

An End-to-End Approach for Multi-Fault Attack Vulnerability Assessment

Vincent Werner, Laurent Maingault, Marie-Laure Potet

▶ **To cite this version:**

Vincent Werner, Laurent Maingault, Marie-Laure Potet. An End-to-End Approach for Multi-Fault Attack Vulnerability Assessment. 2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC 2020), Sep 2020, Milan (virtuel), Italy. 10.1109/FDTC51366.2020.00009 . hal-03248430

HAL Id: hal-03248430
https://hal.science/hal-03248430

Submitted on 3 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

An End-to-End Approach for Multi-Fault Attack Vulnerability Assessment

Vincent Werner*†, Laurent Maingault*, Marie-Laure Potet†
* *Univ. Grenoble Alpes, CEA, LETI, DSYS, CESTI, F-38000 Grenoble, first.last@cea.fr*
† *Univ. Grenoble Alpes, CNRS, VERIMAG, F-38000 Grenoble, first.last@univ-grenoble-alpes.fr*

***Abstract*—Although multi-fault attacks are extremely powerful in defeating sophisticated hardware and software defences, detecting and exploiting such attacks remains a difficult problem, especially without any prior knowledge of the target. Our main contribution is an end-to-end approach for multi-fault attack vulnerability assessment We take advantage of target specific fault models rather than generic fault models to achieve complex multi-fault attacks that can lead to critical vulnerabilities. Target specific fault models are generated thanks to fault models inference process, based on a fault injections simulation and a characterization, in order to elaborate powerful multi-fault attacks based on different fault models. Combining fault models opens up new possible attack paths and adds flexibility to design fault attacks that adapt to countermeasures. Hence, the direct consequence of the increasing complexity of fault attacks question the effectiveness of software countermeasures based on generic fault models for sensitive applications.**

***Keywords*-Multi-Fault Attack, Fault Injection Simulation, Laser Fault Injection, Microcontroller**

I. Introduction

Today secure embedded system are widespread and we trust them to manipulate our sensitive private data. Such devices rely on cryptographic algorithms to ensure confidentiality and integrity properties. Although these algorithms use strong mathematical concepts to provide provable security, their implementation can be insecure and attackers exploit this fact to retrieve cryptographic information. Fault attacks are one way to take advantage of implementation weaknesses. The first theoretical fault attack based on analysing incorrect cipher-texts was introduced by Boneh et al. in 1997 [1] and referred to as the *Bellcore attack*. Five years later, Aumuller et al. [2] proposed the first practical investigation of the Bellcore attack using electrical glitches. Since then, recent works have established that it is possible to use focused light, electromagnetic injections or even nanofocused X-rays to generate local environmental perturbations in order to induce faults and cause computational errors.

As a result, software and hardware defences have been designed to mitigate fault attacks. But if countermeasures are reliable against single fault injection, they are less effective against multi-fault attacks since the countermeasure itself can also be attacked. Kim and Quisquater [3] describe for the first time a practical double fault attack on a CRT-RSA algorithm implemented with first order fault countermeasures. Nowadays, the multi-fault analysis is usually performed *manually*, with *generic* fault models and without considering *combined fault attacks*, leaving significant gaps in coverage. As a consequence, some unnoticed potential weaknesses can be exploited. The security evaluation process can be improved by adopting an end-to-end approach for multi-fault attack vulnerability assessment, from the fault analysis to the fault injection to improve the overall target coverage. However, to the best of our knowledge, this approach does not exist yet.

Accordingly, our main contribution is an end-to-end *tool-assisted* methodology that considers *target specific* fault models as well as *combined fault attacks*. The methodology we propose is divided into three parts; 1) infer target specific fault models, 2) simulate realistic multi-fault attacks using inferred fault models and 3) select the most efficient fault injection settings to calibrate the fault injection equipment to perform combined fault attacks.

The outline of the rest of the paper is as follows. In section II, we present the context, as well as some preliminary notions. Then, we detail the well-known challenges of multi-fault attack vulnerability assessment and we discuss previous related work, providing a brief overview of recent state-of-the-art fault model inference approaches. Finally, we present our contributions to address these challenges. In section III, we explain our end-to-end approach, step by step, from the fault inference to the selection of the most efficient fault injection settings. In section IV, we evaluate our end-to-end methodology on a VerifyPIN algorithm. In section V, we discuss the limitations of our approach. Finally, we conclude this article and give insights for future perspectives.

II. Background

A. Security Evaluation Process

The evaluation process is part of the security certification to identify weaknesses of the Target Of Evaluation (TOE). The Information Technology Security Evaluation Facility (ITSEF), a third party agreed by the certification body, is in charge of evaluating the product robustness against various attacks, and especially fault attacks. The evaluation is based on a conformity analysis (not detailed in this document) and a vulnerability analysis, which includes two main steps:

Fault analysis By detecting implementation weaknesses that could lead to vulnerabilities, potential fault attacks are identified.

Fault exploitation Potential fault attacks are performed on the target using the appropriate equipment according to the fault injection technique.

B. Preliminary Notions

Fault: A *fault* is the cause of an *error*, that is an incorrect program state. If the error caused by the fault does not propagate and the program execution ends normally, the fault is *ineffective*. On the contrary, the fault is *effective* if the error affects the execution of the program, causing a *failure*, that is, an observed behaviour different from that expected. Although it is possible to exploit weak implementations with ineffective faults, most successful fault attacks take advantage of system failures to leak critical information.

Fault model: A fault model is an abstract description of a supposed fault. These models are used during the fault analysis to find potential implementation weaknesses. As fault models are interpretations of observed behaviours, the same fault can be formalized differently. The level of abstraction of the model can be chosen according to the analysis level considered. As the target binary is often available during security evaluations, we perform a fault analysis at the binary level. Accordingly, we use Instruction Set Architecture (ISA) fault models.

Simulation-based fault injection tool: Simulation-based fault injection tools inject modelled faults into simulated hardware. These tools assist the fault analysis by injecting automatically each fault according to the chosen fault models, thus ensuring exhaustiveness and repeatability. Our simulation-based fault injection tool CELTIC inject fault at ISA level, based on the target architecture emulated, using a micro-architectural simulator.

Fault injection settings: Fault injection settings describe the equipment configuration used during the fault exploitation. These parameters depend on the injection technique chosen. For example, the position on the chip focused by the laser, the peak power of the laser pulse or the laser pulse duration are possible fault injection settings specific to the LFI equipment. On the other hand, the injection delay, which is the duration between the start of the target application and the fault injection, is common to different fault injection techniques. The success of a fault attack relies on the experience of the evaluator in selecting the fault injection settings to induce effective faults.

C. Challenges of Multi-fault Attack Vulnerability Assessment

Due to the evolution of fault attack techniques and methods over the last decade, multi-fault attacks are now considered during security evaluation. However, fault analysis and fault exploitation are more difficult for multi-fault attacks than single-fault attacks.

Challenge n°1: Reducing the gap between the fault analysis and fault exploitation: Most of the time, fault analysis is performed with generic fault models (bit-set, bit-reset, etc.) without any consideration of the actual behaviour of the TOE during fault injection, resulting in:

- The fault attacks identified during fault analysis may not be achievable during the fault exploitation, as there is no evidence that the generic fault models will be reproducible. As a result, using generic fault models slow down the fault analysis and the fault exploitation.
- The fault analysis will miss potential implementation weaknesses. Recent target specific fault models [4], [5], [6], bypass software countermeasures in a single fault.
- This challenge becomes even more crucial for multi-fault attacks analysis as there will be more unnoticed vulnerabilities as well as more unexploitable attacks because of the combinatorial explosion of possible attack paths.

Challenge n°2: Considering combined fault attacks: A *combined fault attack* is a multi-fault attack with different fault models. Considering different fault models for multi-fault attacks significantly increases the coverage of the security evaluation by detecting previously unnoticed vulnerabilities. However, this makes the fault analysis even more difficult to perform due to combinatorial explosion of attack paths.

Challenge n°3: Improving fault injection settings selection: Selecting optimal fault injection settings is decisive for successful fault attacks. If manual selection of the best parameters is still possible with single-fault attacks, this approach is not possible with multi-fault attacks, because of the combinatorial explosion of possible fault injection settings to consider.

D. Contribution

Our main contribution is an end-to-end approach for multi-fault attack vulnerability assessment. The basic idea stems from a simple insight: restricting fault analysis to *target specific* fault models improve the overall security evaluation process by reducing the gap between the fault analysis and the fault exploitation. Also, *target specific* fault models mitigate the well-known combinatorial explosion problem of multi-fault attacks and combined fault attacks. The following overviews the important steps of the methodology.

Step 1: Tool-assisted Fault Models Inference: The first step of the methodology is an automated fault models inference. The fault models inferred are *target specific* to reduce the gap between the fault analysis and the fault exploitation (Challenge n°1).

Step 2: Tool-assisted Fault Analysis: Then, the fault analysis is automated with our simulation-based fault injection tool, CELTIC. With the *target specific* fault models previously inferred, this automated fault analysis can con-

sider both multi-fault and combined fault attacks (Challenge n°2).

Step 3: Tool-assisted Fault Exploitation: The last stage selects the best fault injection settings for the fault exploitation, for multi-fault and combined fault attacks, according to the previous results (Challenge n°3). This step is automated as well to speed up the overall process.

E. Related Work

Reducing the gap between fault analysis and fault exploitation has been initially addressed by Dureuil et al. [7]. They describe an end-to-end approach for vulnerability assessment based on probabilistic fault models inference from experimental results. With these probabilistic fault models and a simulation-based fault injection tool, they estimate the robustness of the TOE. However, one important step, the fault model inference, remains *manually* performed. Also, multi-fault attacks as well as combined fault attacks are not considered in their approach. Similarly, Given-Wilson et al. [8] et Rivière et al. [9] propose to combine hardware and software approaches. But they do not consider combined-fault attacks and they do not improve the selection of fault injection settings. Finally, Laurent et al. [10] use Register-Transfer Level (RTL) fault models, that closely match the processor faulty behaviours, to assist in the fault analysis at source code level by detecting previously unnoticed vulnerabilities. Nevertheless, they do not optimize the selection of fault injection settings and do not consider combined fault attacks.

On the other hand, Carpi et al.[11], Wu et al.[12] and Maldini et al.[13] propose fault injection settings search strategies respectively for Voltage Fault Injection (VFI), LFI and ElectroMagnetic Fault Injection (EMFI). Using Genetic Algorithms or Deep Learning, they find the most efficient fault injection settings, such as injection delays and other parameters specific to the fault injection technique, to speed up the fault exploitation. However, they do not infer fault models from experimental results to improve the fault analysis and they do not consider combined fault attacks. Then, Endo et al. [14] present a methodology to find multi-fault attacks that does not required a prior knowledge of the TOE. However, this approach is limited to identical fault models so combined fault attacks are not possible.

Although these works propose strategies to improve the fault exploitation or the fault analysis process, multi-faults attacks, and especially combined fault attacks, remain difficult to achieve due to the three challenges presented above. Table I summarizes the related works presented, according to these challenges. To the best of our knowledge, no tool-assisted methodology has been proposed to effectively perform practical combined fault attacks from the fault analysis to the fault exploitation. The following section overviews each step of our end-to-end approach for multi-fault attack vulnerability assessment.

Table I: Comparison of the related work according to the challenges.

	Challenge n°1	Challenge n°2	Challenge n°3
Our contribution	✓	✓	✓
[7]	✓	✗	✓
[8], [10], [9]	✓	✗	✗
[11], [12], [13], [14]	✗	✗	✓

III. Our Approach

A. Step 1: Tool-Assisted Fault Models Inference

With a characterization and a fault injection simulation performed simultaneously, we can infer fault models from experimental results. An overview of our fault models inference process is depicted in Figure 1. Then, the fault models and fault injection parameters are combined into Target Specific Fault Model (TSFM). These TSFM are transferable towards different samples of the same component, thereby reducing the initial time investment of the fault model inference step.

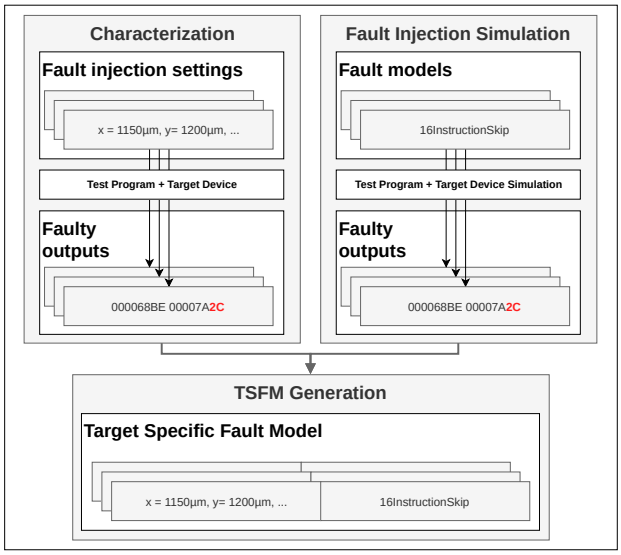


Figure 1: Overview of the tool-assisted fault model inference step

Characterization: The characterization is a common technique to help select the most efficient fault injection settings for the evaluated component. The Flash memory of the device must not be read or write protected, which is often the case during security evaluation. Instead of the target application, a program designed specifically for the characterization, referred to as the *test program*, is used to propagate most of the computational *errors* caused by the *faults*, in order to produce more often *faulty outputs*. A faulty output is detected by comparing the observed output with the expected output of the test program. During this step, a *grid search* (semi-exhaustive search on a predetermined

and progressively refined range of values) for fault injection settings is performed. The parameters that generate the most faulty outputs are the most efficient. However, it is difficult to diagnose the faults causing the observed faulty outputs. Most of the time, the step of fault models generalisation from the experimental results is performed manually, straight after the characterization, which can be difficult and tedious, especially when the data set of faulty outputs is large (above 10000 faulty outputs). To circumvent this issue, we perform a fault injection simulation to help fault models generalisation.

Fault Injection Simulation: We performed simultaneously a fault injection simulation with CELTIC. The goal is to quickly generate faulty outputs according to a set of state-of-the art fault models at ISA level and the *test program*. CELTIC injects faults according to the selected fault models into the *test program* and keeps track of each faulty outputs, detected by comparing the observed output with the expected output.

TSFM Generation: The faulty outputs acquired during fault injection simulation with CELTIC are automatically compared with those obtained during the characterization. When the same faulty output o_i is obtained during both characterization and simulation, we combine the fault injection settings s_i inducing the fault output o_i with the fault model m_i generating this same output o_i, into a triplet (s_i, m_i, o_i). This process is repeated for each identical faulty output through characterization and simulation; so as to generate a list of triplets (s_i, m_i, o_i). The pair (s_i, m_i) is referred to as Target Specific Fault Model. Similarly to the probabilistic fault models proposed by Dureuil et al. [7], we also estimate the probability of occurrence of each TSFM, that is the probability of each fault model according to the fault injection settings, $Pr(M = m|s)$, by dividing the number of faulty outputs according to the fault model m with the fault injection settings s by the total number of fault injected with fault injection settings s.

Example: Table II presents a list TSFM and the associated $Pr(M = m|s)$, according to positions focused by the laser on the evaluated component (a Cortex M4). In this example, inferred fault models are similar to those detailed by Dutertre et al. [4], referred to as the *InstructionSkip* fault model. This model represents a fault that skips one or more instructions. The prefix corresponds to the number of bytes skipped (Ex: *16InstructionSkip* $\leftrightarrow$ 16 bytes skipped $\leftrightarrow$ 4 32-bit instructions skipped). The faulty output coverage will be discussed in subsection IV-C.

Table II: Examples of Target Specific Fault models according to positions focused by the laser on the evaluated component (a Cortex M4), and the probability $Pr(M = m|s)$ associated.

Target Specific Fault Model		
Fault model	**Position**	$Pr(M = m\|s)$
16InstructionSkip	x = 1080 µm y = 1250 µm	0.65
	...	...
32InstructionSkip	x = 1060 µm y = 1240 µm	0.68
	...	...
48InstructionSkip	x = 1050 µm y = 1270 µm	0.72
	...	...
...	...	...

B. Step 2: Tool-Assisted Fault Analysis

The fault analysis is restricted to a limited set of TSFM. The selected TSFM are those whose probability $Pr(M = m|s)$ is greater than an arbitrary threshold. The threshold level influences both the fault analysis and the fault exploitation. The lower the threshold, the more exhaustive the fault analysis, but the longer it takes to simulate and the more false positives (unexploitable fault attacks) it generates.

In this step, the *target application*, that is the binary code embedded on the *target device*, is analysed. The fault analysis is automated with CELTIC to find exhaustively all the successful attacks with respect to the fault models. Multi-fault attacks with different and identical fault models are considered. At the end, CELTIC returns a list of all successful attacks, with respect to the oracle (a boolean condition to detect the successful attacks), and the selected fault models. For each successful attack, the injection delays from the start of the target application execution, expressed in clock cycle, as well as the fault models are automatically computed. As CELTIC does not simulate pipeline mechanisms, each clock cycle corresponds to one instruction fetched, decoded and executed. Algorithm 1 provides the CELTIC pseudocode to find successful combined fault attacks. Finally, CELTIC can be configured to simulate any state-of-the-art fault models at ISA level.

First, CELTIC generates the reference execution trace, that is an execution trace of the program without fault. The reference execution trace will be used during the fault generation to identify the potential injection sites and also to detect abnormal behaviour. Then the function `FindCFA` is called to find successful combined fault attacks. It takes as inputs an execution trace X of the target application, a set of fault models M, the attack order n (Ex: $n = 2$ for double-fault attacks), and the oracle `Oracle`. For each saved program state of the execution trace X^i and for each fault model m, CELTIC generates all the possible faults according to m at X^i. Next, CELTIC injects each fault f to produce a faulty execution trace X_{fault}. CELTIC saves X_{fault} if it is a successful attack according to the oracle, otherwise, CELTIC continues the process until all the faults have been injected.

Algorithm 1: FindCFA pseudocode

Function `FindCFA(`X, n`)` **is**
  Input: An execution trace X, the set of fault models M, the attack order n and the oracle `Oracle`.
  Result: The sucessful attacks S.
  $S \leftarrow \emptyset$;
  foreach $X^i \in X$ **do**
    foreach $m \in M$ **do**
      foreach $f \in$ `FindAllFaults(`X^i, m`)` **do**
        $X_{fault} \leftarrow$ `InjectFault(`X^i, f`)`;
        if `Oracle(`X_{fault}`)` **then**
          $S \leftarrow S \cup X_{fault}$;
        else if $n > 1$ **then**
          $n \leftarrow n - 1$;
          $S \leftarrow S \cup$ `FindCFA(`X_{fault}, n`)`;

Example: The case study is a VerifyPIN embedded on a Cortex-M4. The VerifyPIN has been hardened to mitigate single fault attacks. A VerifyPIN is a simple authentication program which verifies that the secret PIN entered is correct. It allows PIN entering for at most 3 times. The target microcontroller is the same as the previous step, so we can use, for the fault analysis, the fault models from Table II. The oracle is designed to detect a successful authentication with an invalid PIN without triggering a countermeasure. The VerifyPIN is robust to single-fault attacks, verified by experiments and using CELTIC, thus following attacks found are truly double-fault attacks. CELTIC find 417 successful double-fault attacks, including 292 combined fault attacks, as detailed in Table III. Figure 2a presents all the injection delays (in cycles) of the 97 successful combined fault attacks with *32InstructionSkip* and *48InstructionSkip* fault models.

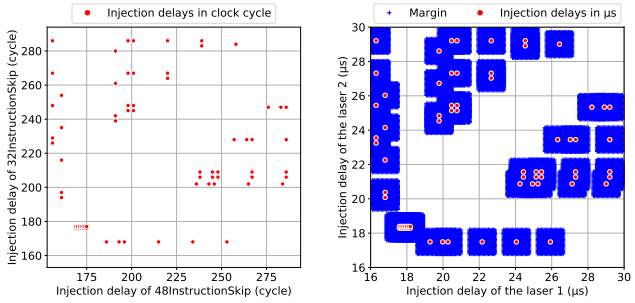
Table III: Number of successful double fault attacks found with CELTIC according to fault models.

2nd fault \ 1st fault	16Instruction Skip	32Instruction Skip	48Instruction Skip
16InstructionSkip	30	25	30
32InstructionSkip	62	46	42
48InstructionSkip	78	55	49

C. Step 3: Tool-Assisted Fault Exploitation

In this step, we take advantage from previous results in order to generate automatically the set of fault injection settings according to the successful attacks found with CELTIC.

Only the TSFM leading to successful simulated attacks are considered and, as fault analysis has been limited to the TSFM with the highest probability $Pr(M = m|s)$, optimal fault injection settings, specific to the FI technique (positions, power of the laser, etc.), are already known. Next, the injection delays of successful attacks simulated by CELTIC, expressed in clock cycles, are converted into microseconds with a linear relationship and then adding an arbitrary margin of error, as depicted in Figure 2b. The margin mitigates errors and inaccuracies in experimentation and simulation. Finally, injection delays and fault injection settings are combined to obtain a complete equipment configuration for all the successful fault attacks.



(a) (b)

Figure 2: Injection delays of successful double fault attacks, in clock cycles, with *32InstructionSkip* and *48InstructionSkip* fault models (Figure 2a). Injection delays of successful double fault attacks, in microseconds, with *32InstructionSkip* and *48InstructionSkip* fault models with a 10 clock cycles margin of error. (Figure 2b).

Example: Table IV is the generated list of equipment configurations to perform double combined fault attacks with two independent laser sources, from previous results. Here, two fault injection parameters vary at the same time: The injection delay and the chip position targeted by the laser. A 10 clock cycles margin has been chosen, thus from the 97 successful combined fault attacks with *32InstructionSkip* and *48InstructionSkip* fault models in Figure 2a, 42777 injection delays in microseconds has been generated for two independent laser sources in Figure 2b. The experimental validation of the generated configurations is detailed in the subsection IV-D.

Table IV: Generated list of equipment configurations for double-fault attacks with two independent laser sources according to previous results.

	Laser n°1		**Laser n°2**	
	Position	**Injection delay**	**Position**	**Injection delay**
Configuration n°1	x=1050µm y=1270µm	17.7 µs	x=1060µm y=1240µm	21.2 µs
Configuration n°2	x=1050µm y=1270µm	16.7 µs	x=1080µm y=1250µm	22.7 µs
...	...	...	...	...

IV. Experimental Results

This section presents the experimental results of our approach applied to a hardened VerifyPIN for a Cortex-M4. First, the experimental setup and the target device is described. Then, the faulty output coverage, an indicator of the performance of the fault models inference step, is detailed. Next, the generated configurations are validated using an exhaustive search as a reference. Finally, the performance of our end-to-end methodology is compared to other different approaches.

A. Experimental Setup

To demonstrate combined fault attacks in both spatial and time position, our LFI test bench consists in two similar and independent laser diodes emitting around 1 µm wavelength. The optical setup is depicted on Figure 3. Each laser beam can be adjusted independently from each other; we can tune the laser power, beam shape, and x, y spatial position. Altough the field of view of the microscope limits the fault models we can used, it is always possible to add an independent fiber laser mount on a robotized arm if necessary. Finally, the laser illumination is triggered by a simple electrical signal which can be sent at the proper time. Therefore, we can simultaneously inject two precise faults at different time and spatial positions. For this experiment, the laser pulse duration is set to one clock cycle ($\approx$ 70 ns) and does not vary, as the laser power.

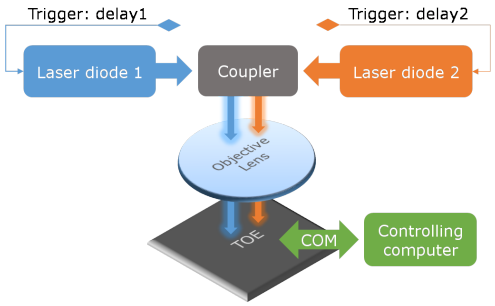


Figure 3: Schematic view of the double laser testbench

B. Target Device

The target device is an ARM Cortex-M4 32-bit microcontroller, opened from the backside to preform the LFI. This is the same device use in the example in step 1 (section III-A). This Harvard architecture microcontroller includes a three-stage pipeline, cache mechanisms and Thumb-2 instruction set compatibility. The Flash memory logic of the chip has been scanned, as shown in Figure 4. A grid search has been performed to find the most sensitive area in the Flash memory logic. Finally, the set P of targeted positions of the chip's Flash memory is a rectangle of 150 µm by 200 µm, with a scan step of 10 µm, for a total of 336 positions.

C. Performance of the Fault Model Inference

The characterization in step 1 (section III-A) has been performed using the set of positions P and varying the injection delays over 1 µs range. Figure 4 presents the chip sensitivity to LFI for this area. More than 50000 faults have been injected during 6 hours using the laser diode 1. During the characterization, more than 12000 faulty outputs has been generated, i.e. 24% of all faults injected. The rest of the faults injected (76%) cause fatal errors, that is the test program either terminates prematurely with a CPU exception or never terminates (timeout). The central region of the focused area (x = 1070 µm, y = 1250 µm) is the most sensitive to faults with more than 80% of the fault injections resulting in faulty outputs. Then, using our fault model inference method, 75% of the faulty outputs are covered with fault models (i.e. 18% of all the faults injected). The remaining faulty outputs (25%) have not been covered by the fault injection simulation. Table V sums up the previous results. The most common inferred fault models are 16InstructionSkip, 32InstructionSkip and 48InstructionSkip that represent more than 70% of all the faulty outputs. The probabilities of the other fault models given the fault injection settings are low (i.e. $Pr(M = m|s) < 0.01$), thus very difficult to predict, so they will not be considered during the fault analysis.

Table V: Overview of model inference results.

	Ratio (%)	**Count**
Total	100	51172
├ Fatal Errors (mute,timeout,etc.)	76	39048
└ Faulty Outputs	24	12124
├ Faulty Outputs not Covered	25	3173
└ Faulty Outputs Covered	**75**	8951
├ 16InstructionSkip	50	4497
├ 48InstructionSkip	34	3025
├ 32InstructionSkip	10	887
└ Other Fault Models	6	542

D. Validation of the Generated Configurations

To validate the generated configurations in step 3 (Table VI), injection delays of successful attacks found with exhaustive search (red) are compared to generated injection delays with our approach (blue), as depicted in Figure 5a. During the exhaustive search, the positions focused by the two lasers are fixed, as shown in Figure 5b, while the injection delays are those of the set D that ranges from 16 µs to 30 µs, corresponding to the start and the end of the execution of VerifyPIN, with a scan step of 10 ns, for a total of 1401 delays. Note the exhaustive search took one week to complete. The generated delays with our approach covers 894 out of 1744 possible injection delays leading to successful attacks found with exhaustive search (i.e. 51%). There is room for optimization, in particular to improve the

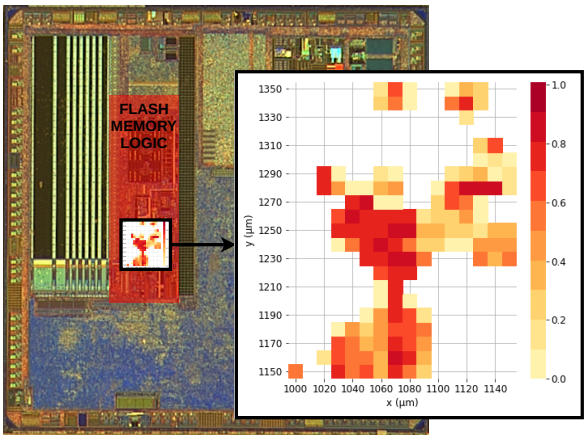
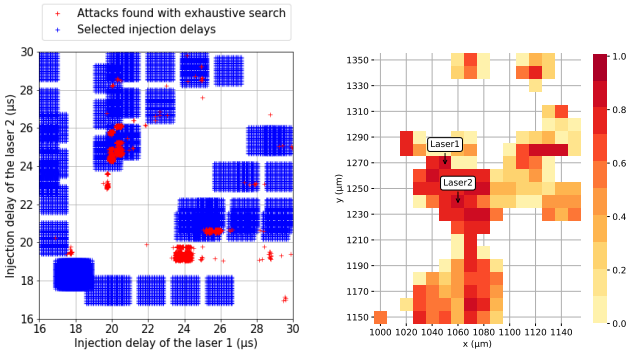


Figure 4: Backside view of our ARM Cortex-M4 chip, with the Flash logic memory highlighted in red, and the magnified view of the positions P. The sensitivity to faults for each positions P is given in the heatmap and ranges from 0 (not sensitive) to 1 (very sensitive).



(a) (b)

Figure 5: Attacks found with exhaustive search and the selected injection delays (Figure 5a). The positions focused by the two lasers are fixed (Figure 5b) while the injection delays range from 16 µs to 30 µs.

acquisition of the start and the end of the target application which is critical to convert accurately the injection delays of the simulated attacks.

E. Performance of our End-to-End Approach

1) Experiment Protocol: We evaluate the performance of three different methodologies, by comparing the time elapsed before a successful combined fault attack. The three approaches considered are as follows:

- Approach A: naive approach, the evaluator has no prior knowledge of the injection delays and the positions.
- Approach B: hybrid approach, the evaluator uses the sensitivity heatmap (Figure 4) to find positions but has no prior knowledge of the injection delays.
- Approach C: our approach, the evaluator has the generated list of equipment configurations (Table IV) with delays and positions to perform double combined fault attacks with two independent laser sources.

The target application is a VerifyPIN identical to the example in step 2 (section III-B). The goal is to be successfully authenticated with an incorrect PIN without triggering the countermeasures of the VerifyPIN in a minimum of trial, 100 times in a row. Each trial consists in performing one combined fault attack according to the set of delays and positions of each methods.

2) Results: Table VI details the experiment results between approaches **B** and **C**. Naive approach (**A**) did not pass the experiment within a reasonable time. Note that, with respect to the set P of targeted positions and the set of injection delays D, the exhaustive search of all possible combined fault attacks would take $P^2 \times D^2 \approx 1000$ years to complete, as an order of magnitude. Our approach (**C**) is three times faster on average than the hybrid approach (**B**). The VerifyPIN is a short program (execution trace around 200 clock cycles) and our approach will outperform even more the hybrid approach on a real program application (10,000 cycles or more).

Table VI: Performance comparison between the hybrid approach (**B**) and our approach (**C**).

	B	**C**
Avg Trials	1466	453
Avg Elapsed Time	13min58sec	4min18sec
Max Elapsed Time	2h35min59sec	31min4sec

V. Discussion

In this study, we assume faults do not depend from the executed code and only depend on the fault injection settings and also that a multi-fault is a combination of multiple single faults. This is theoretically true as soon as the injected faults affect a visible state of the microcontroller. In our test-case with ISA fault models, this means that the fault should affect the program state considered in our simulator (namely registers and memories). However the methodology can be applied to any kind of faults given the faulty output can be observed at step 1 and the simulation takes into account any specific hidden states. These assumptions have been verified in practice by other studies [6], [5], [15].

Then, we discuss different situations that may arise at each step. First, it may be difficult to generate faulty outputs during the characterization in step 1 because of hardware countermeasures. To mitigate this issue, it should be interesting either to follow smart search of fault injection settings [11], [12] instead of the classical grid search or to try different test programs. On the other hand, the faulty outputs

coverage may be too low to continue the approach, even considering all state-of-the-art fault models. The issue has not been solved yet, and still remains a topic for future work. A solution based on genetic algorithm to combine existing models into new ones could significantly improve the overall process. Then, CELTIC may not find any vulnerability in step 2, and thus the target application is likely to be robust to the TSFM considered. Finally, the generated configurations in step 3 may not result in successful attacks during fault exploitation. Indeed, as fault models are an abstraction and the program test used in step 1 is different from the target application, it is still possible that simulated fault attacks cannot be exploited. Nevertheless, we have never experienced this scenario so far.

VI. Conclusion

We presented an end-to-end approach for multi-fault attack vulnerability assessment. Using target specific fault models, we successfully exploit combined fault attacks, opening up a new way to find powerful attack paths. Target specific fault models are generated using a characterization and a fault injection simulation performed simultaneously, and they are transferable towards different samples of the same TOE, thereby reducing the initial time investment. Combined fault attacks are performed on a state-of-the-art laser bench using two laser sources, in order to control injection delays and positions focused on the chip independently. As future work, we will generalize this methodology to other devices such as secure components and we will consider real-world applications which should lead to more impressive results. Finally, combined fault attacks question the dependability of generic software countermeasures alone to protect a device against increasingly complex faults attacks.

Acknowledgement

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).

References

[1] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1997, pp. 37–51.

[2] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert, "Fault attacks on rsa with crt: Concrete results and practical countermeasures," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 260–275.

[3] C. H. Kim and J.-J. Quisquater, "Fault attacks for crt based rsa: New attacks, new results, and new countermeasures," in *IFIP International Workshop on Information Security Theory and Practices*. Springer, 2007, pp. 215–228.

[4] J.-M. Dutertre, T. Riom, O. Potin, and J.-B. Rigaud, "Experimental analysis of the laser-induced instruction skip fault model," in *Nordic Conference on Secure IT Systems*. Springer, 2019, pp. 221–237.

[5] L. Riviere, Z. Najm, P. Rauzy, J.-L. Danger, J. Bringer, and L. Sauvage, "High precision fault injections on the instruction cache of armv7-m architectures," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2015, pp. 62–67.

[6] B. Colombier, A. Menu, J.-M. Dutertre, P.-A. Moëllic, J.-B. Rigaud, and J.-L. Danger, "Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller." *IACR Cryptology ePrint Archive*, vol. 2018, p. 1042, 2018.

[7] L. Dureuil, M.-L. Potet, P. de Choudens, C. Dumas, and J. Clédière, "From code review to fault injection attacks: Filling the gap using fault model inference," in *International conference on smart card research and advanced applications*. Springer, 2015, pp. 107–124.

[8] T. Given-Wilson, N. Jafri, and A. Legay, "Bridging software-based and hardware-based fault injection vulnerability detection," 2018.

[9] L. Rivière, M.-L. Potet, T.-H. Le, J. Bringer, H. Chabanne, and M. Puys, "Combining high-level and low-level approaches to evaluate software implementations robustness against multiple fault injection attacks," in *International Symposium on Foundations and Practice of Security*. Springer, 2014, pp. 92–111.

[10] J. Laurent, V. Beroulle, C. Deleuze, F. Pebay-Peyroula, and A. Papadimitriou, "Cross-layer analysis of software fault models and countermeasures against hardware fault attacks in a risc-v processor," *Microprocessors and Microsystems*, vol. 71, p. 102862, 2019.

[11] R. B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub, "Glitch it if you can: parameter search strategies for successful fault injection," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2013, pp. 236–252.

[12] L. Wu, G. Ribera, N. Beringuier-Boher, and S. Picek, "A fast characterization method for semi-invasive fault injection attacks," in *Cryptographers' Track at the RSA Conference*. Springer, 2020, pp. 146–170.

[13] A. Maldini, N. Samwel, S. Picek, and L. Batina, "Optimizing electromagnetic fault injection with genetic algorithms," in *Automated Methods in Cryptographic Fault Analysis*. Springer, 2019, pp. 281–300.

[14] S. Endo, N. Homma, Y.-i. Hayashi, J. Takahashi, H. Fuji, and T. Aoki, "A multiple-fault injection attack by adaptive timing control under black-box conditions and a countermeasure," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 214–228.

[15] J. Laurent, V. Beroulle, C. Deleuze, F. Pebay-Peyroula, and A. Papadimitriou, "On the importance of analysing microarchitecture for accurate software fault models," in *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, 2018, pp. 561–564.