CEA - DRT/DPACA
Secure Architectures and Systems laboratory

Brice Colombier
brice.colombier@cea.fr

Pierre-Alain Moëllic
pierre-alain.moellic@cea.fr

Experimental evaluation of software countermeasures

PROSECCO Workshop

6 novembre 2018

Goal of the presentation

Goal: Present the first results of the security evaluation we perform at the Secure Architectures and Systems laboratory (joint team CEA Tech, Mines Saint-Etienne).

This evaluation helps to design **efficient countermeasures** by prodiving a **feedback to the designer**.

Evaluation carried out for different:

- Physical threats:
  - Side-channel analysis
  - Fault-attacks
- Hardware targets:
  - 8-bit microcontrollers
  - 32-bit microcontroller ARM Cortex M/A
- Practical use-cases:
  - VerifyPIN
  - AES encryption

Evaluation method

Two main axes:

- **Leakage** assessment using statistical tools
  - Attack-independent
- **Attack**-based methodology:

Complexity / Cost	Side-channel attacks	Fault attacks
+ / $	Correlation power analysis Template attacks	Clock glitches
+++ / $$$	Machine learning (deep neural networks)	Laser

Outline

1. **Side-channel leakage assessment**
2. **Fault attacks**
   - VerifyPIN
   - AES-128 encryption
3. **Combination of protections**
4. **Conclusion**

Side-channel leakage assessment

Leakage assessment

Aim: conduct a statistical study to evaluate the leakages.

Statistical tests: reject or not a *null hypothesis* (i.e. the means of the target populations are equal)

Two common tools in SCA context:

- ***t*-test** [1]: split the traces in two sets w.r.t an intermediate value, see if they differ statistically.
  - The t statistic follows a Student law. For sufficient number of traces, $|t| > 4.5$ give a confidence of 99.999 % to reject the NH.
  - In our experiments: target at **bit level**.
- ***F*-test** [2], **SNR**: generalization of t-test for multiple sets. Takes the variance into consideration.
  - Ratio of inter-class VS intra-class variance.
  - In our experiments: target at **byte level**.

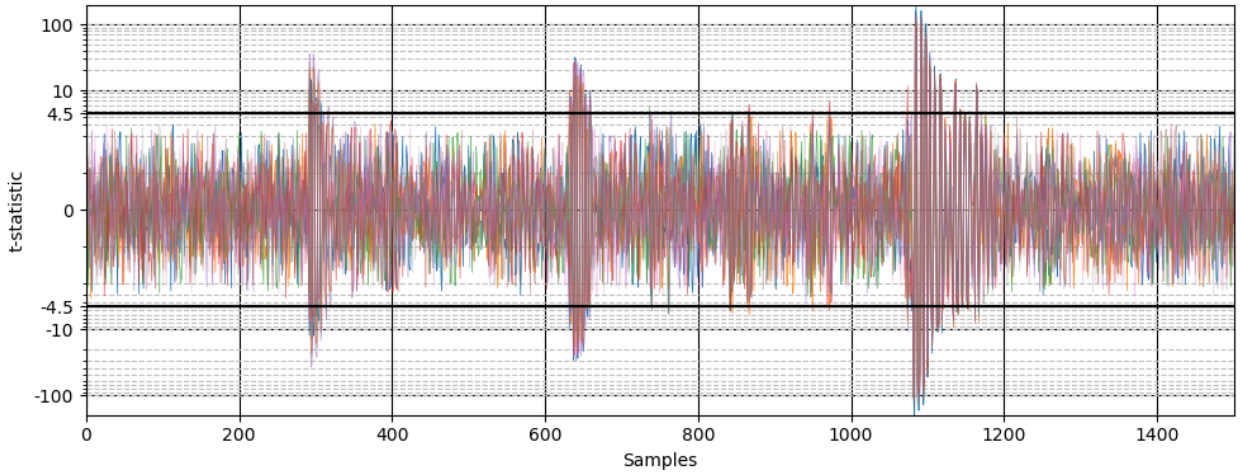
---

[1] Tobias Schneider and Amir Moradi. "Leakage Assessment Methodology - a clear roadmap for sidechannel evaluations". IACR ePrint 2015.

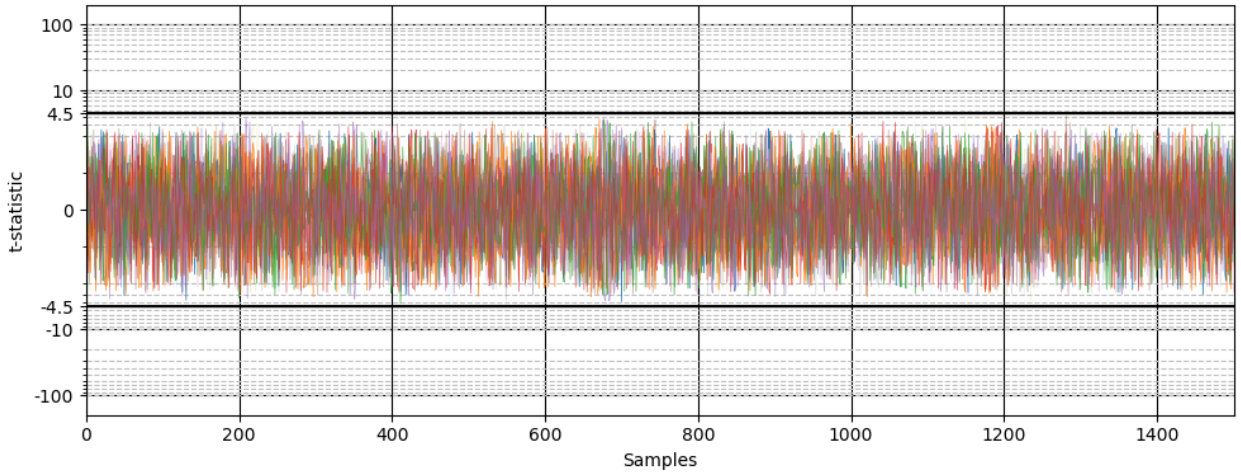
[2] Omar Choudary and Markus G. Kuhn. "Efficient template attacks." International Conference on Smart Card Research and Advanced Applications. 2013.

Comparison of unmasked and masked S-boxes

Splitting according to the value of the 8 bits at the 1st S-box output.
20000 traces of 128-bit AES encryption.



unmasked

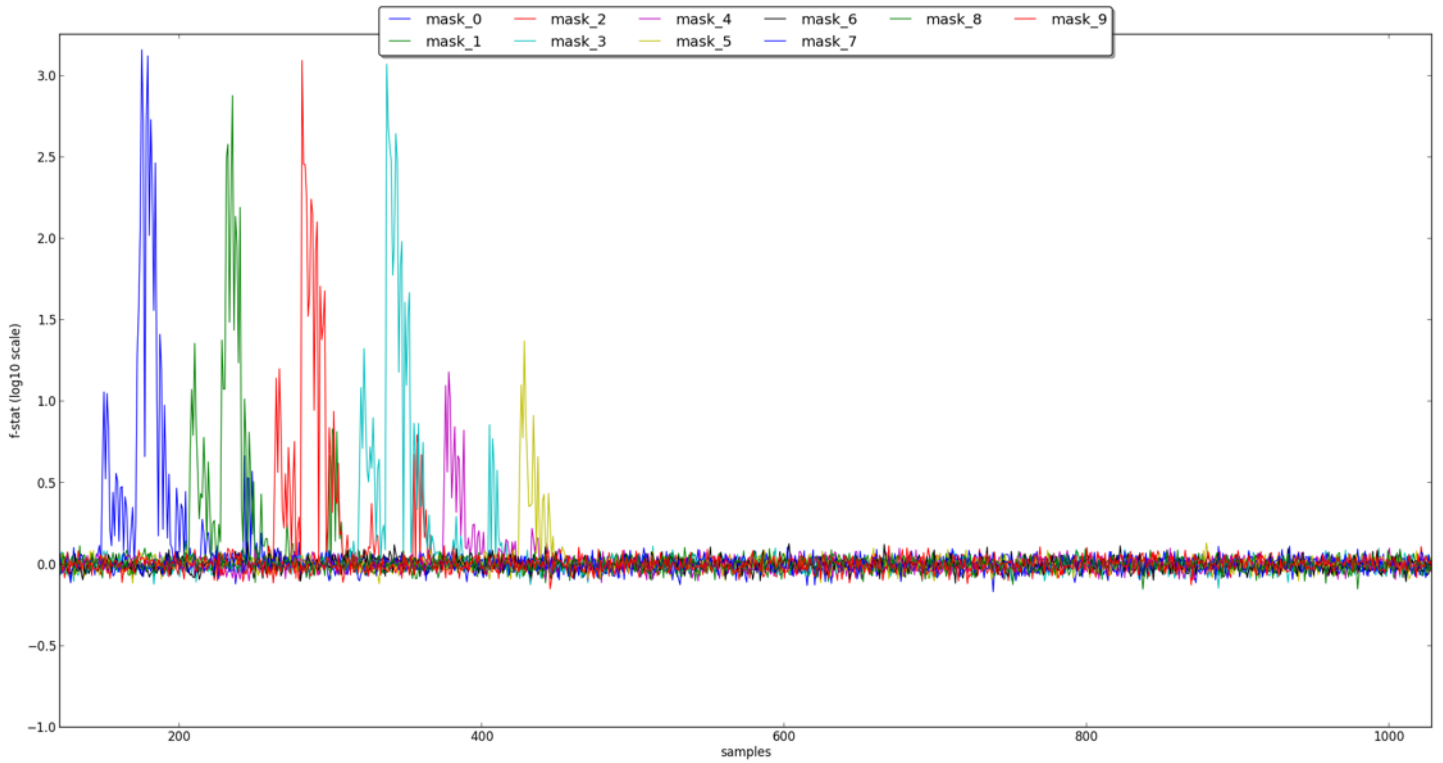


masked

➔ No more 1st order leakage with this masking scheme.

Identification of new leakage points

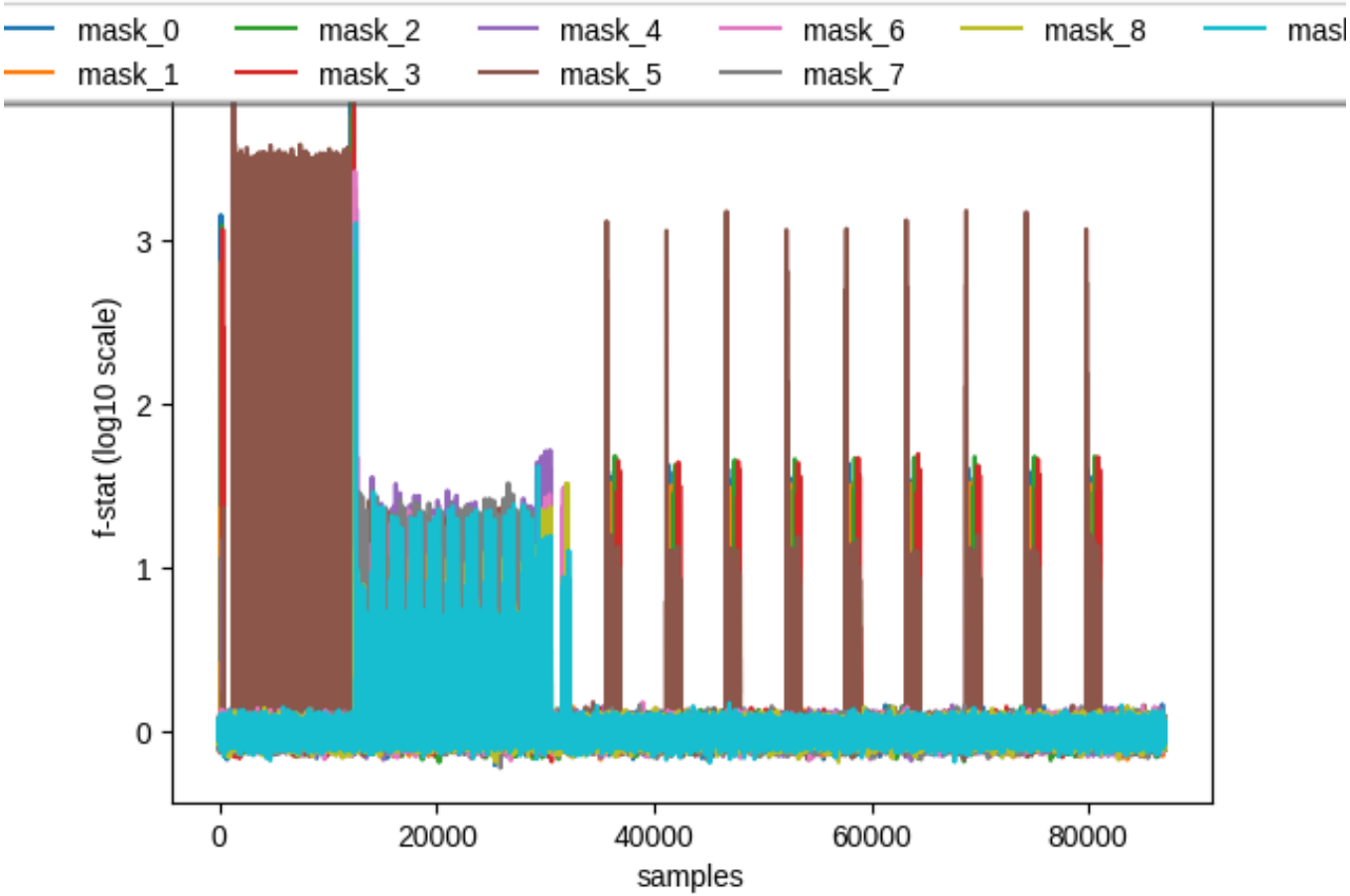
The **masks generation process** leaks information as well (*F*-test).
Generation of the 6 random masks (4 for MixColumn, 2 for SubBytes):



In the **worst case scenario** (profiled attacks), these can be **combined** with other leakage points later to perform a **second order attack**.
(M ; SBOX(P⊕K)⊕M) ➔ SBOX(P⊕K)

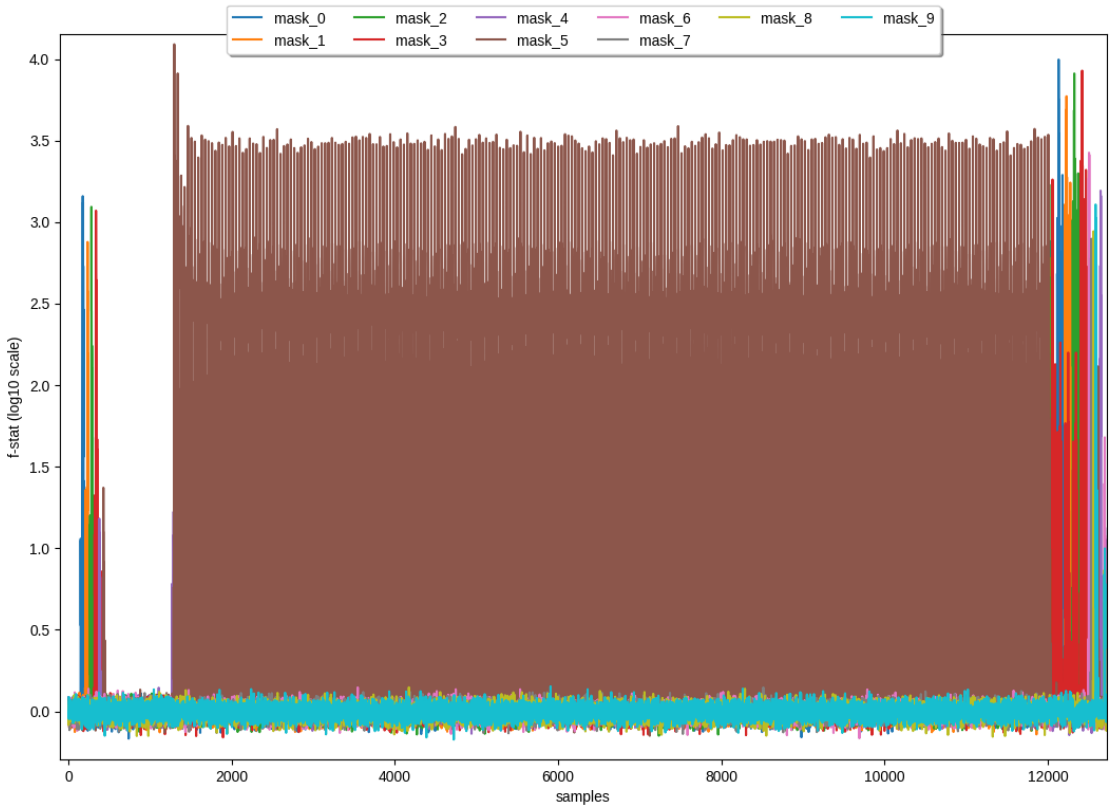
Identification of new leakage points

Interestingly, we can **see the masks manipulation** during the encryption process. The initial (masked) key schedule can also leak information or be profiled for efficient differential fault attack (DFA):



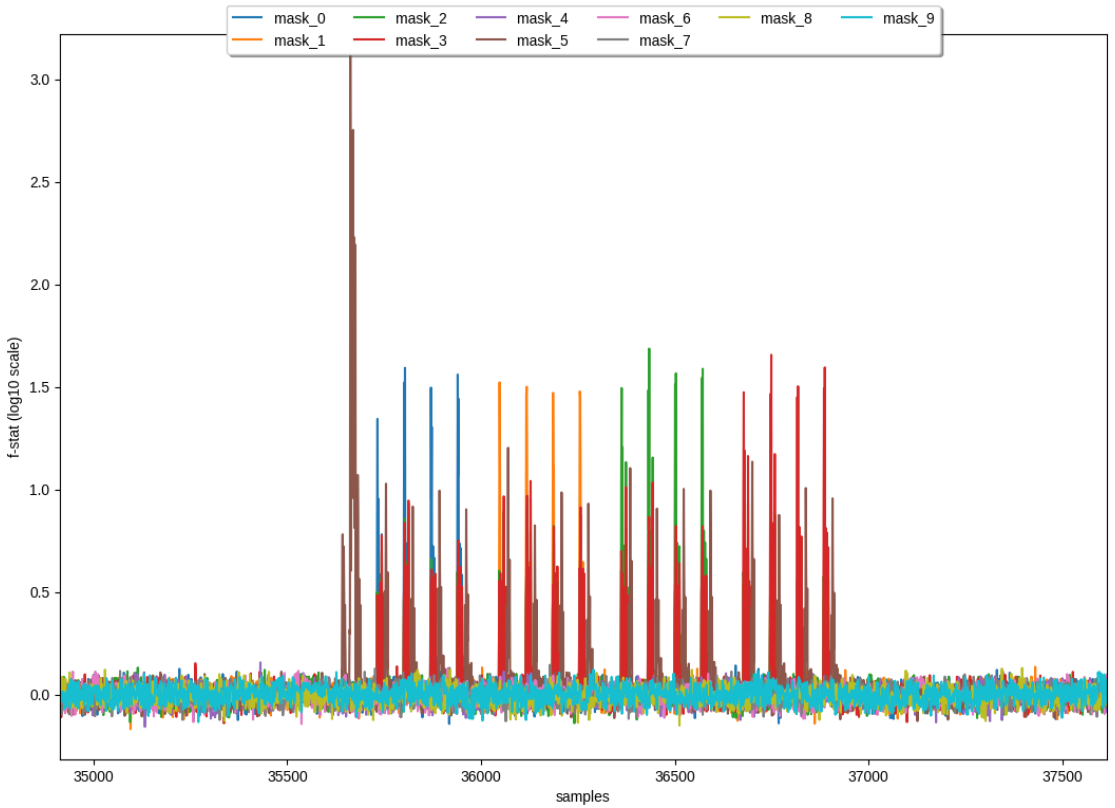
Identification of new leakage points

Interestingly, we can **see the masks manipulation** during the encryption process. The initial (masked) key schedule can also leak information or be profiled for efficient differential fault attack (DFA):



Identification of new leakage points

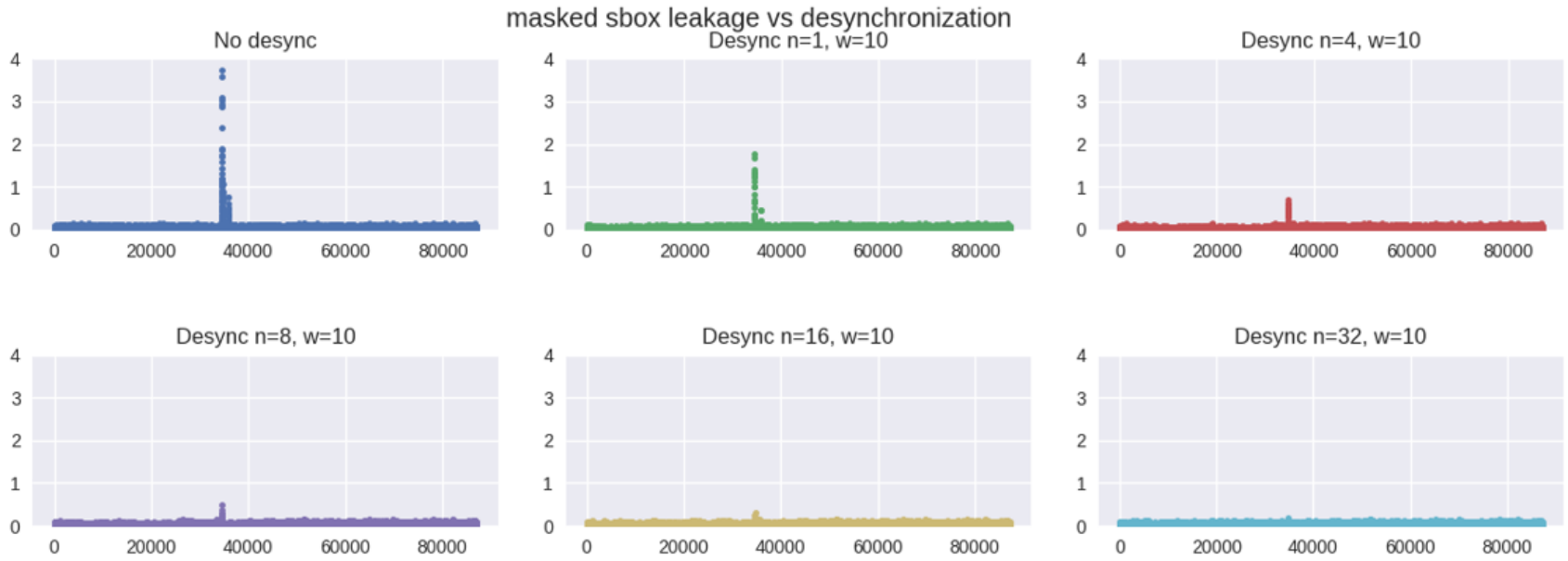
Interestingly, we can **see the masks manipulation** during the encryption process. The initial (masked) key schedule can also leak information or be profiled for efficient differential fault attack (DFA):



F-test on desynchronised traces

A second order CPA can target – jointly – the **two shares**.
Desynchronization-based protections can **reduce this exploitability**.

Leakage evaluation when **simulating desynchronisation** by randomly inserting *n* blocks of *w* NOPs during the execution:



➔ Leakage shrinks and becomes **unexploitable** (20000 traces here).

➔ Provide hints for protecting the design.

Ongoing works

On protected AES (masking, hiding), powerful **template** attacks need:

- Strong **information compression** (PCA, LDA) or
- Detection of **points of interest**
- **Resynchronization** techniques

➔ can become rapidly difficult in practice.

Machine Learning-based analysis can be helpful here [3] [4]

- **Deep learning**-based attacks against **masking**
- **Denoising** and **resynchronization** with **autoencoder**
- …

[3] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, François-Xavier Standaert
Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis). COSADE 2015

[4] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, Cécile Dumas
Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database. IACR ePrint 2018

Fault attacks

Low cost: Clock glitches on a VerifyPIN

Different **hardened** VerifyPIN have been successfully bypassed:

- ✔ Constant-time
- ✔ Constant-time and inlined functions
- ✔ Constant-time and inlined functions and loop counter
- ✖ Constant-time and inlined functions and double call

Limitations

The ChipWhisperer platform **cannot** glitch at two different times.

Plan to overcome

We shall shoot with the **laser**!

Laser faults

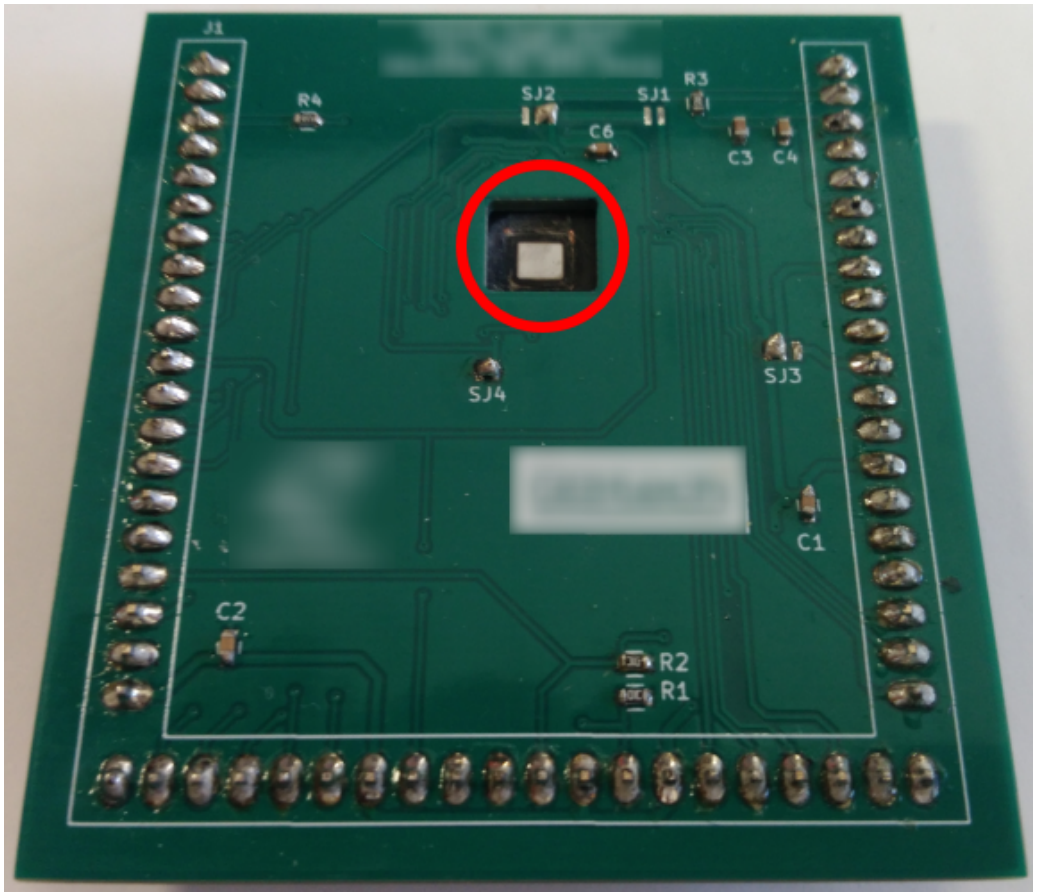
Preparatory work

- ✔ Design a **custom** ChipWhisperer target board:
  - ✔ Front-side access
  - ✔ Back-side access
- ✔ Prepare the target: **decapsulate** the chip to access the die
- ✔ **Mechanical setup** of the target on the bench
- … Mapping out the faults:
  - x-y position,
  - power,
  - duration,
  - delay,
  - type of fault (skip, set, reset, flip, …)

Laser setup

Characteristics

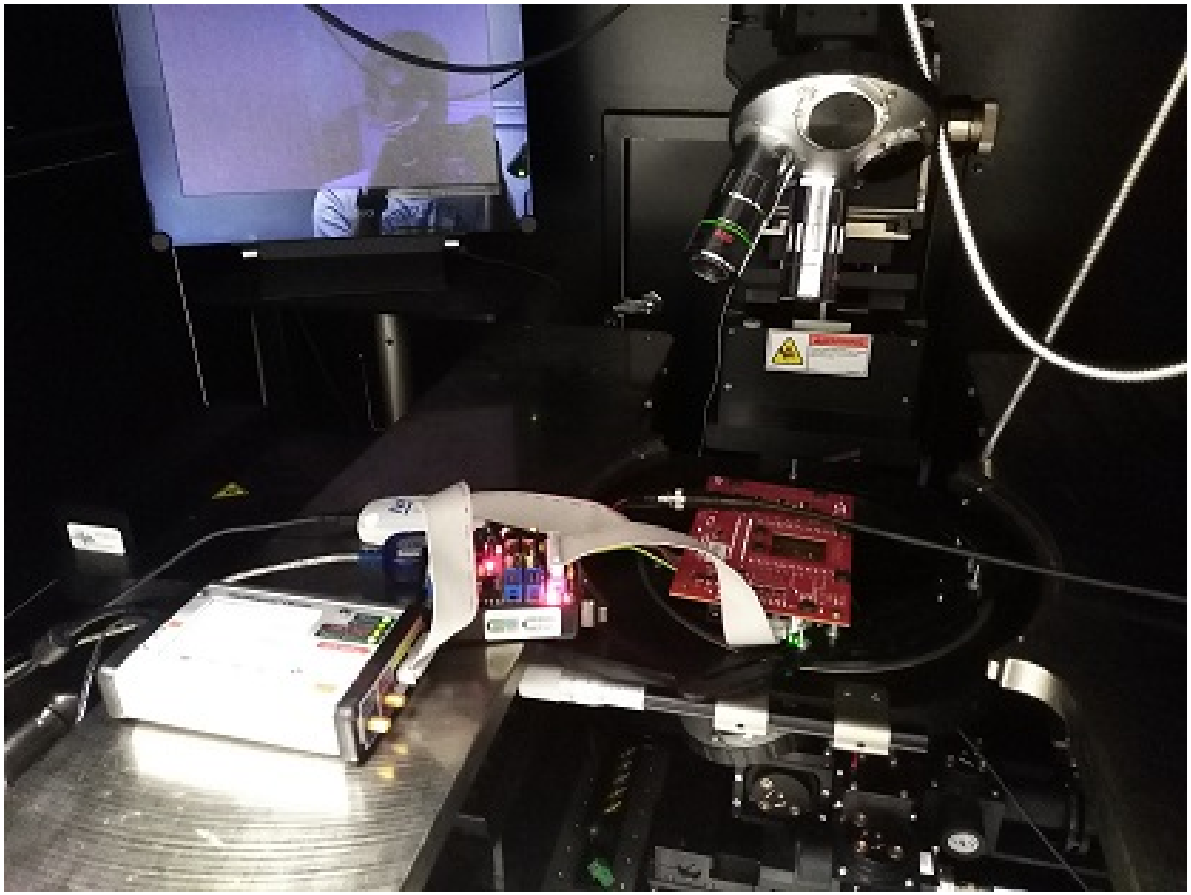
- IR (1064nm)
- >30ps
- 0-3W
- 3 objective lenses:
  - x5 (20µm)
  - x20 (5µm)
  - x100 (1µm)



Laser setup

Characteristics

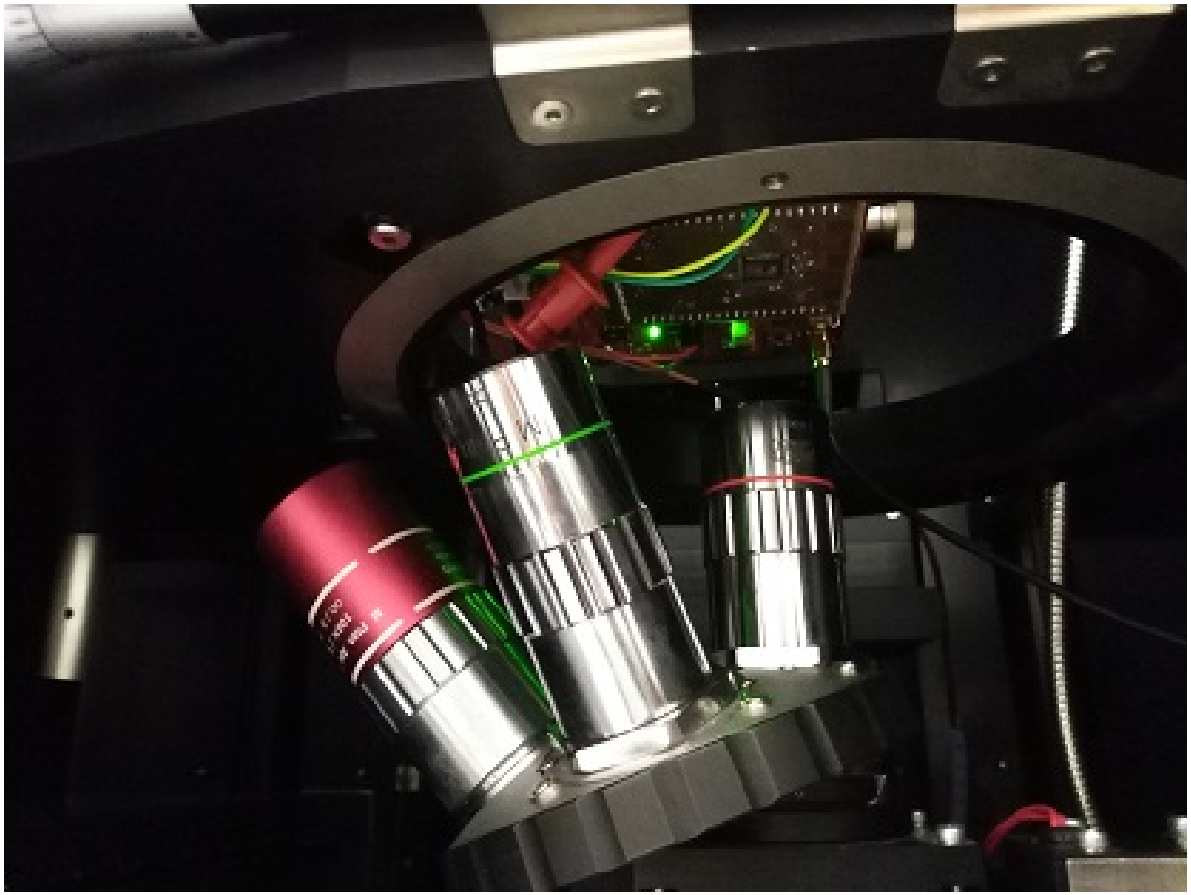
- IR (1064nm)
- >30ps
- 0-3W
- 3 objective lenses:
  - x5 (20µm)
  - x20 (5µm)
  - x100 (1µm)



Laser setup

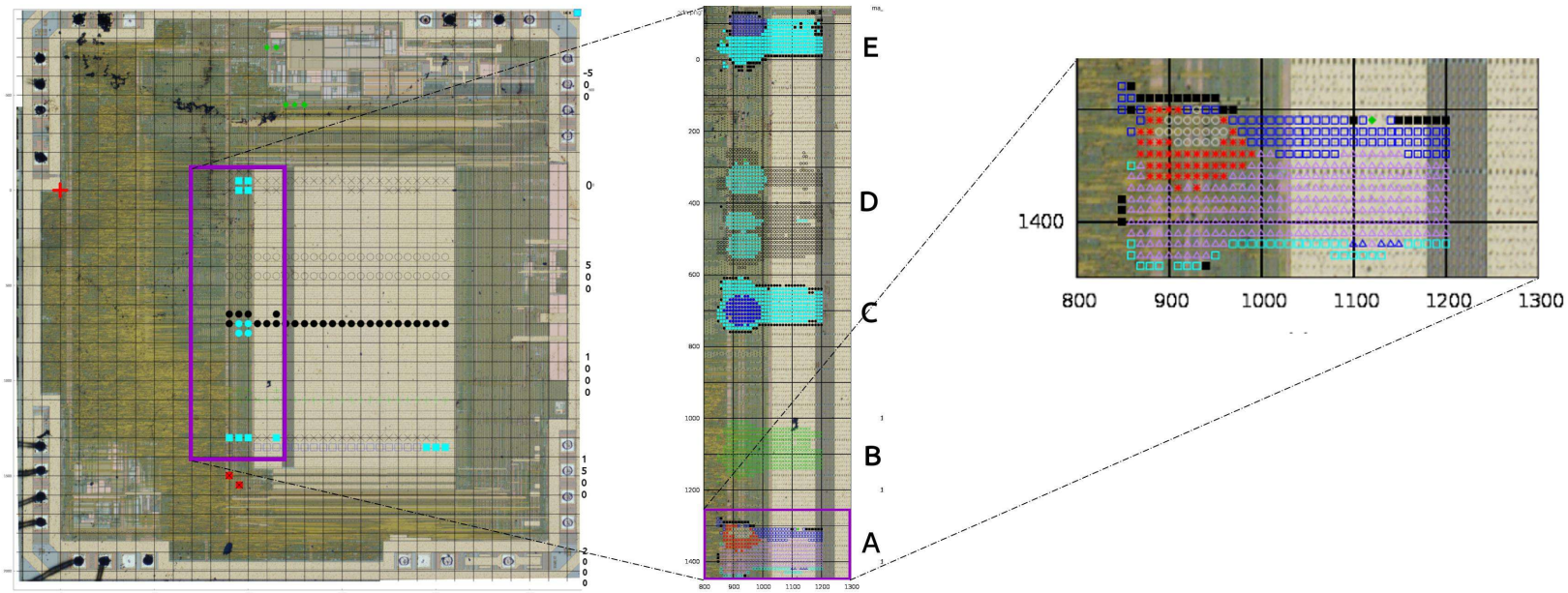
Characteristics

- IR (1064nm)
- >30ps
- 0-3W
- 3 objective lenses:
  - x5 (20µm)
  - x20 (5µm)
  - x100 (1µm)



8-bit microcontroller results

Instruction skip fault model previously validated experimentally [5]



[5] Practical results on laser-induced instruction-skip attacks into microcontrollers.
T. Riom, J.-M. Dutertre, O. Potin, J.-B. Rigaud, TRUDEVICE Workshop 2016, Barcelona

8-bit microcontroller results

This time, **all implementations are vulnerable**.

- ✔ Constant-time
- ✔ Constant-time and inlined functions
- ✔ Constant-time and inlined functions and loop counter
- ✔ Constant-time and inlined functions and double call
- ✔ Constant-time and inlined functions and control-flow integrity

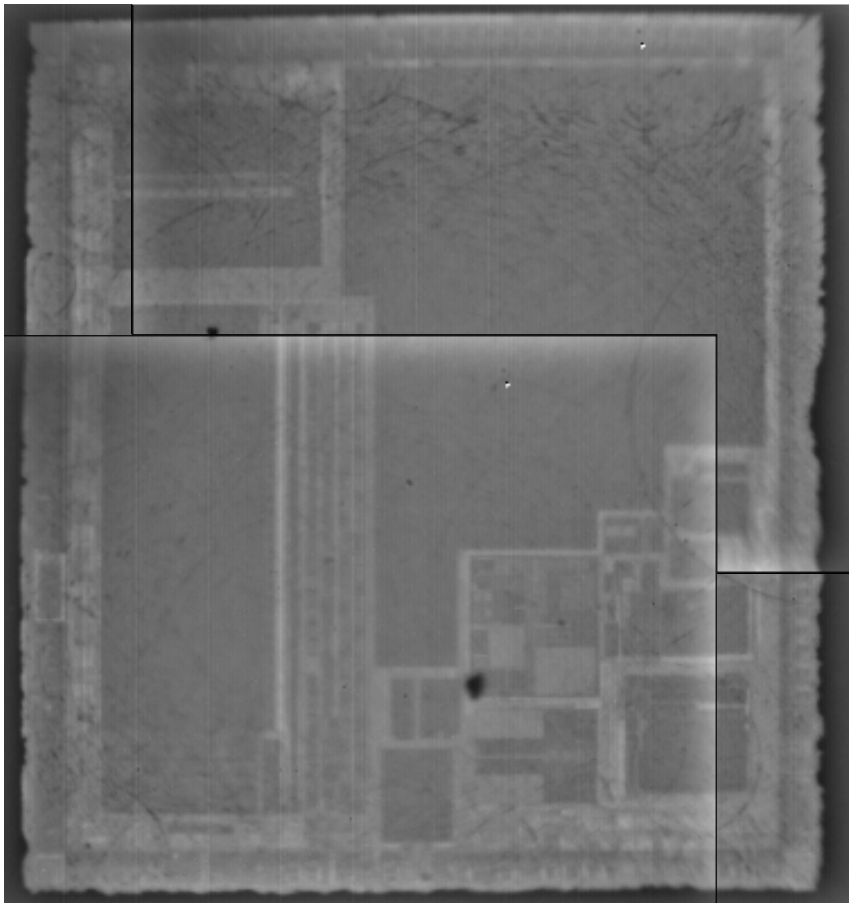
Paradox

Constant-time implementation makes laser attacks much **easier**

32-bit microcontroller

A **more complex** target (32 bits) implies:

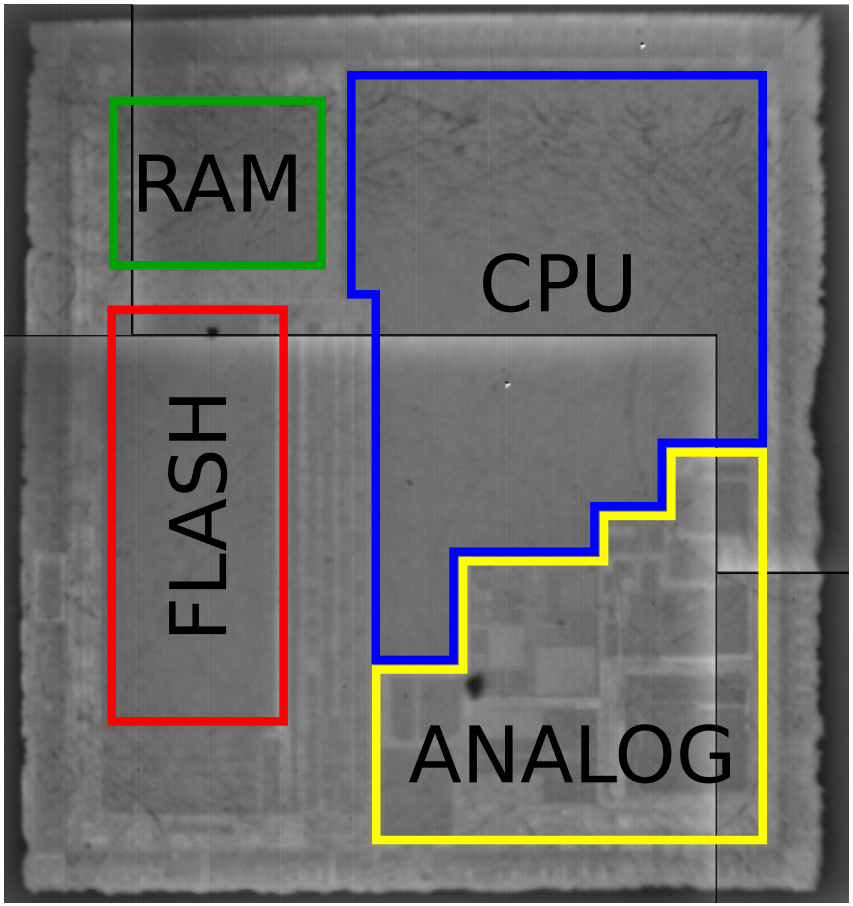
- **Larger area** to cover for cartography (2.5x2.5cm),
- Complex micro-architecture,
- More time variability



32-bit microcontroller

A **more complex** target (32 bits) implies:

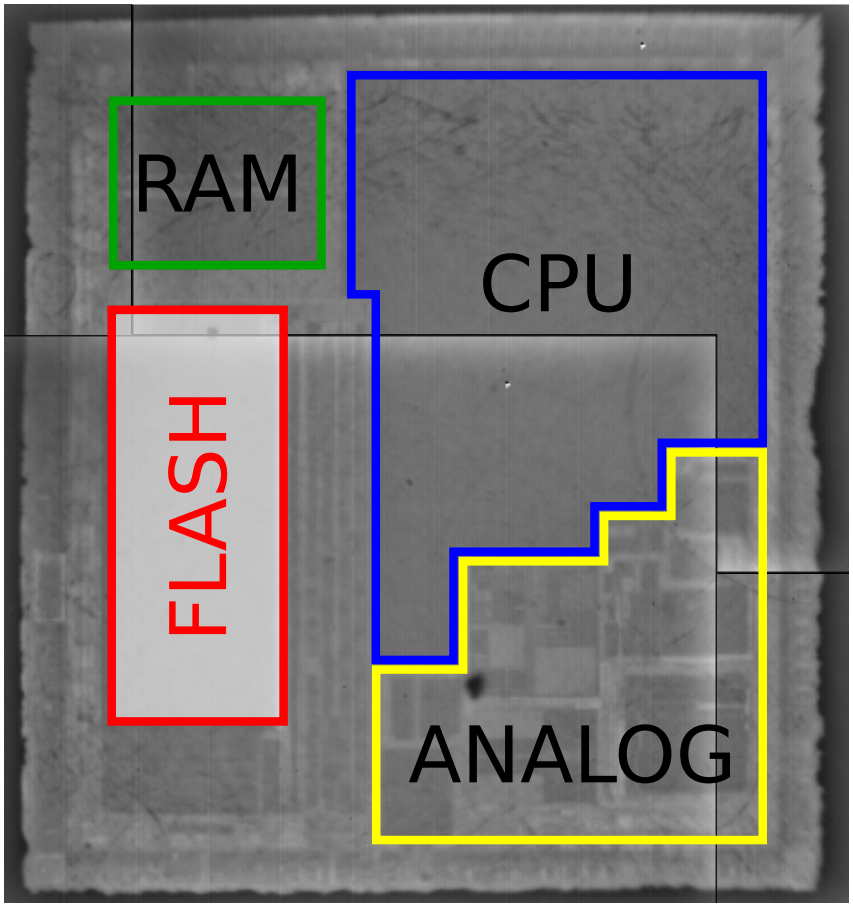
- **Larger area** to cover for cartography (2.5x2.5cm),
- Complex micro-architecture,
- More time variability



32-bit microcontroller

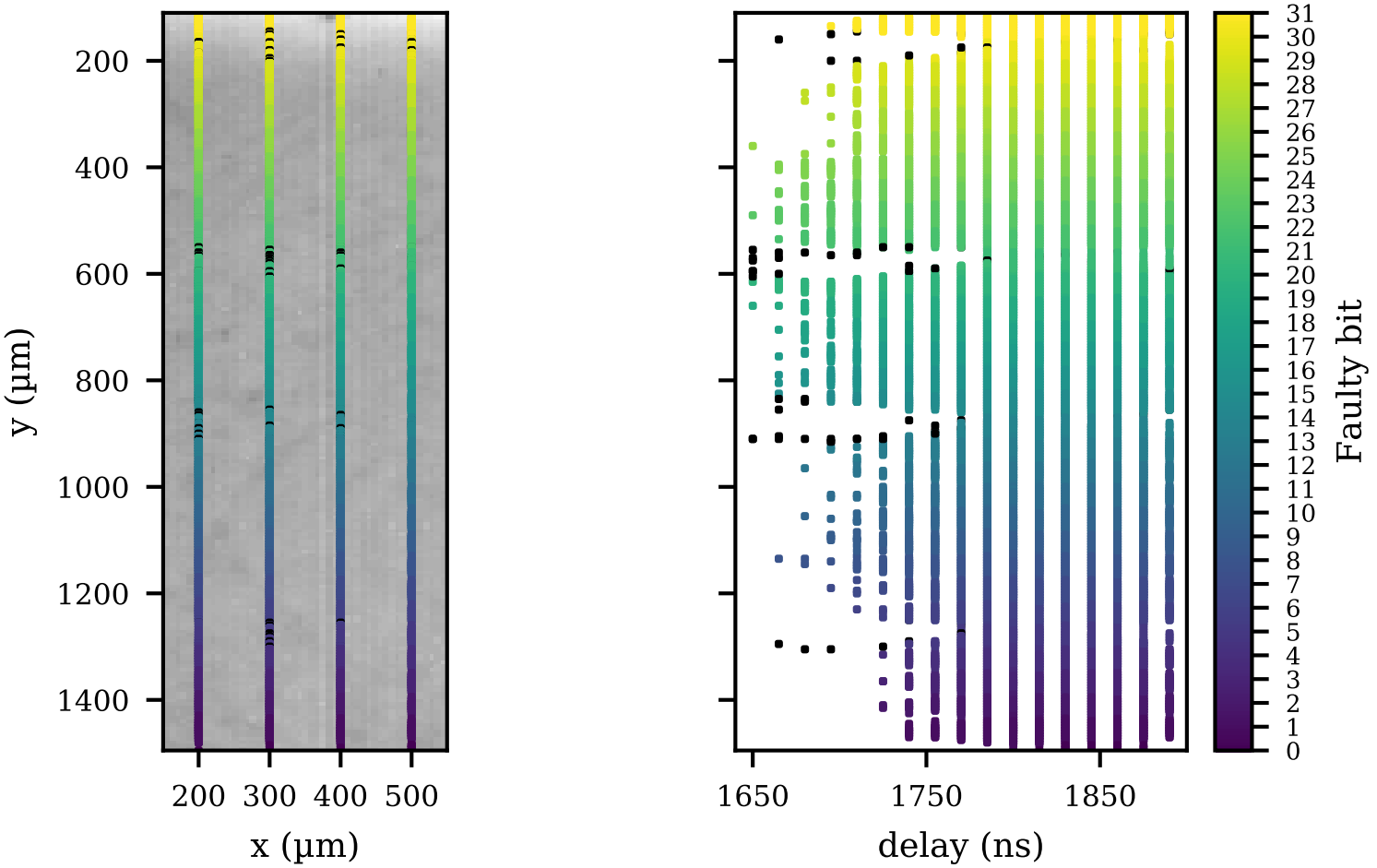
A **more complex** target (32 bits) implies:

- **Larger area** to cover for cartography (2.5x2.5cm),
- Complex micro-architecture,
- More time variability



Instruction corruption in Flash memory

A laser shot in flash memory alters the fetched data **on-the-fly**.



Fault model

Bit-set on data (and instructions) fetched from flash memory

Examples of instruction corruption

Modifying a `MOVW` instruction(32 bits).

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `MOVW`	1	1	1	1	0	i	1	0	0	1	0	0	imm4				0	imm3			Rd				imm8							
`MOVW, R0, 0`	1	1	1	1	0	i	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Data corruption

`MOVW, R0, 4`	1	1	1	1	0	i	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0

Destination register corruption

`MOVW, R1, 0`	1	1	1	1	0	i	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0

Opcode corruption

`MOVT, R0, 0`	1	1	1	1	0	i	1	0	**1**	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Laser fault injection on VerifyPIN

Constant-time implementation with hardened booleans.

```
1:  trials = 3
2:  ref_PIN[4] = {1, 2, 3, 4}
3:  procedure VerifyPIN(user_PIN[4])
4:      authenticated = FALSE
5:      diff = FALSE
6:      if trials > 0 then
7:          for i ← 0 to 3 do
8:              if user_PIN[i] != ref_PIN[i] then
9:                  diff = TRUE
10:         if diff == TRUE then
11:             trials = trials - 1
12:         else
13:             authenticated = TRUE
14:     return authenticated
```

C code:

```
if (trials > 0)
```

Assembly code:

```
CMP R3, 0
BLE address
```

Fault injection on the CMP instruction

Performing a bit-set at **index 10**.
Instead of comparing R3, we compare **R7**.
By design, R7 stores the frame pointer, **always positive**.

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reference instructions																
Generic `CMP`	0	0	1	0	1	Rd			imm8							
`CMP R3, 0`	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0
Register corruption						↓										
`CMP R7, 0`	0	0	1	0	1	**1**	1	1	0	0	0	0	0	0	0	0

Outcome

The *trials* value is never compared ➔ unlimited number of trials.

The PIN value can be **brute-forced**.

Laser fault injection on AES-128

```
1: procedure Add_round_key
2:   for i ← 0 to 3 do
3:     for j ← 0 to 3 do
4:       state[i][j] ^= round_key[round][i][j]
```

C code:

```
for (int i=0; i<4; i++){
  for (int j=0; j<4; j++){
    ...
  }}
```

Assembly code:

```
MOV R0, 0
addr_i:
MOV R1, 0
addr_j:
...
ADD R1, 1
CMP R1, 3
BLE addr_j
ADD R0, 1
CMP R0, 3
BLE addr_i
```

Faulting the loop variable increment

Add 5 instead of 1 after executing the body of the loop.

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reference instructions																
Generic `ADD`	0	0	1	1	0	Rd			imm8							
`ADD R0, 1`	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
Data corruption														↓		
`ADD R0, 5`	0	0	1	1	0	0	0	0	0	0	0	0	0	**1**	0	1

Outcome

The *for* loop exits after **one execution only**.

Faulting the for loops

Faulting the **inner** *for* loop on its first execution

$C_{0,0}$	$C_{1,0}$	$C_{2,0}$	$C_{3,0}$
$C_{0,1} \oplus K^{10}_{0,1}$	$C_{1,1}$	$C_{2,1}$	$C_{3,1}$
$C_{0,2} \oplus K^{10}_{0,2}$	$C_{1,2}$	$C_{2,2}$	$C_{3,2}$
$C_{0,3} \oplus K^{10}_{0,3}$	$C_{1,3}$	$C_{2,3}$	$C_{3,3}$

Faulting the **outer** *for* loop

$C_{0,0}$	$C_{1,0} \oplus K^{10}_{1,0}$	$C_{2,0} \oplus K^{10}_{2,0}$	$C_{3,0} \oplus K^{10}_{3,0}$
$C_{0,1}$	$C_{1,1} \oplus K^{10}_{1,1}$	$C_{2,1} \oplus K^{10}_{2,1}$	$C_{3,1} \oplus K^{10}_{3,1}$
$C_{0,2}$	$C_{1,2} \oplus K^{10}_{1,2}$	$C_{2,2} \oplus K^{10}_{2,2}$	$C_{3,2} \oplus K^{10}_{3,2}$
$C_{0,3}$	$C_{1,3} \oplus K^{10}_{1,3}$	$C_{2,3} \oplus K^{10}_{2,3}$	$C_{3,3} \oplus K^{10}_{3,3}$

What is left?

Holding one correct and two faulty ciphertexts, the attacker only needs to brute-force the tenth round-key byte $K^{10}_{0,0}$ ➔ 2^7.

Conclusion on laser faults in flash memory

Capabilities

- **Temporarily alter instruction/data** from flash memory,
- Corrupt the **control flow** of a program,
- **Weaken security** of embedded programs.

Limitations

- **Bit-set** only (so far),
- **Adjacent** bits only,
- **Control-flow** corruption mostly.

Future possibilities

- **Multispot** laser

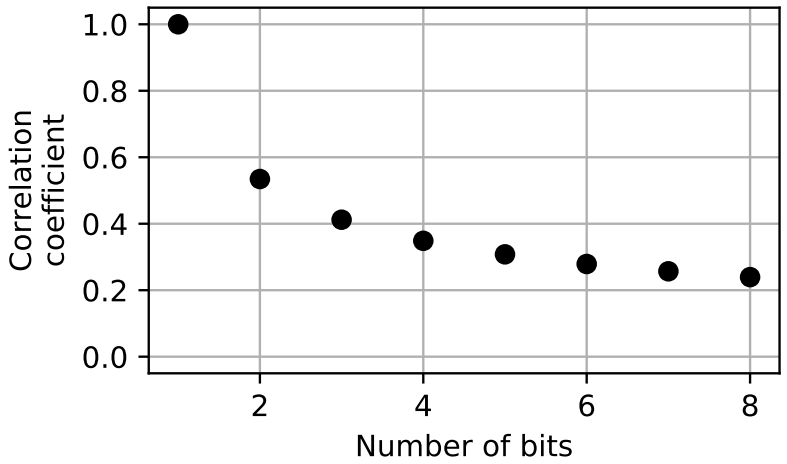
Combination of protections

For the best: 2nd order CPA made harder

Principle of 2nd order CPA: attack two S-box output bytes. Traditionally, target the two shares (mask + masked value) but two consecutive bytes work well:

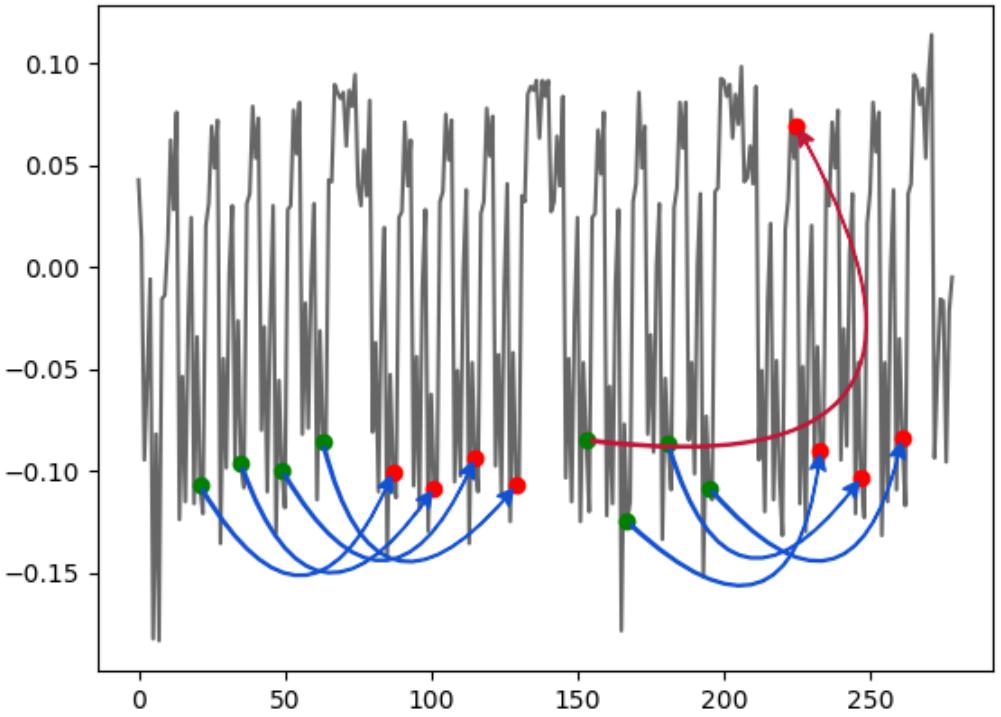
- $|\mathrm{Leak}(\mathrm{Sbox}(P_i \oplus K_i) \oplus M') - \mathrm{Leak}(\mathrm{Sbox}(P_j \oplus K_j) \oplus M')|$
- $\mathrm{HW}(\mathrm{Sbox}(P_i \oplus K_i) \oplus M' \oplus \mathrm{Sbox}(P_j \oplus K_j) \oplus M')$

$= \mathrm{HW}(\mathrm{Sbox}(P_i \oplus K_i) \oplus \mathrm{Sbox}(P_j \oplus K_j))$ ➔ no more mask !



For the best: 2nd order CPA made harder

Combining leakages is **easy** when traces are **perfectly synchronised**.



800 traces required to break 1st-roder masked AES on STM32.

A **desynchronising countermeasure** is very **powerful** here!

For the worst

Countermeasure against FA **or** SCA are usually compatible.

Countermeasure against FA **and** SCA can be incompatible.

Example

Redundancy-based protection against Fault Injection Analysis can **enhance** side-channel leakages…

Side-Channel Analysis is not only for key recovering purpose, it also helps in temporaly **profiling** fault injection (bypassing secure boot [6])

Each case must be evaluated **separately**.

[6] Niek Timmers, Albert Spruyt, Bypassing Secure Boot using Fault Injection, Black Hat Europe 2016

Conclusion

Conclusion

- Inserting protections at **software level** is powerful
- Leakage assessment is a great tool to design protections
  - Provides **metrics** of leakage reduction efficiency
- New attack on **flash memory** of a 32-bit microcontroller
- **Combinations** of protections is a **double-edged** sword

Conclusion

- Inserting protections at **software level** is powerful
- Leakage assessment is a great tool to design protections
  - Provides **metrics** of leakage reduction efficiency
- New attack on **flash memory** of a 32-bit microcontroller
- **Combinations** of protections is a **double-edged** sword

— Questions ? —

Contacts: Brice Colombier
brice.colombier@cea.fr

Pierre-Alain Moëllic
pierre-alain.moellic@cea.fr

Commissariat à l'énergie atomique et aux énergies alternatives
17 rue des Martyrs | 38054 Grenoble Cedex
www.cea-tech.fr

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019