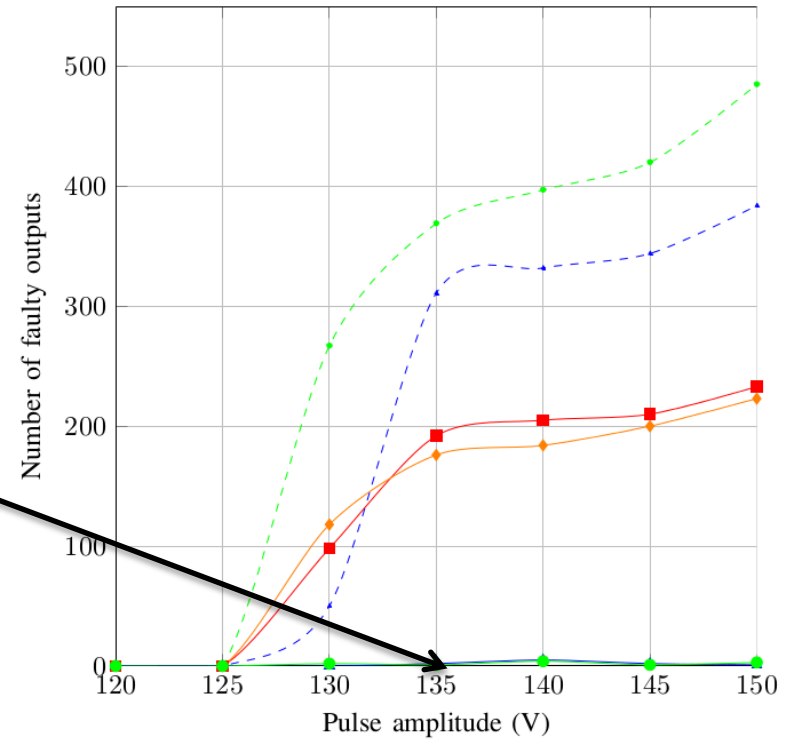
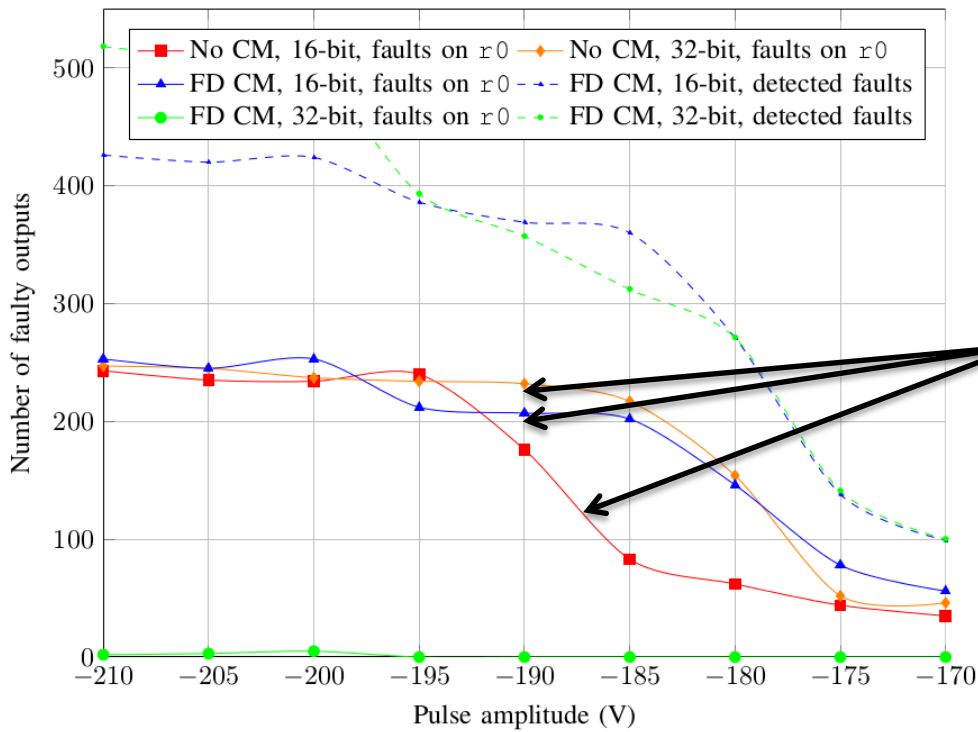
```
ldr    r0, [pc, #40]
ldr    r1, [pc, #38]
cmp    r0, r1
bne    <error>
```



Countermeasures against fault attacks on software implemented AES

A. Barengi, L. Breveglieri, I.Koren, G. Pelosi, F. Regazzoni – WESS Workshop, New-York, 2010

cea FAULT DETECTION COUNTERMEASURE



- Faults for 16-bit and 32-bit encodings, some due to the corruption of the data transfer
- The FD countermeasure is **not effective with a 16-bit encoding** (blue curve)
- However, **countermeasure + 32-bit encoding** → very effective (green curve)

I. Experimental setup

II. Preliminaries about the fault model

III. Evaluation on simple assembly codes

➔ IV. Evaluation on a FreeRTOS implementation

V. Conclusion

- Portable RTOS written in C, **multitasking** operating system

Fault **tolerance** countermeasure

- At the OS initialization
- The systems starts in privileged mode
- Then it switches to unprivileged mode

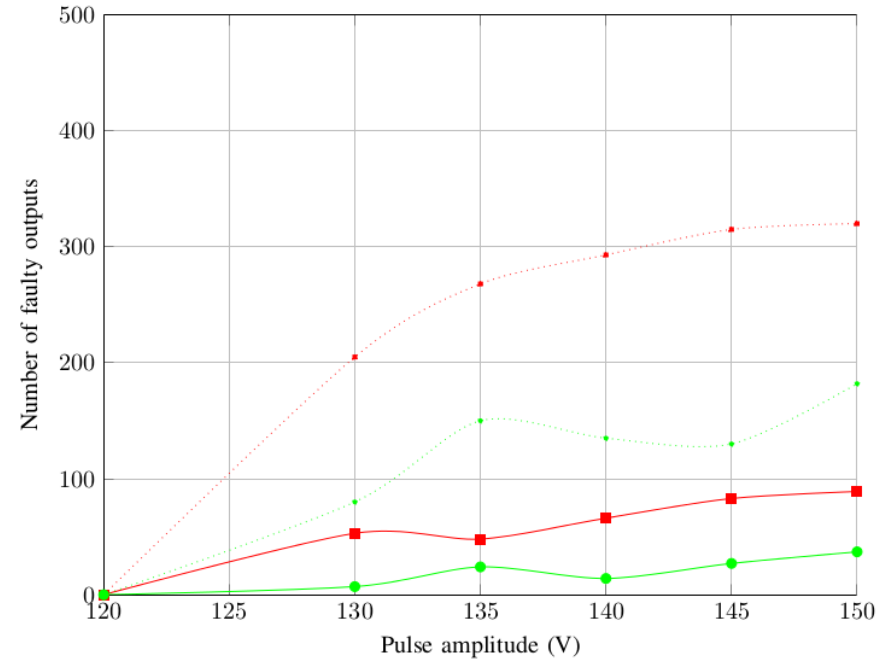
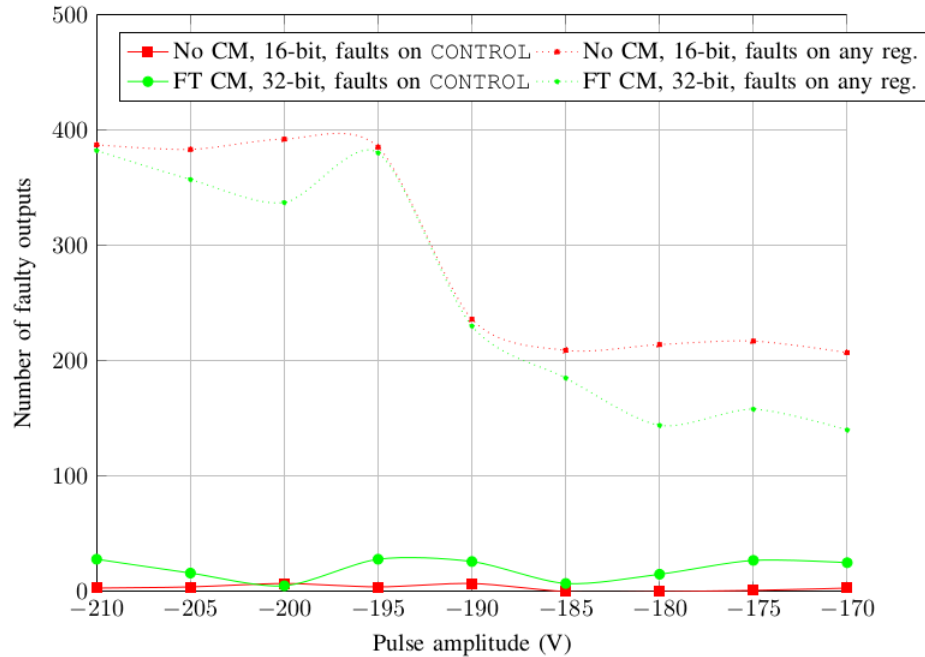
```
msr    control, r3 → Changes priv. mode
msr    psp, r0
mov    r0, #0
add    lr, r1, #1
msr    basepri, r0
ldr    lr, =0xfffffdd
```

prvRestoreContextOfFirstTask function

➔ An attacker may try **to stay in privileged mode**

To evaluate the effectiveness,

we observe the **number of faults in the control register**



- **Not very good effectiveness** for the fault tolerance countermeasure on this code
- The protected `msr` instruction is maybe too specific or the fault model too simplistic
- **Further experiments are required** to deeply analyze the effectiveness of this CM

Fault detection countermeasure

- During task creation
- Each task has its own priority level
- The priority level is loaded from the Flash

```
ldr    r0, [r0, #0] → uxPriority in r0
str    r0, [sp, #0]
movs   r3, #0
movs   r2, #128
movs   r1, #0
ldr    r0, =address_fct
bl     <xTaskGenericCreate>
```

} Arguments for the function

Code before calling `xTaskGenericCreate`

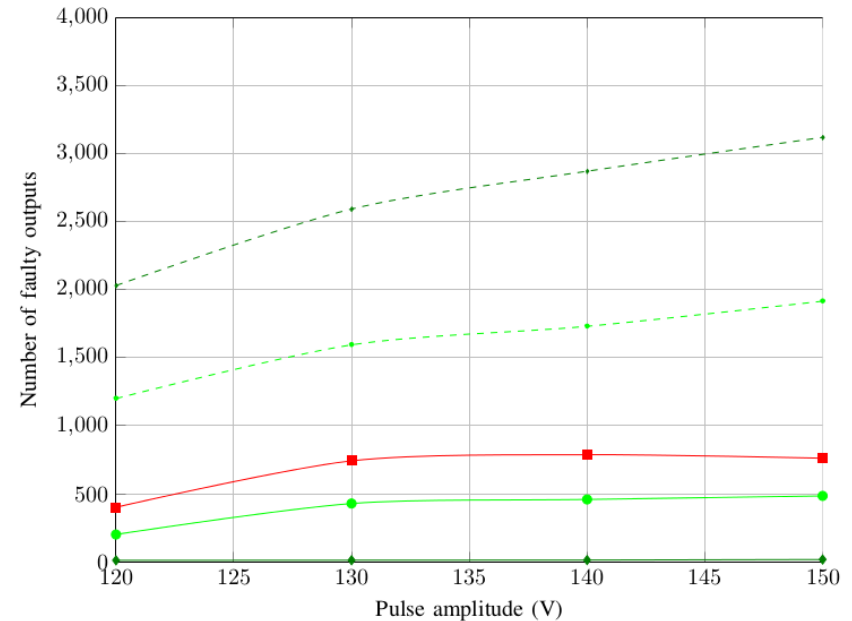
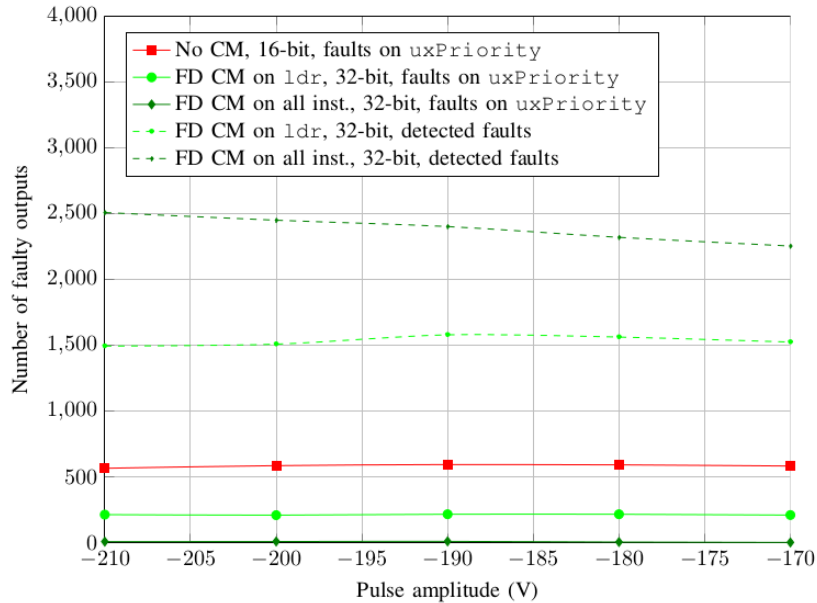
→ An attacker may try to **change a priority level**

To evaluate the effectiveness,

we observe the **number of faults in this priority level**

(in the `xTaskGenericCreate` function)

cea FAULT DETECTION COUNTERMEASURE



- The countermeasure when only applied to `ldr` instructions still misses some faults
- The countermeasure is **very effective on this code** when applied to every instruction
- However, not all the instructions can be protected with this countermeasure
- This countermeasure **must be combined with other techniques against faults**

- I. Experimental setup
- II. Preliminaries about the fault model
- III. Evaluation on simple assembly codes
- IV. Evaluation on a FreeRTOS implementation
- ➔ V. Conclusion

- The effectiveness of both CM **can be nullified** if not well implemented
On this platform, we need to check that the 32-bit encoding of instructions is used
- The fault tolerance CM **can significantly reinforce** an isolated bl instruction
- However, it was **not very effective on the FreeRTOS tested code**
The instruction skip fault model may be too simplistic
- The fault detection CM was **very effective on all the tested codes**
But its applicability is limited since it cannot be applied to several instructions

Perspectives

- Further experiments are required for the **fault tolerance countermeasure**
- Can we **combine** those countermeasures to secure an assembly code ?
- What about **side-channel leakages** on cryptographic implementations ?

Any questions ?

To download the presentation (PDF file)




Nicolas MORO

PhD student, CEA

Graduation expected in Sep. 2014

 www.nicolasmoro.net

 [nicolas.moro \[at\] gmail.com](mailto:nicolas.moro@gmail.com)

 +33.(0)4.42.61.67.13