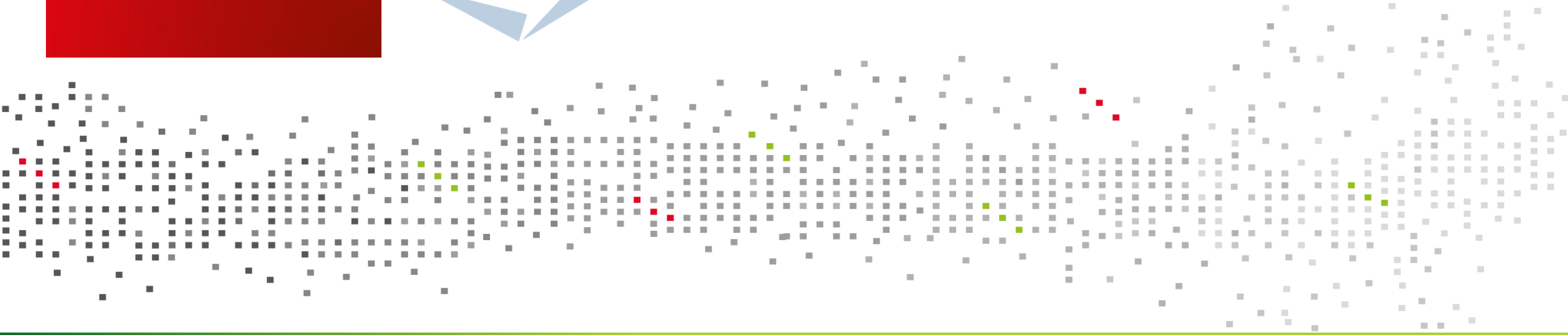


AN END-TO-END APPROACH FOR MULTI-FAULT ATTACK VULNERABILITY ASSESSMENT

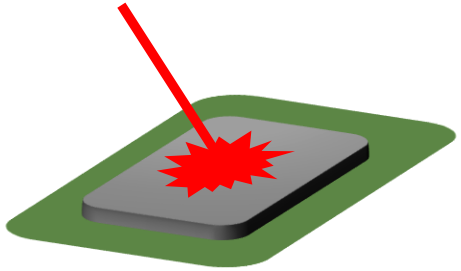
FDTC 2020 | **Vincent Werner**[1 2] Laurent Maingault[1] Marie-Laure Potet[2] | 13/09/2020

[1] Univ. Grenoble Alpes, CEA, LETI, DSYS, CESTI, F-38000 Grenoble, first.last@cea.fr
[2] Univ. Grenoble Alpes, CNRS, VERIMAG, F-38000 Grenoble, first.last@univ-grenoble-alpes.fr

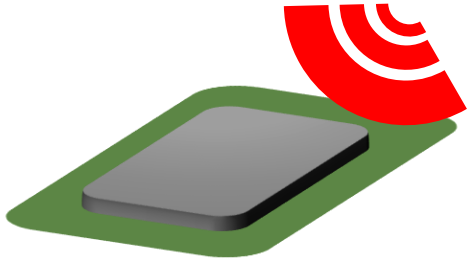
This work is supported by the French National Research Agency in the framework of the ”Investissements d’avenir” program (ANR-15-IDEX-02 and ANR-10-AIRT-05).

CONTEXT

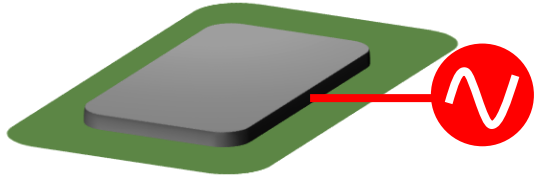
- Fault attack can leak information



Laser



EM



Power/Clock glitch

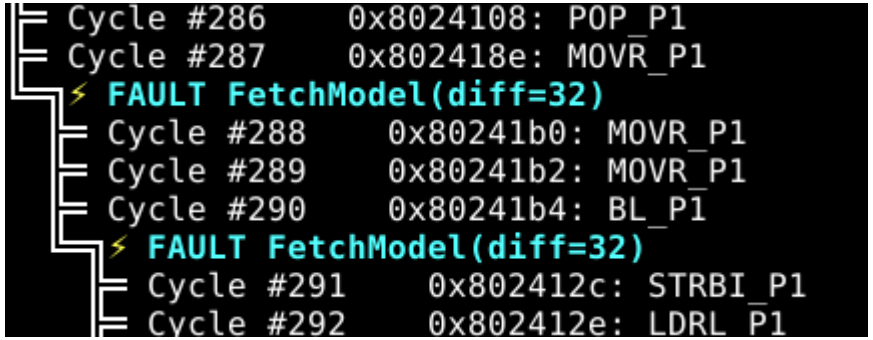
- Difficulties to find fault attack in embedded **software** :
  1. Find implementation weaknesses ➔ Fault Analysis ➔ **Fault model assumptions ??**
     ⚠ Wrong assumption ➔ False positives, miss potential fault attacks
  2. Fault Exploitation ➔ Equipment Configuration ➔ **Fault injection settings ??**
     ⚠ Too many combinations possible
  3. The more faults we inject, the harder the attack
     ⚠ Combinatorial explosion

THE THREE MAIN CHALLENGES TO DO MULTI-FAULT ATTACKS

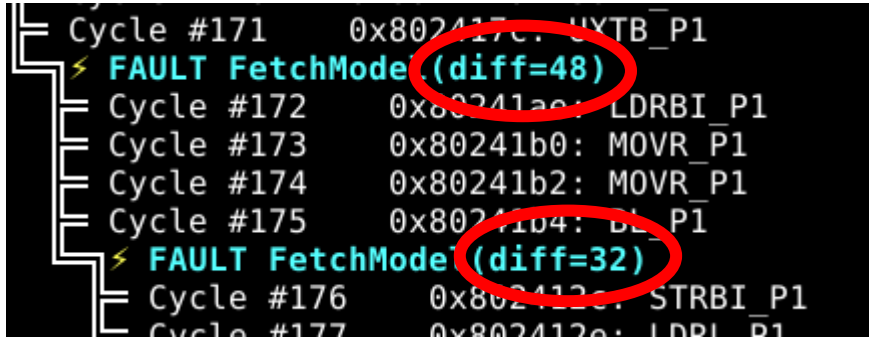
- **Challenge n°1** Reduce the gap between fault analysis and fault exploitation
  - Stronger fault model assumptions

- **Challenge n°2** Improve the selection of fault injection settings
  - Fault injection settings selection according to fault models

- **Challenge n°3** Find multi-fault attacks with different fault models ➔ **Combined Fault Attacks**
  - ⚠ Combinatorial explosion
  - Open new attack paths
  - Find unnoticed vulnerabilities



Same fault models



Different fault models

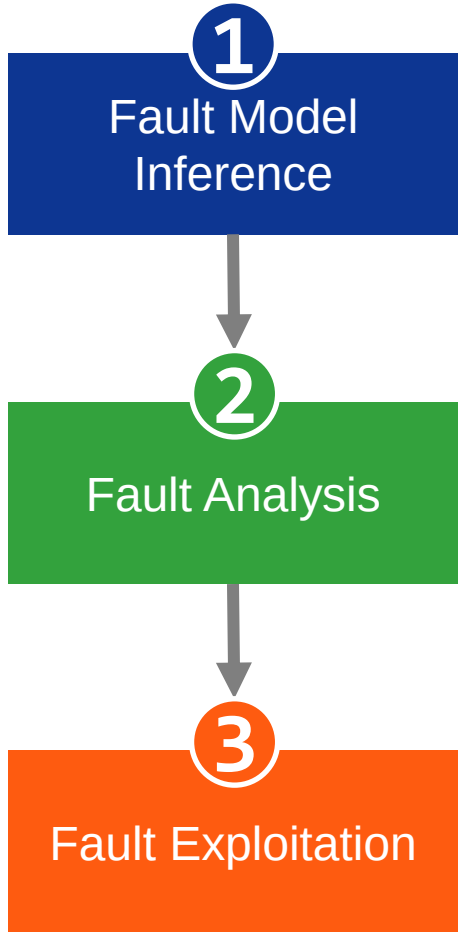
METHODOLOGY OVERVIEW

- **Proposition** A 3-step end-to-end approach

- **Step 1** Tool-assisted **fault model inference**
  - Find target specific fault models
  - Better fault model assumptions
  - Improve the fault injection settings selection

- **Step 2** Tool-assisted **fault analysis**
  - With target specific fault models
  - Find efficiently combined fault attacks
  - Fewer false positives

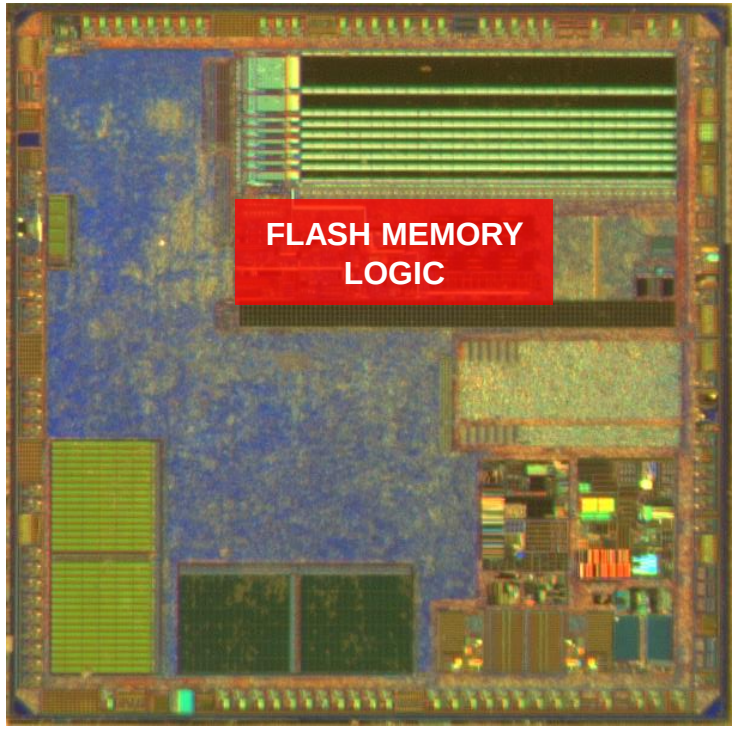
- **Step 3** Tool-assisted **fault exploitation**
  - Generate full equipment configuration



APPLICATION ON A REAL TARGET

- Target Chip: ARM Cortex M4
  - 32-bit processor
  - 3-stage pipeline
  - Widely-used in embedded systems
  - Target Area: Flash Logic
    - To perturb fetch/decode stage of the pipeline

- Target Application
  - Another VerifyPin
  - Authentication program
  - Hardened with software countermeasures
  - Robust to single-fault attacks



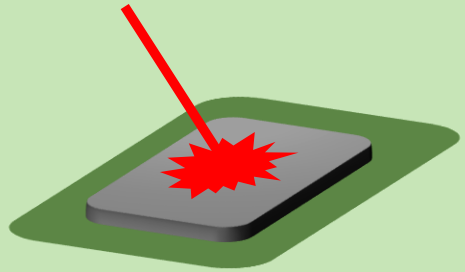
FAULT MODEL INFERENCE
➔ QUICK OVERVIEW, 3 SUB STEPS

1 Tool-assisted Fault Models Inference → 2 Tool-assisted Fault Analysis → 3 Tool-assisted Fault Exploitation

Characterization

Fault injection settings

(x=12µm, y=10µm, delay=10µs)



Faulty outputs

000068BE 00007A2C

Fault Injection Simulation

ISA Fault Models

INSTRUCTIONSKIP_32

Faulty outputs

000068BE 00007A2C

Target Specific Fault Models Generation

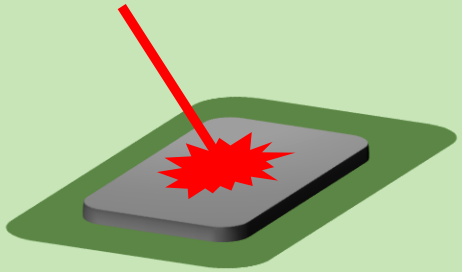
INSTRUCTIONSKIP_32	(x=12µm, y=10µm, delay=10µs)

FAULT MODEL INFERENCE
➔ CHARACTERIZATION & TEST PROGRAM

Characterization

Fault injection settings

(x=12µm, y=10µm, delay=10µs)



Faulty outputs

000068BE 00007A2C

- Find fault injection settings
- **Use test program**
  - Easier to propagate errors
  - Generate more faulty outputs

```
INIT(); # initialize registers
TRIGGER_IO(); # easier synchro
ADD R0, R0, #2
ADD R1, R1, #3
ADD R2, R2, #5
… # several times
SEND_RESULT(); # send result to PC
```

- Main assumption
  - Faults **should not depend** on the executed code
  - Faults **should depend** on the fault injection settings

➔ **Same** fault model for **different** applications

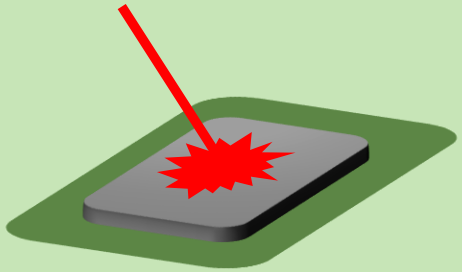
➔ Characterization results are **transferable** from sample to sample

FAULT MODEL INFERENCE
➔ CHARACTERIZATION & RESULTS

Characterization

Fault injection settings

(x=12µm, y=10µm, delay=10µs)

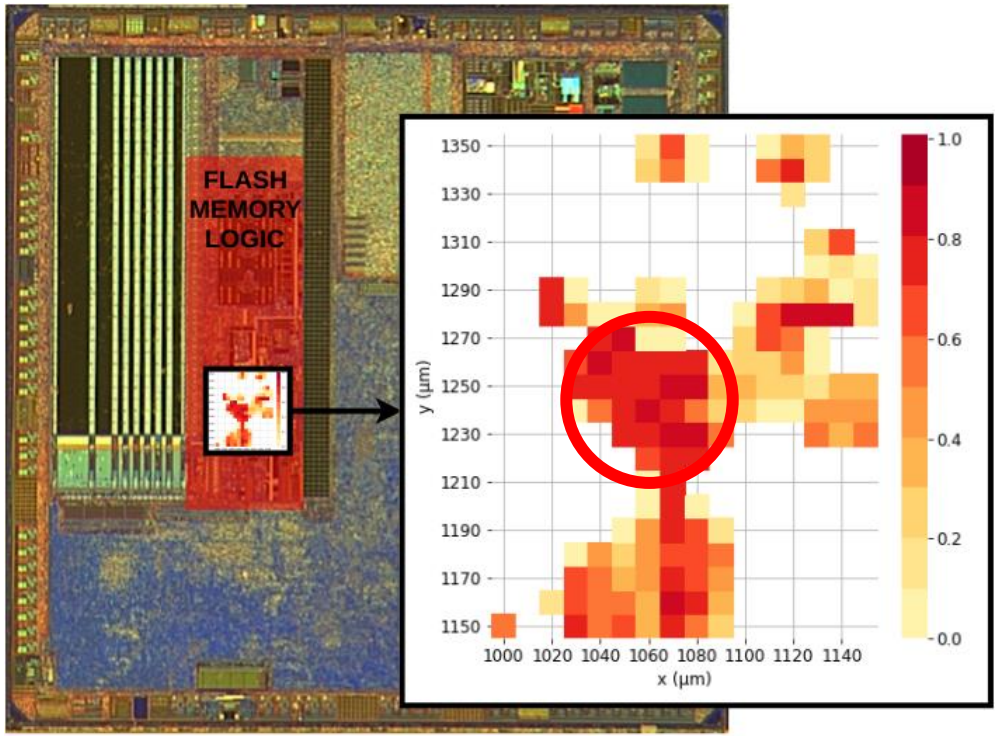


Faulty outputs

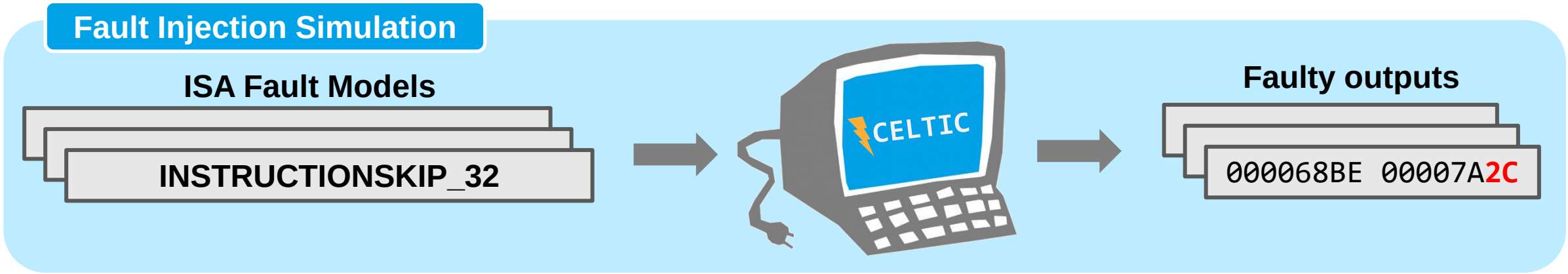
000068BE 00007A2C

- ~50,000 fault injection settings tested in 6 hours
- ~12,000 faulty outputs
- Laser Fault injection:
  - Fixed power, fixed pulse duration
  - Variable delay, variable positions
  - Try to find different fault models using different positions
- Some area more sensitives
  - Some faults do not depend on the injection delay

  ➔ do not depend on the instruction executed.



FAULT MODEL INFERENCE
➔ FAULT INJECTION SIMULATION



- CELTIC, a simulation-based fault injection tool at **binary level**
- CELTIC simulates ISA fault models:
  - "A fault that jumps eight 32-bit instructions" ➔ PC = PC + 32 ➔ **INSTRUCTIONSKIP_32**
- Database generation with faulty outputs based on known fault models
- **Same test program**
- Emulation of the target architecture using CELTIC

FAULT MODEL INFERENCE
➔ FAULT INJECTION SIMULATION & RESULTS

1 Tool-assisted Fault Models Inference → 2 Tool-assisted Fault Analysis → 3 Tool-assisted Fault Exploitation

Fault Injection Simulation

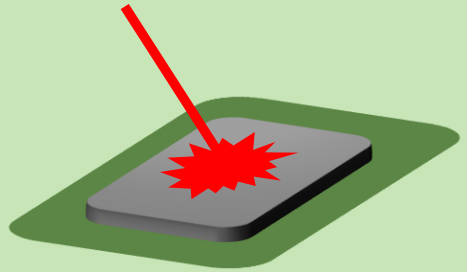
- Simulation of instruction jumps and opcode bit flips
- ~100 fault models
- 5 min simulation
- 50,000 faulty outputs

FAULT MODEL INFERENCE
➔ TSFM GENERATION

Characterization

Fault injection settings

(x=12µm, y=10µm, delay=10µs)



Faulty outputs

000068BE 00007A2C

Fault Injection Simulation

ISA Fault Models

INSTRUCTIONSKIP_32

Faulty outputs

000068BE 00007A2C

Target Specific Fault Models Generation

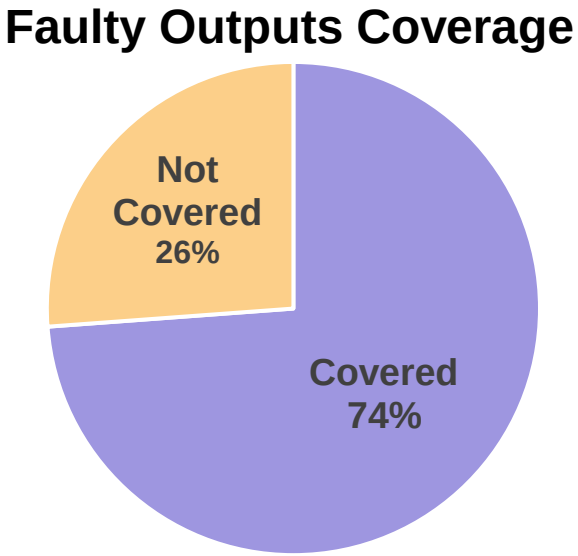
INSTRUCTIONSKIP_32	(x=12µm, y=10µm, ~~delay=10µs~~)

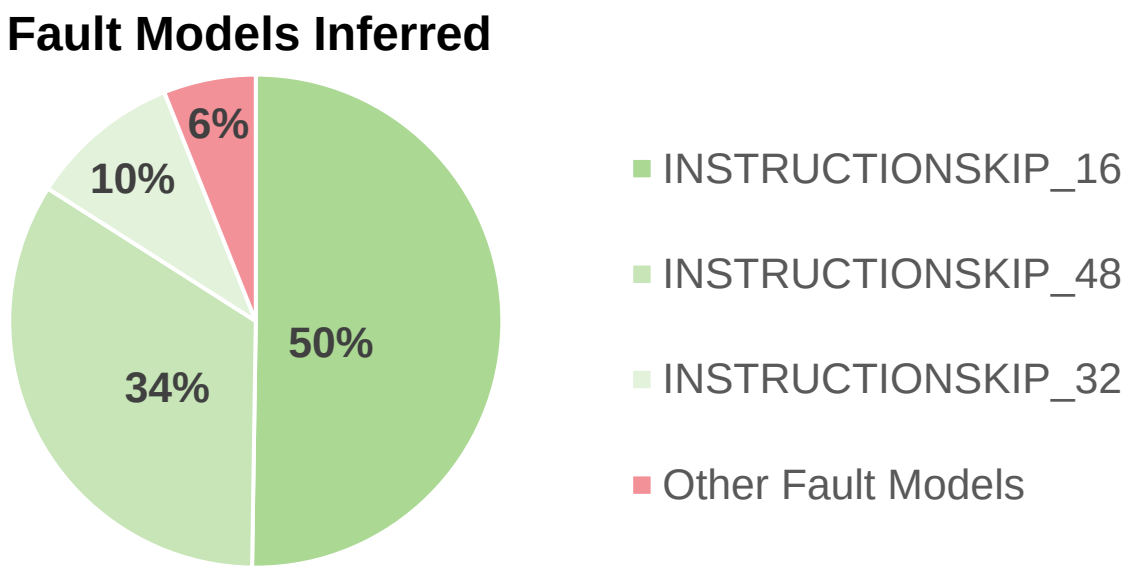
Faults should not depend on injection delay

FAULT MODEL INFERENCE
➔ DO WE FIND ALL THE FAULT MODELS ?

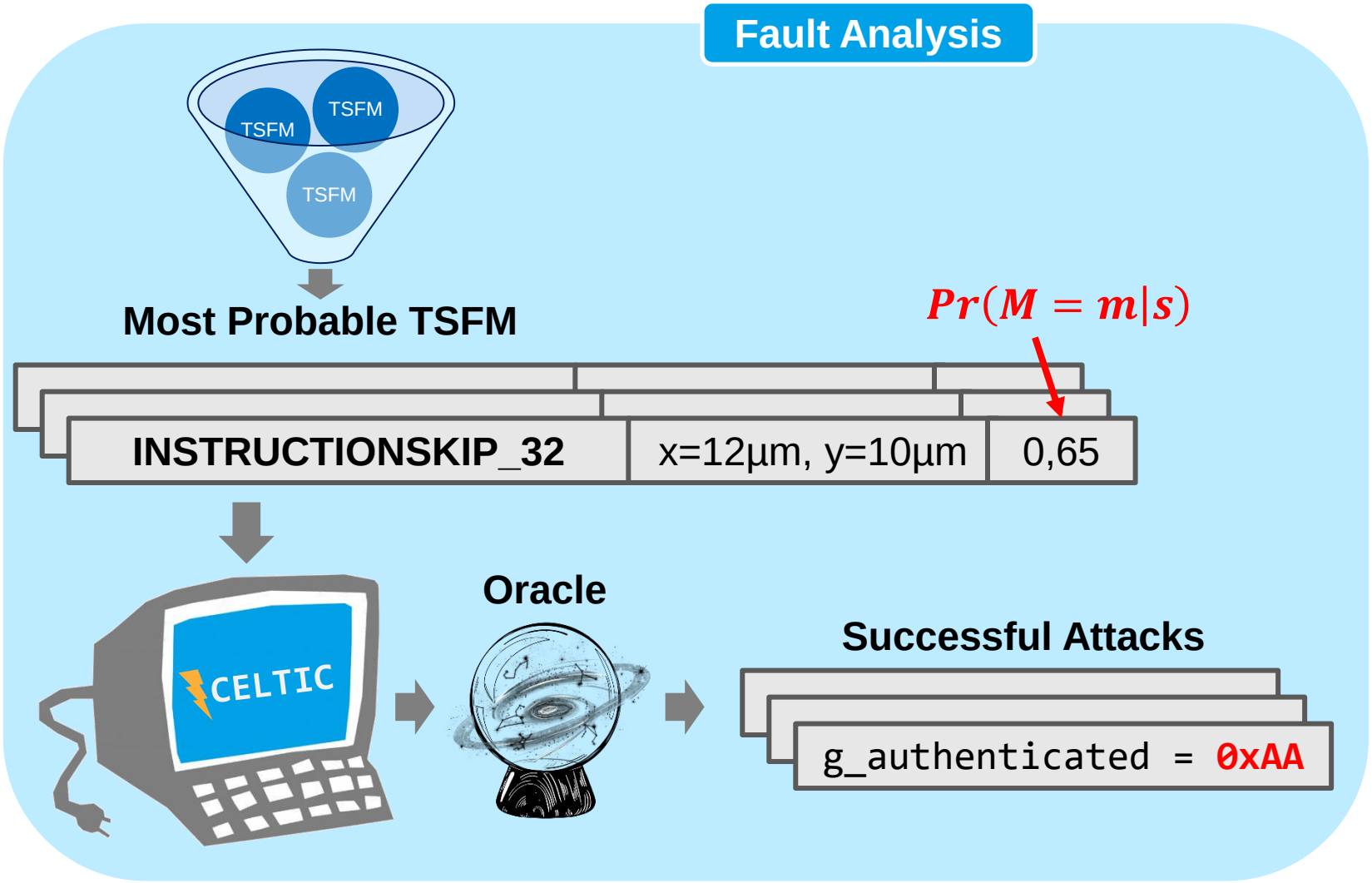
- ~12,000 faulty outputs
  - ~9,000 faulty outputs covered
  - Faulty output coverage rate is around **74%**

- The most probable fault models are instruction jumps (94% of the fault models found)
  - Not a surprise ➔ Fault in Flash Memory



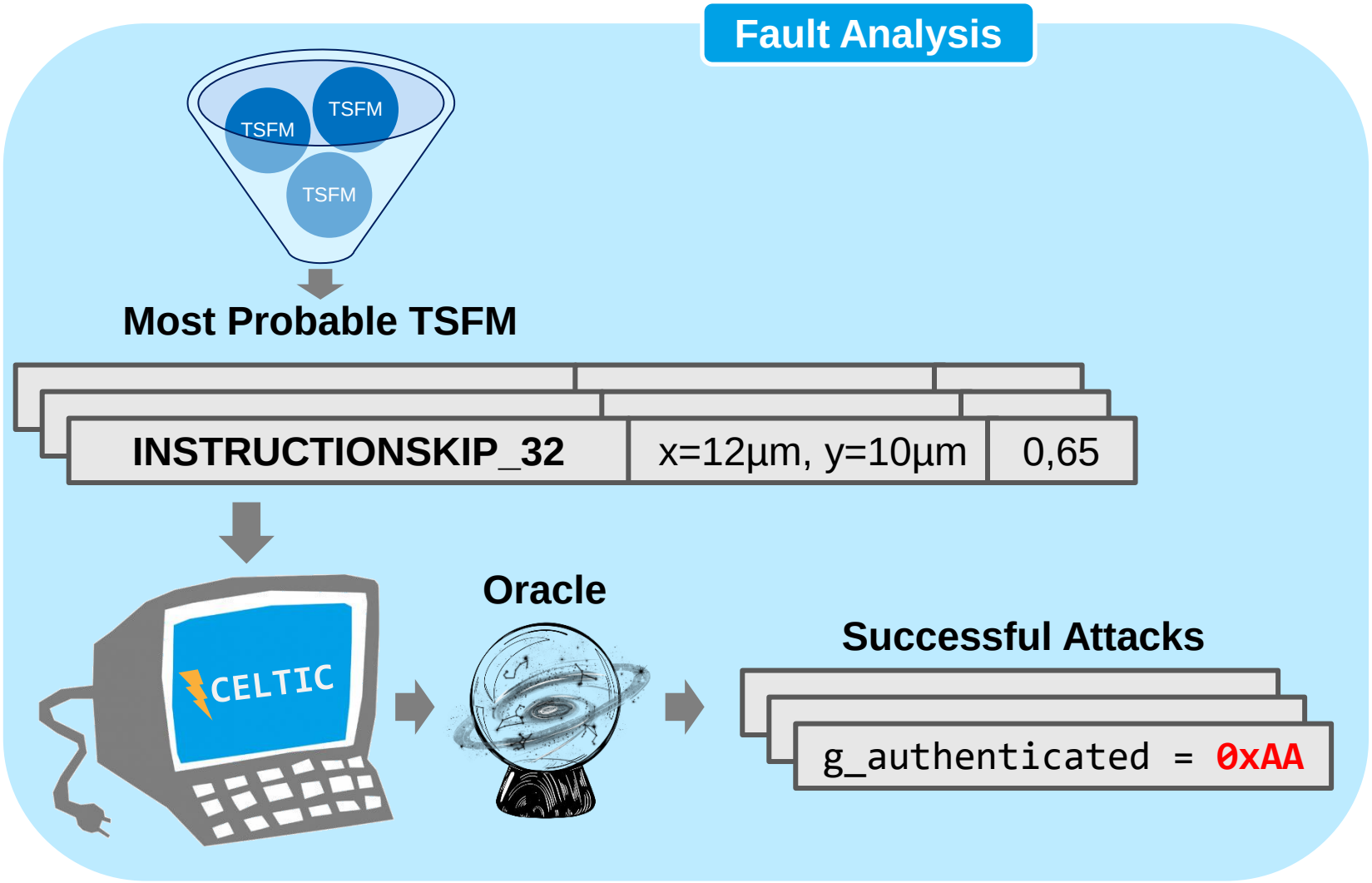


FAULT ANALYSIS
➔ SELECTION MOST PROBABLE TSFM



- Keep the most probable TSFM
- Max the probability $\Pr(M = m|s)$
- Advantages:
  - Increase attack exploitation success rate
  - Reduce combinatorial explosion of the fault analysis

FAULT ANALYSIS
➔ FIND SUCCESSFUL ATTACKS



- CELTIC simulates selected fault models
- **Target application**
- Set an oracle ➔ Victory Conditions
- Find successful attacks

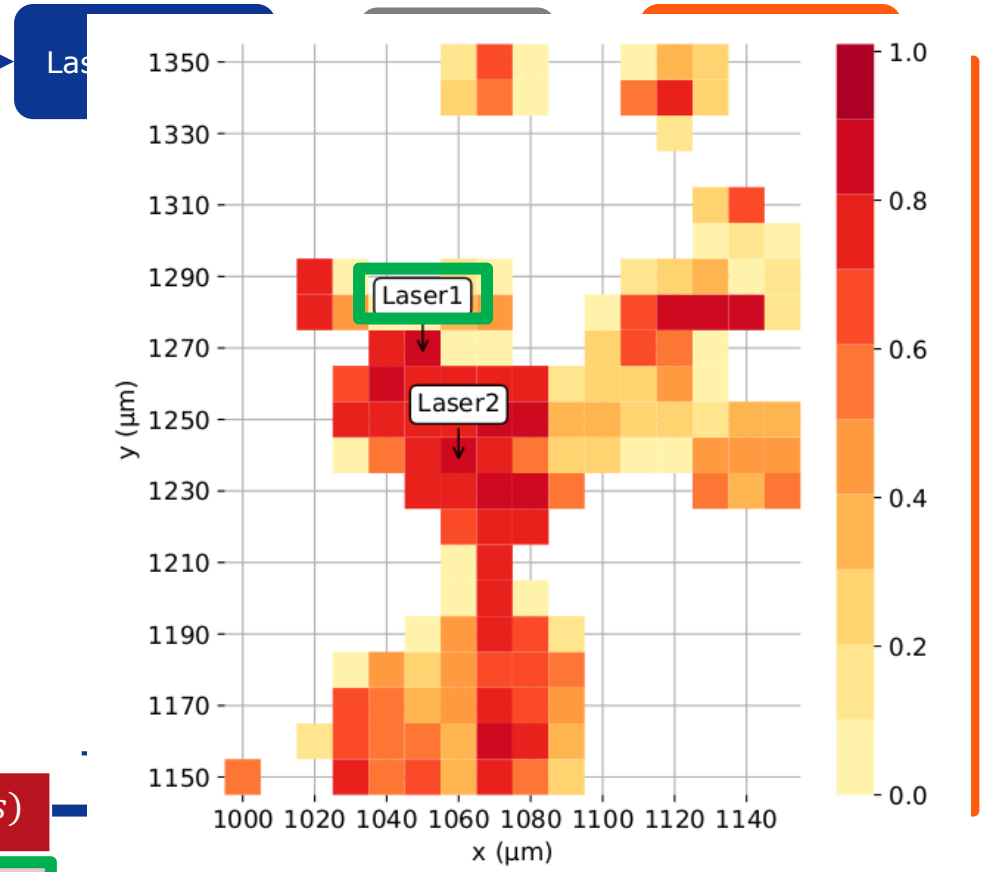
FAULT ANALYSIS
➔ QUICK DEMO

FAULT EXPLOITATION
➔ SETUP & POSITIONS

- Combined fault attacks using 2 lasers.

- Laser Fault Injection with 2 laser sources
  - Independent IR Lasers
  - Different positions
  - Different injection delays
  - Same power
  - Same pulse duration

- Lens x20
  - The field of view limits fault models we can do



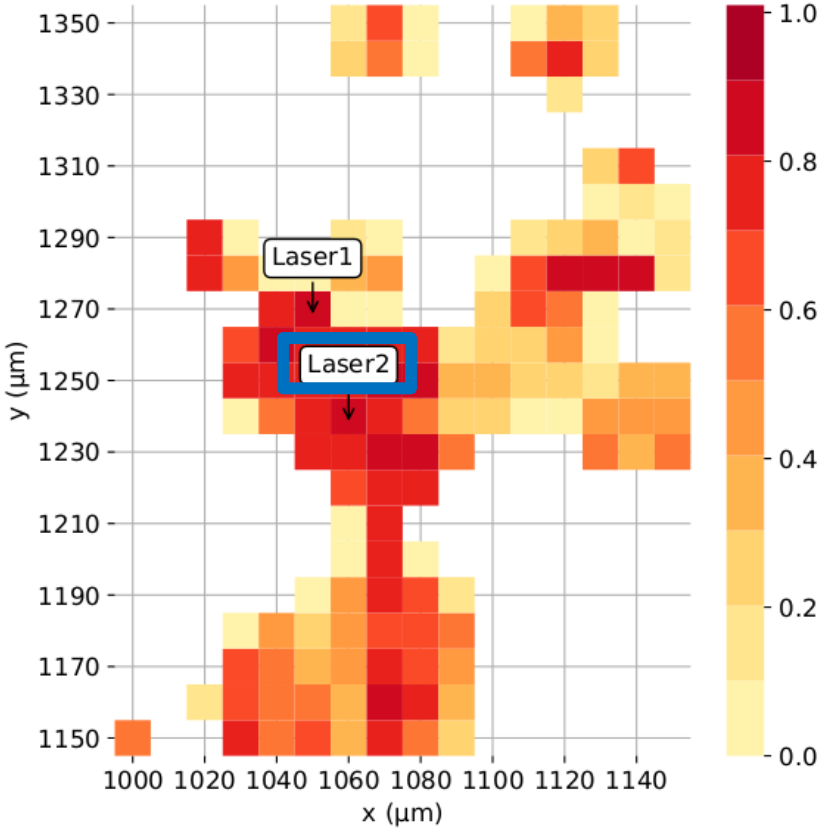
Laser	Fault model	Positions	$\Pr(M = m\|s)$
Laser 1	**INSTRUCTIONSKIP_48**	X=1050 µm, Y=1270 µm	0,72
Laser 2	**INSTRUCTIONSKIP_32**	X=1060 µm, Y=1240 µm	0,68

FAULT EXPLOITATION
➔ SETUP & POSITIONS

- Combined fault attacks using 2 lasers.

- Laser Fault Injection with 2 laser sources
  - Independent IR Lasers
  - Different positions
  - Different injection delays
  - Same power
  - Same pulse duration

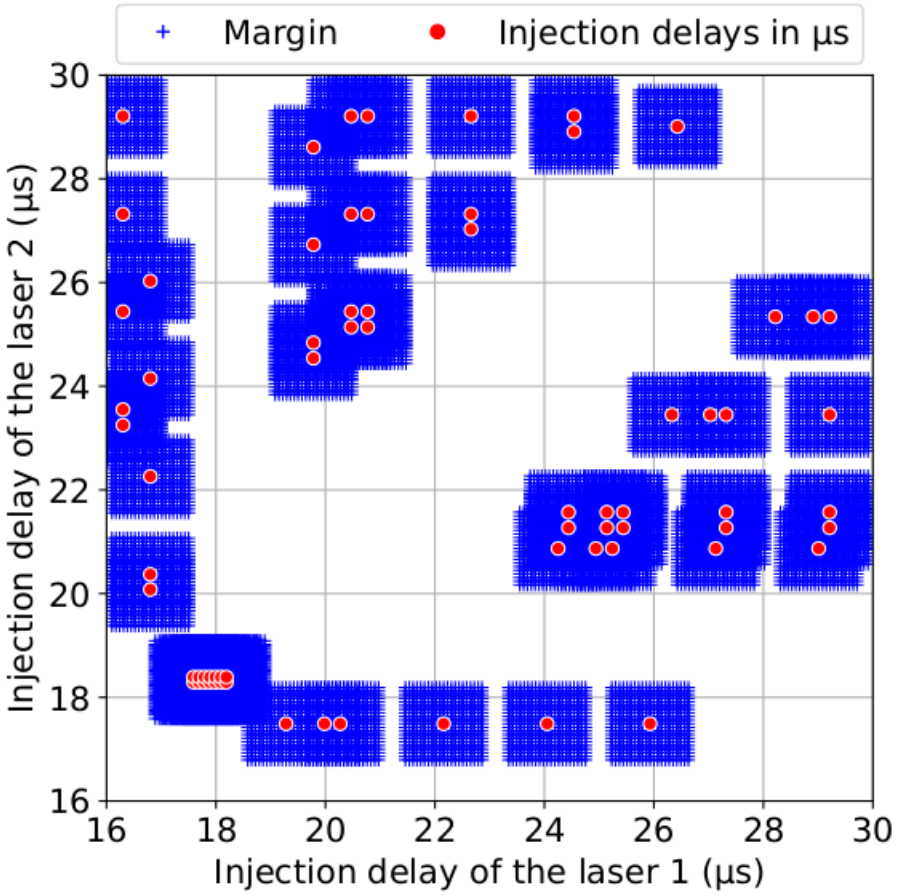
- Lens x20
  - The field of view limits fault models we can do



Laser	Fault model	Positions	$\Pr(M = m\|s)$
Laser 1	**INSTRUCTIONSKIP_48**	X=1050 µm, Y=1270 µm	0,72
Laser 2	**INSTRUCTIONSKIP_32**	X=1060 µm, Y=1240 µm	0,68

FAULT EXPLOITATION
➔ INJECTION DELAYS

- CELTIC find injection delays in clock cycle
- We want injection delay in µs rather than in clock cycle
  - Conversion with a linear relationship
- Mitigation of potential inaccuracies:
  - Target synchronization
  - CELTIC doesn't simulate pipeline stage
  - ISA models are less accurate than RTL models
- Margin of error
  - In this example 10 clock cycles

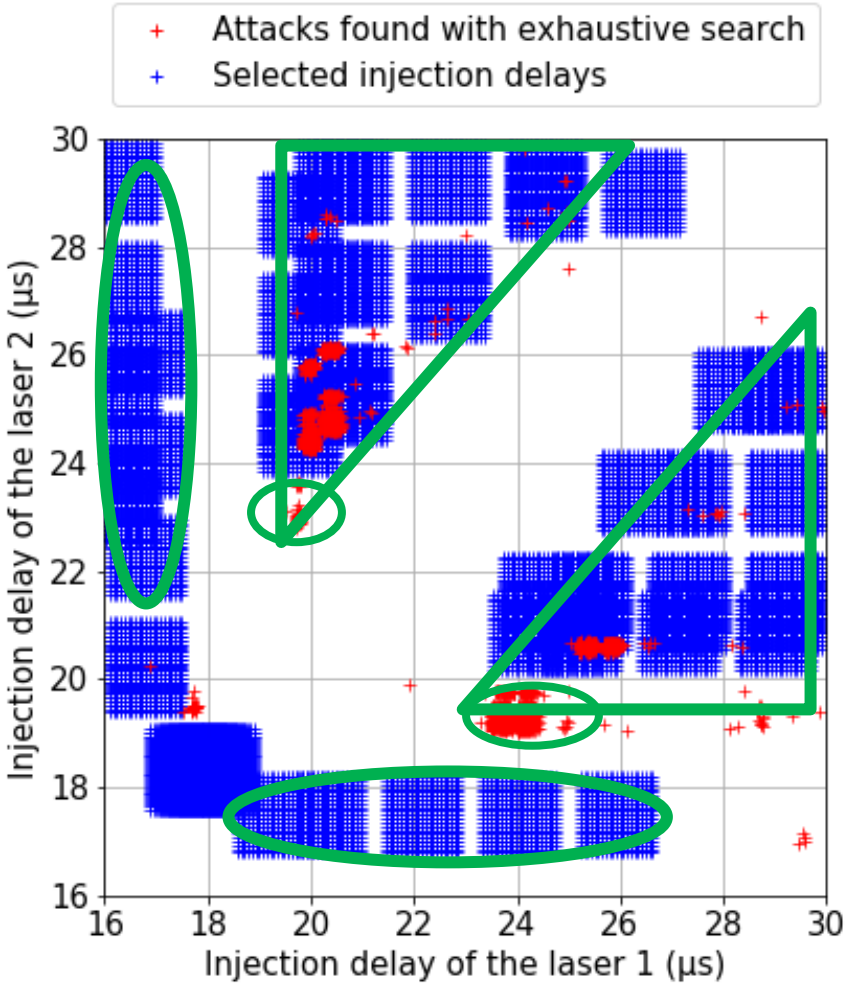


FAULT EXPLOITATION
➔ DO WE FIND ALL THE FAULT ATTACKS ?

- Comparison between **exhaustive search** and **our approach**.
- Exhaustive search on injection delays configuration:
  - 1st laser ➔ **INSTRUCTIONSKIP_48**
  - 2nd laser ➔ **INSTRUCTIONSKIP_32**
  - During **1 week**

- **Pros:**
  - We find ~900 attacks out of ~1800 possible (50%).
  - We identify the triangular patterns

- **Cons:**
  - Still miss 50% of the possible attacks
  - We have also false positives

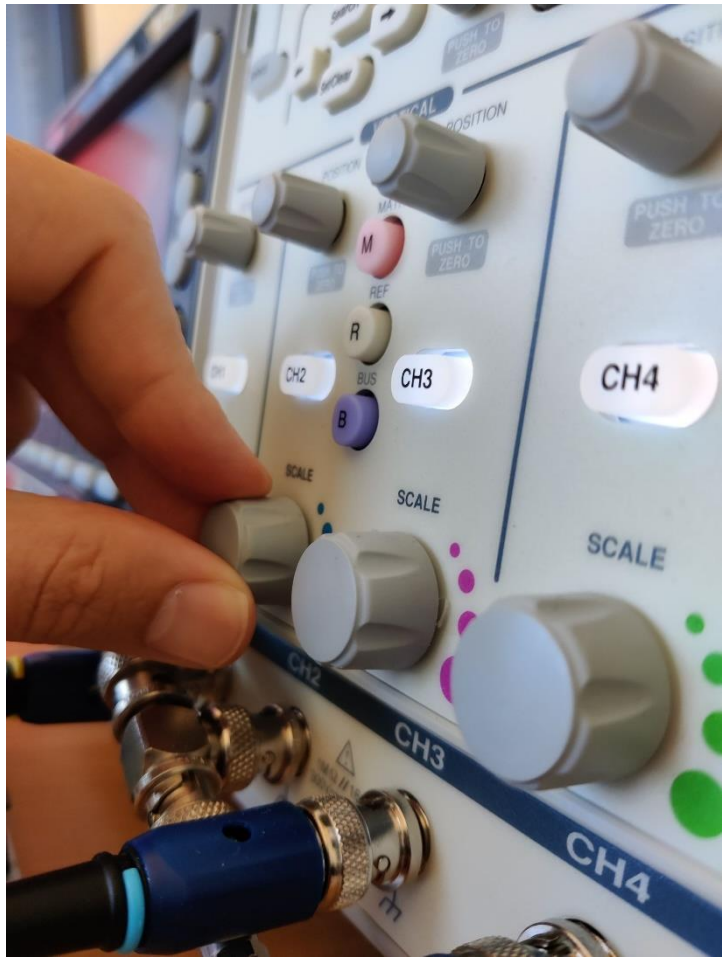


FAULT EXPLOITATION
➔ IS OUR APPROACH THE FASTEST ?

1 Tool-assisted Fault Models Inference → 2 Tool-assisted Fault Analysis → 3 Tool-assisted Fault Exploitation

- Comparison between 3 approaches:
  - Approach A : Naïve approach ➔ exhaustive search
  - Approach B : Hybrid approach ➔ characterization
  - Approach C : Our approach

- Goals
  - Be the fastest to get authenticated
  - 100 repetitions for statistical purposes



FAULT EXPLOITATION
➔ IS OUR APPROACH THE FASTEST ?

- Naive approach (Approach A) did not pass the experiment within a reasonable time.
- Our approach (Approach C) is **3 times faster** on average than characterization only (Approach B)

- The VerifyPIN is a short program (~200 clock cycles),

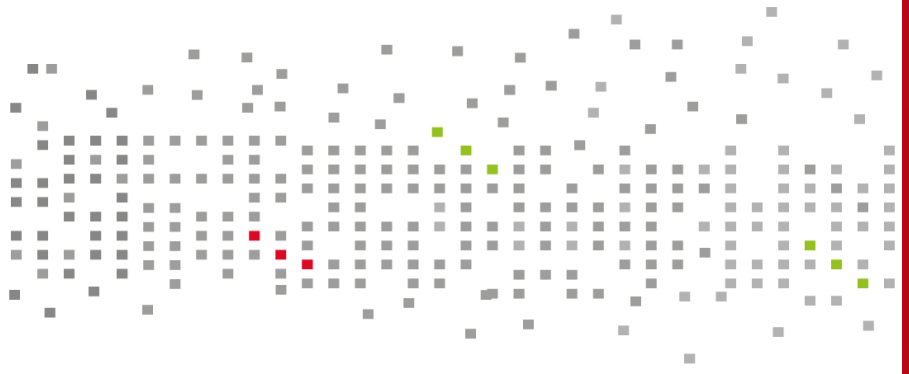
➔ Elapsed time difference **could** be bigger on a longer program

	B	C
Avg Elapsed Time	13min58sec	4min18sec
Max Elapsed Time	2h35min59sec	31min04s

CONCLUSION

- We have presented the whole methodology step by step
- We have find multi-fault attacks with different fault models
  - Complex fault attacks
  - Difficult to find them without proper methodology
- Our approach is 3 times faster on average than characterization to find combined fault attacks on a VerifyPIN
- Further Work:
  - Test different target devices and target applications
    - Secured microcontrollers + Larger applications
  - Test different fault injection techniques
    - Power glitch fault injection

Questions ?



Leti, technology research institute
Commissariat à l'énergie atomique et aux énergies alternatives
Minatec Campus | 17 avenue des Martyrs | 38054 Grenoble Cedex | France
www.leti-cea.com

This work is supported by the French National Research Agency in the framework of the "Investissements d'avenir" program (ANR-15-IDEX-02 and ANR-10-AIRT-05).