IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 4, pp. 425–450. DOI:10.46586/tches.v2024.i4.425-450

Switching Off your Device Does Not Protect Against Fault Attacks

Paul Grandamme[1,2], Pierre-Antoine Tissot[1], Lilian Bossuet[1], Jean-Max Dutertre[2], Brice Colombier[1] and Vincent Grosso[1]

[1] Université Jean Monnet Saint-Etienne, CNRS, Institut d Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France
paul.grandamme@univ-st-etienne.fr,pierre.antoine.tissot@univ-st-etienne.fr,lilian.bossuet@univ-st-etienne.fr,b.colombier@univ-st-etienne.fr,vincent.grosso@univ-st-etienne.fr

[2] Mines Saint-Etienne CEA, Leti, Centre CMP, F-13541, GARDANNE, France,
p.grandamme@emse.fr,dutertre@emse.fr

Abstract. Physical attacks, and among them fault injection attacks, are a significant threat to the security of embedded systems. Among the means of fault injection, laser has the significant advantage of being extremely spatially accurate. Numerous state-of-the-art studies have investigated the use of lasers to inject faults into a target at run-time. However, the high precision of laser fault injection comes with requirements on the knowledge of the implementation and exact execution time of the victim code. The main contribution of this work is the demonstration on experimental basis that it is also possible to perform laser fault injection on an unpowered device. Specifically, we targeted the Flash non-volatile memory of a 32-bit microcontroller. The advantage of this new attack path is that it does not require any synchronisation between the victim and the attacker. We provide an experimental characterization of this phenomenon with a description of the fault model from the physical level up to the software level. Finally, we applied these results to carry out a persistent fault analysis on a 128-bit AES with a particularly realistic attacker model which reinforces the interest of the PFA.

Keywords: Fault attack · Laser injection · Unpowered devices · Persistent fault analysis · Flash memory

1 Introduction

Since the late 1990s, it is known that electronic devices are vulnerable to physical attacks [Koc96, BDL97]. Physical attacks can be classified as either passive or active. Passive attacks are mainly based on the analysis of side-channel leakage in the power consumption or electromagnetic emanation of a targeted device. Active attacks aim to exploit the results of a program execution while it is being disrupted by injected faults. In this paper, only active attacks are considered.

Fault injections can be classified as either local or global. Global fault injections aim to disturb large blocks or the entire component under attack. Among these attacks, one can find voltage [ZDCR14, O'F16, BFP19] or clock [ADN+10, BGV11] glitches. Local fault injection techniques aim to change the device behavior by affecting a specific part of the device. For instance, memories, such as SRAM or Flash, the central processing unit or any peripherals can be individually targeted by faults. The two most common ways to inject local faults are electromagnetic or laser fault injection. Electromagnetic fault injection was first proposed in [SSAQ02], experimentally demonstrated in [SH07] and

Received: 2024-04-15 Accepted: 2024-06-15 Published: 2024-09-05

later improved in [DDRT12] with a strong temporal and spatial accuracy. The first effect of light on microcontrollers was shown by Skorobogatov in 2002 by exposing circuits to flash-lights and laser beams [SA02]. In 2009, he also demonstrated that it is possible to erase bits with a 650 nm wavelength laser by heating memory cells [Sko09].

When discussing fault injection, a fault model is defined as an abstraction framework upon which an attacker can build attack scenarios. On *powered* devices, fault models associated with laser fault injection are well understood from the algorithmic down to the physical level [SA02, ADM+10, RSDT13, SRD+13, CBD+15, SBHS15, DBC+18, CMD+19, MDC+20, CGV+21].

With a few notable exceptions [SHP09, GBD23] all the attacks mentioned were carried out on powered devices. However, in these cases, on-board sensors can detect the fault injection and react if necessary because the device is powered. For instance, in their study [NRV+06], Neto *et al.* used sensors to detect anomalous transient currents induced in the bulk of an integrated circuit during laser fault injection. In [BGH+14], Beringuier-Boher *et al.* propose to combine known digital and analog countermeasures against voltage glitches in order to leverage the benefits of each solution in mixed-signal systems. In [ERM16], El-Baze *et al.* propose a new fully digital detector against electromagnetic injection based on propagation delay violation. There are other techniques besides sensors. For example, we can use control-flow integrity techniques to detect suspicious behaviour. Although effective, these countermeasures are considered *active* as they assume the device is powered on.

Conversely, in this paper, we demonstrate the use of laser fault injection on *unpowered* devices. Specifically, we target the Flash memory of unpowered 32-bit microcontrollers. It was observed that individual bits of the memory can be set permanently and with great spatial accuracy. This vulnerability could enable an attacker to carry out powerful attacks, or alter stored firmware prior to program initiation for example.

Contributions

Our contributions are as follows:

- The main contribution is to demonstrate, for the first time, the possibility of performing laser fault injection on unpowered devices with the intent to exploit the injected fault when the target is switched on again. We take advantage of the unmatched spatial accuracy of laser fault injection. Specifically, faults are obtained by repeating laser shots in the Flash memory of unpowered devices. This enables us to relax the attacker model typically associated with laser fault injection when performed on powered devices.

- We characterize an unidirectional bitset fault model using this method. After a complete experimental characterization, we present a physical fault model for laser-induced faults in NOR Flash memory cells of unpowered devices. This fault is then lifted from the physical level up to the software level.

- We leverage this new laser fault injection method to perform a complete reverse engineering of the mapping between the logic and physical addresses of the Flash memory. The organization of the Flash memory can be retrieved at both the page and bit level.

- We exploit this fault model in the PFA setting to recover a 128-bit AES secret key. This is accomplished by using a realistic attacker model, which emphasizes the PFA as an increasingly relevant threat model.

Outline

The article is structured as follows. Section 2 provides an overview of laser fault injection, attacks on unpowered devices and recalls the mode of operation of Flash memory and floating gate transistors. Section 3 describes the fault model observed experimentally, at several levels of abstraction. Section 4 provides details on the experimental setup, the methodology employed and the analysis of experimental results according to the fault model. Section 5 exploits this fault model to recover a 128-bit AES key in the persistent fault analysis framework. Section 6 presents a discussion on the proposed attack. Finally, Section 7 concludes this article.

2 Related works

Definitions and notations

The following conventions and notations are used in the rest of this paper.

A *bitset* is defined as a transition from a logic `0` to a logic `1` after a fault injection. Conversely, a *bitreset* is defined as a transition from a logic `1` to a logic `0` after the fault injection. Finally, a *bitflip* is defined as a transition from a logic `0` (respectively `1`) to a logic `1` (respectively `0`) after the fault injection regardless of the initial value.

In Flash memory, data is stored in 32-bit words or 4-byte words. The 256 32-bit words stored in a page are referred to as w_k ($k \in [\![0, 255]\!]$). $b_{i,j}$ is the i^{th} bit of the j^{th} 32-bit word of the page and $S_{i,j}$ is the j^{th} bit of the i^{th} value of the S-box. The j^{th} byte of a ciphertext is referred to as c_j.

2.1 Laser fault injection

In 2002, Skorobogatov and Anderson were the first to use lasers to inject transient faults into microcontroller memories for attack purposes. Indeed, in [SA02], individual bits were set or reset inside the SRAM of a microcontroller. This breakthrough paved the way for further studies, including the development of a complete bitset/bitreset fault model in SRAM memories as described in [RSDT13]. This model was then successfully used to mount a key recovery attack on an implementation of AES using differential fault attack (DFA) [ADM+10]. In 2016, Selmke *et al.* also obtained good results by attacking a target from the backside with an infrared laser beam [SBHS15].

Further work has shown that D flip-flops can also be faulted by performing laser injection at the 40 nm technology node [CBD+15]. The study revealed that targeting certain parts of the DFF can result in a bitset fault model, while targeting other parts can result in a bitreset fault model. This fault model was further refined in 2018 with a focus on the 28 nm technology node [DBC+18].

Furthermore, other work has shown that laser fault injection can be used to corrupt Flash memories [CMD+19, MDC+20, CGV+21]. Indeed, it is possible to corrupt the device behaviour by performing laser fault injection during the read operation. These faults occur during the fetch process, hence the stored value remains unaltered. Laser fault injection in Flash memory exhibits a unidirectional bitset fault model. In [VDDM21], the authors also managed to perform permanent laser fault injection in a Flash memory during the write operation. They obtain a bitreset fault model with a single bit precision. The written data is permanently corrupted until the device is reprogrammed.

In all the cases described above, laser fault injection offer a sound fault model and enables an attacker to be extremely precise in terms of time and location. Agoyan *et al.* [ADM+10] were able to recover the AES key by performing a DFA using laser fault injection, which cannot be achieved with other fault injection tools such as clock or voltage glitches Therefore, laser fault injection is the most precise method for injecting faults

in terms of time and space. This precision enables advanced attacks on cryptographic applications. Other studies have also demonstrated that utilising multi-spots laser fault injection sources can enable advanced attacks. In [CGV+21], the authors describe the possibilty to use four laser spot simultaneously. Each spot can be controlled in time and position independently.

The sole limitation of all this work is that the device must be powered. Indeed, all fault models obtained in the previously described works are only valid if charges created by the laser beam through the photoelectric effect are separated by the electric fields within the device as described in [SLD+12, SGS+13]. In this case, a current is created and a fault may be obtained. However, if the device is unpowered, then charges recombine quickly and do not induce any faults.

2.2 Attacks on unpowered devices

Only a few studies have been conducted on attacks involving fault injection on unpowered devices.

In 2009, Schmidt *et al.* were able to retrieve an AES key by altering the AES S-box of an unpowered device. To perform the fault injection, the EEPROM memory was exposed to UV light. UV-resistant ink was used to protect the program code from the UV irradiation and target the S-box specifically. However, specific data could not be targeted due to the ink deposition's lack of spatial accuracy. Cryptanalysis is performed using differential analysis and is not developed to exploit persistent faults.

Other works deal with circuit editing using a focused ion beam on an unpowered device. In [HNT+13], hardware security functions and countermeasures were permanently disabled by modifying the circuit in a non-reversible manner. This method has the significant drawbacks as it is expensive, destructive, and difficult to set up.

2.3 Non-volatile memory architecture

The floating gate transistor is the unit component of Flash memories. This subsection describes the floating gate transistors that are used to store one bit of information and the operation of Flash non-volatile memory. This subsection only gives the necessary information to understand the rest of the article, the interested reader can find more information in [CGOZ99, CMN05].

2.3.1 Floating gate transistor

From an electrical perspective, each bit of information stored in a Flash memory is an electrical charge. This charge is stored in a floating gate transistor. A charge-storage element, called the floating gate, is located into the oxide layer between the control gate and the transistor's channel. The floating gate is electrically isolated from the rest of the structure by the oxide. The structure of a floating gate transistor is shown in Figure 1.

(a) Discharged cell. **(b)** Charged cell.

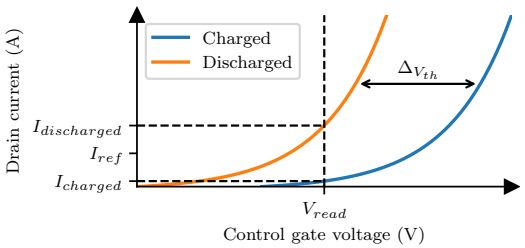
Figure 1: Cross-sectional view of a floating gate transistor.

When charges are stored in the floating gate, the field created by the carriers causes the energy bands in the oxide to bend. The carriers stored in the floating gate cause a shift in the current-voltage characteristics of the cell, as shown in Figure 2a. For a given V_{read} voltage applied to the control gate, the current drawn by the transistor is lower (respectively higher) if the floating gate is charged (respectively discharged). Thus, the logical state of a floating gate transistor is determined by the charges present in the floating gate, which affect its threshold voltage.

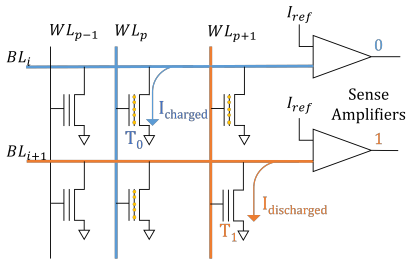
To read the content of a memory cell, the control gate is biased at a fixed voltage of V_{read} which is set between the threshold voltage of a charged and an uncharged floating gate transistor. The current drawn by the memory cell is compared to a reference value I_{ref} by a sense amplifier. The read mechanism is illustrated in Figure 2b.

In order to read the value stored in transistor T_0 which is a charged floating gate transistor, the wordline WL_p and the bitline BL_i are selected. The floating gate transistor will then draw a current $I_{charged}$, which is lower than I_{ref} and will produce a logic `0` at the output of the sense amplifier. This scenario is depicted in blue in Figure 2.

In order to read the value stored in transistor T_1 which is a discharged floating gate transistor, the wordline WL_{p+1} and the bitline BL_{i+1} are selected. The floating gate transistor will then draw a current $I_{discharged}$, which is greater than I_{ref} and will produce a logic `1` at the output of the sense amplifier. This scenario is depicted in orange in Figure 2.



(a) I-V characteristics of a charged and discharged floating gate transistor.



(b) Read operation of a Flash memory

Figure 2: Read operation and I-V characteristics of a charged and discharged floating gate transistor of a Flash memory.

2.3.2 Flash memory

Microcontrollers use non-volatile memories, such as NOR Flash or EEPROM, to store permanent data. Both Flash and EEPROM memories consist of Flash memory cells, control logic, X and Y decoders, latches, read sense amplifiers, and charge pumps. The high-level architecture of these memories is shown in Figure 3.

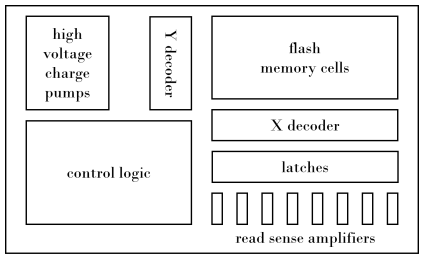


Figure 3: Usual organization of Flash memories [Sko10].

The control logic part manages the mapping between logical addresses and physical

addresses and then selects a group of cells by controlling the X and Y decoders. The sense amplifiers are used to compare the current drawn by the cell with a reference current to determine whether the selected cell stores a logical `0` or `1`. Typically, the memory contains as many sense amplifiers as the width of the data bus. For instance, in order to fetch one 32-bit word, one wordline (WL) and 32 bitlines (BL) have to be selected. The Flash memory is a very large and easily identifiable structure. This makes it easy for an attacker to locate the Flash memory using an infrared image of the backside of the chip.

3 Fault model

To describe the impact of laser fault injection on a target, it is necessary to define a fault model. This description can be done at various levels of abstraction, including physical, logic and software. It is essential to have a thorough understanding of the effects of laser fault injection to propose effective countermeasures. Otherwise, the countermeasure could be too costly or, in the worst case, ineffective. Among the various existing fault injection means, the laser is the one where we best know the fault model at these different levels.

In [SRD+13], an electrical model was proposed to describe the interaction between a laser beam and a *powered* electronic circuit . According to this model, laser illumination in reverse-biased PN junctions of CMOS gates generates a photoelectric current. This model is usually applied to drain-bulk junctions of floating gate transistors whose columns have been biased at a fixed potential by the column decoder in order to fetch data from non-volatile memory. This effect can only happen in powered devices because of the presence of the electric field in the oxide. Without the latter, charge carriers which were induced will recombine very quickly and nothing happens. Conversely, the electric field quickly separates the charge carriers and an electrical current appears. This effect is not applicable in our case as the target is *unpowered*.

3.1 Physical level

The physical level refers to the lowest possible level of abstraction, closest to transistors and electrical charges. It is known that the energy supplied to the circuit by the laser beam causes the temperature inside the component to rise locally [San11]. This thermal energy is applied to the charges stored in the floating gates, allowing them to escape from the potential well. Therefore, heating induced by the laser beam depletes the charges stored in the floating gate. If enough charges are removed, the logic state of the floating gate transistor can be altered.

The intensity of the laser beam in the focal plane of the lens follows a Gaussian radial distribution as described in [BMPM13] and is given in Equation 1, with ω_0 being the dispersion of the Gaussian distribution in the focal plane of the objective.

$$I(r) = I_0 \cdot e^{-\frac{2r^2}{\omega_0^2}} \tag{1}$$

The ω_0 parameter depends on the wavelength λ of the laser and the numerical aperture NA of the objective with the relation given in Equation 2.

$$\omega_0 = \frac{2\lambda}{\pi \times NA} \tag{2}$$

The laser spot size is defined by the FWHM (Full-Width-at-Half-Maximum) criterion, *i.e.* the diameter of the perimeter where the beam intensity is half the maximum intensity. Equation 3 provides the relationship between the spot size d_0 and the parameter ω_0.

$$d_0 = \omega_0 \sqrt{\frac{\ln 2}{2}} \tag{3}$$

The laser fault injection setup used in the experiments is described in more details in Subsection 4.1. The laser wavelength is $\lambda = 1{,}064\,\text{nm}$ with a numerical aperture $NA = 0.16$. A numerical simulation was conducted to generate the heatmap depicted in Figure 4. Each black rectangle corresponds approximately to a Flash memory cell. The laser spot size, represented by the white circle, is determined by the FWHM criterion. The scale is expressed as a percentage of I_0 which is the maximum intensity present at the center of the spot. Alternative laser sources with varying wavelengths could have been used during the experiments. A source commonly used for laser fault injection was selected. For instance, a source with a wavelength of 1,300 nm, which is considered "purely thermal", may have been chosen.

The hardware target, as described in Subsection 4.2, embeds 128 kB of Flash memory. The latter is approximately 1,400 µm long and 600 µm wide. The target's Flash memory is divided into 2,048 columns and 512 rows, as stated by [Men21].

This information allows us to estimate the size of a Flash memory cell:

$$width = \frac{1400}{2048} \approx 0.68\,\mu\text{m} \qquad length = \frac{600}{512} \approx 1.17\,\mu\text{m}.$$

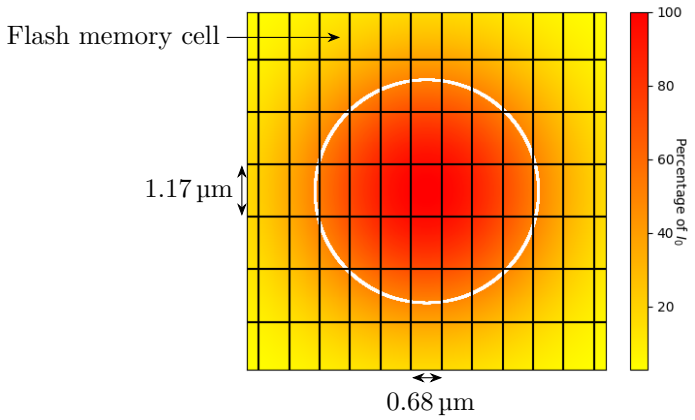


Figure 4: Heatmap induced by the laser exposition (numerical simulation with $\lambda = 1{,}064\,\text{nm}$ and $NA = 0.16$).

On the one hand, one can see that the laser spot is larger than a Flash memory cell, indicating that multiple cells are being heated. On the other hand, the laser energy is concentrated in the centre of the spot and decreases exponentially as the square of the radius increases. The number of injected faults is determined by the number of sufficiently heated transistors. The latter depends on the parameter I_0 which is unknown.

Laser exposure may cause a reduced number of floating gate transistors to discharge, resulting in fault injection on unpowered targets. This phenomenon can ultimately corrupt the stored value, as explained in the following subsection.

3.2 Logic level

From the physical level fault model described previously, it is known that heating the device with a laser beam can deplete the charges stored in the floating gate transistor. Therefore, it is important to note that only the *discharge* of floating gate transistors is possible, not the reverse. This leads to a unidirectional fault model. With this fault model, a logic `0` stored in the Flash memory may become faulty and change to a logic `1` if enough

charges are removed. However, a logic 1 stored in the Flash memory will always be read as 1 since it is not possible to inject charges into a floating gate transistor using a laser beam.

To conclude, the laser fault injection on an unpowered target results in a unidirectional bitset fault model, following the most common convention. The convention chosen by the manufacturer of the target considered can determine whether it is the opposite or not.

3.3 Software level

As the Flash memory is non-volatile, it is used to store any permanent data. For instance, the firmware is always stored in the Flash memory to avoid the need for device programming at each power cycle. Additionally, constants, access rights, and cryptographic keys may also be stored in Flash memory.

Code corruption

Instructions executed by the device are stored in the Flash memory. As the target used in this study has an ARM Cortex M3 core the description of each instruction can be found in [ARM12]. An instruction consists of an opcode and may include a source register, a destination register and an immediate value. By faulting the opcode, we can alter the type of instruction. By faulting one of the other elements, the result of the operation can be altered. These modifications disrupt the program's execution. Figure 5 shows instances of instruction corruptions that were obtained through laser fault injection in the Flash memory of a powered device [CMD+19]. The first case represents an immediate value corruption (`0x0004` instead of `0x0000`), the second one a destination register corruption (`R1` instead of `R0`) and the last one an opcode corruption (`MOVT`[1] instead of `MOVW`[2]).

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
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	0	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
`MOVW,R0,4`																															
1	1	1	1	0	0	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
`MOVW,R1,0`																															
1	1	1	1	0	0	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
`MOVT,R0,0`																															
1	1	1	1	0	0	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

Figure 5: Examples of possible corruptions on a `MOVW` instruction [CMD+19].

Data corruption

In addition to firmware, the Flash memory can also store other data such as cryptographic keys, access rights, passwords, and other constants. For instance, in [VDDM21] Viera *et al.* demonstrate how permanent laser fault injection in a Flash memory can be used to force a password to a known value.

More specifically, in the case where AES is implemented, the S-box will be stored in the Flash memory. This is the scenario chosen in this study and described in Section 5.

[1] Moves 16-bit immediate value to top halfword. Bottom halfword unchanged.
[2] Moves immediate value to the destination register.

4 Experiments

This section describes the experimental setup including the laser fault injection setup and the hardware targets used in our experiments.

4.1 Laser fault injection setup

The laser fault injection setup used in our experiments is a nanosecond near-infrared laser source and an optical system which focuses the laser beam inside the silicon substrate. The device is mounted on a micrometric XYZ-positioning table.

The laser source emits a beam with a wavelength of 1,064 nm and a maximum power of 3 W. The laser beam can penetrate hundreds of micrometers inside the substrate through the backside of the target [BMPM13]. The pulse duration can be set between 50 ns and 1 s. The focusing system allows to obtain a laser spot of 5 µm in diameter using a 20x magnifying lens. With the x20 magnification, the focusing system has an optical transmission coefficient of 57 % and a numerical aperture of $NA = 0.16$.

To prepare for laser fault injection, it is common practice to obtain an infrared image of the target. This requires illuminating the backside of the target with an infrared lamp (λ=1,064 nm). The metallization layers reflect the light, which is then captured by a CCD camera. Figure 6 on the right shows the resulting image.

4.2 Hardware targets

The hardware target is a 32-bit microcontroller, the STM32F1, manufactured by STMicroelectronics. It is placed on a custom board designed for the ChipWhisperer platform [OC14]. A picture and an infrared zoom-in of the hardware target can be seen in Figure 6. The procedure described in [LVD+22] was followed to prepare the target for laser fault injection. The first step involves removing the package using a milling machine. For backside access, the operator can drill through the silicon. The second step is to polish the silicon surface to enable observation with infrared light and attacks through the die. Other methods, such as chemical or laser resin removal, also exist.

The microcontroller embeds an ARM Cortex-M3 core and 128 kB of Flash memory. The Flash memory is divided into 128 pages of 1 kB each [STM10]. For this device, the *erased* state of the Flash memory is stored as the value `0xFFFFFFFF` at 32-bit word level. Therefore, erasing an elementary bit cell of Flash memory (or floating gate transistor) sets the data bit stored in it to `1`, while programming sets its value to `0`. The Flash memory contains 2,048 bitlines and 512 wordlines. The results described in the next subsection have been obtained on four different devices.

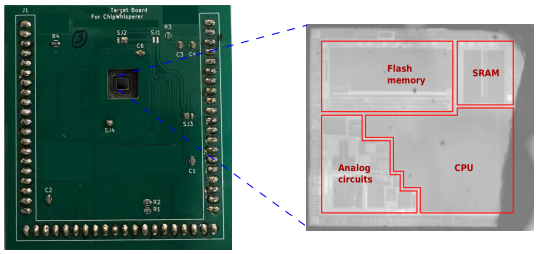


Figure 6: Picture of the board (left) and infrared image (right) of the hardware target.

4.3 Experimental results

Laser fault injection characterization As explained in Subsection 4.2, the *erased* state is stored as the value `0xFFFFFFFF` at 32-bit word level. Our main assumption regarding laser fault injection in a Flash memory of an unpowered target was that it would allow some of its bit cells to be erased by heating them due to electrons leaving their floating gates thanks to the thermal energy provided by laser exposure. That is why we have carried out our first tests with the Flash memory filled with zeroes, for the purpose of a first experimental validation (erasing a bit cell sets its stored value to one).

The X and Y coordinates of the Flash were scanned with the laser beam in order to obtain a mapping. The target is unpowered before each series of laser shots and powered after in order to read the content of the Flash memory. As we were looking for a cumulative effect, the discharge of floating gates, many laser shots must have been performed at each position. We chose to perform series of 1,000 shots with the maximum pulse duration of 0.9 s at a frequency of 1 Hz. The laser source does not have a continuous mode, so a longer pulse cannot be set. As the illumination time for each position is relatively long, it is not reasonable to perform a mapping with a fine spatial step. Algorithm 1 shows the experimental procedure.

Algorithm 1 Mapping of injected faults.

Require: $X_{min}, X_{max}, X_{step}, Y_{min}, Y_{max}, Y_{step}$
  for $x \in$ `range`$(X_{min}, X_{max}, X_{step})$ **do**
    for $y \in$ `range`$(Y_{min}, Y_{max}, Y_{step})$ **do**
      reset target memory
      do
        move laser to (x,y)
        power target off
        for $i \in [0, \ldots, 999]$ **do**
          laser shot
        power target on
        dump target memory
      while #faults == 0
      mapping$[x][y]$ = #faults
  return mapping$[x][y]$

In laser fault injection, the procedure typically involves starting with a low power and gradually increasing it until the first effects are observed. The first effects appear at a power of 1 W at the end of the optical fiber and before the microscope. Bitsets were obtained for each position, corresponding to an erasure of the floating gate transistors. Results are shown in Figure 7. By reading the Flash memory content and analysing the file after each series of laser shots, we can obtain the address and the value of each fault.

The distribution of the number of faults injected is described in Figure 8. In many cases, multiple faults were obtained. The red line represents the mean value: 2.2 bits are faulty on average. In 33% of cases, the fault obtained is a single bit fault. When multiple bits are faulty, they are always stored at adjacent physical positions in the Flash memory. This phenomenon is explained by the fact that the laser spot is larger than a floating gate transistor as shown in Subsection 3.1. Although an average number of 2.2 faulty bits may seem like a limitation to obtain single bit faults, it is important to note that these faults were obtained with the Flash memory filled with zeroes. In a real case scenario, however, it can be estimated that half the bits of the firmware will already be set to 1. Therefore, a single-bit bitset fault model can definitely be obtained in practice, as demonstrated in Section 5.

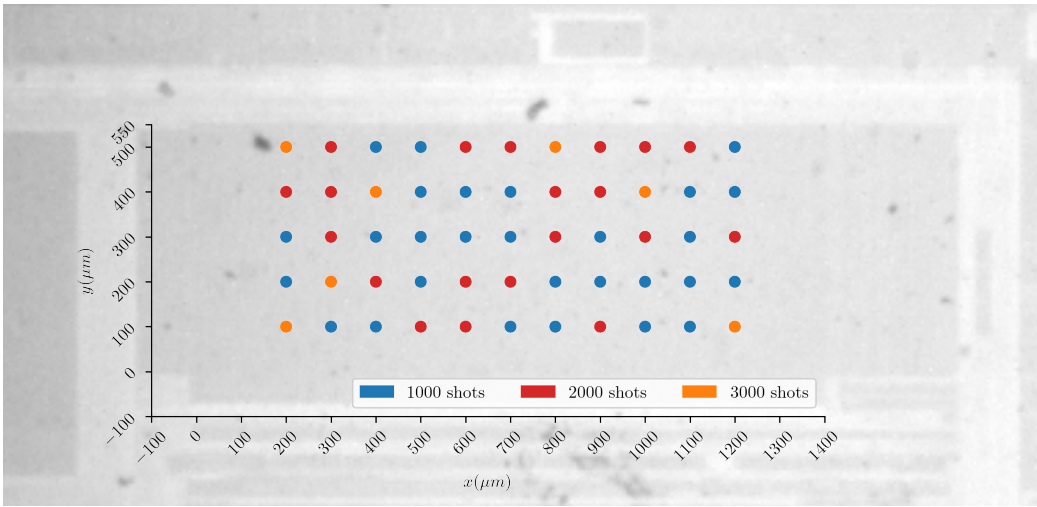


Figure 7: Mapping of obtained faults. $P_{laser} = 1\,\mathrm{W}, f_{laser} = 1\,\mathrm{Hz}, T_{pulse} = 0.9\,\mathrm{s}$

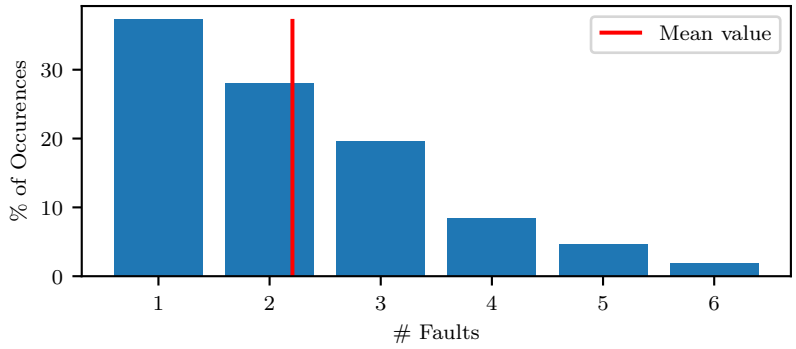


Figure 8: Experimental distribution of the number of injected faults.

The number of laser shots required to inject a fault at a given position may vary slightly from one device to another. This might be due to aging effects, previous attacks or uses which are different from one device to another because floating gate transistors have limited lifetimes in terms of erasing and programming. It is worth noting that no sign of degradation on the hardware targets were observed during the experiments. Therefore, there is no risk of shooting too many times and inducing a destructive fault in the device. From an attacker point of view, there is no need to find the good number of laser shots to perform: it is possible to perform the maximum number of laser shots without any adverse effects. Therefore, the number of laser shots is not an experimental parameter to explore. Furthermore, this method has the advantage of being undetectable by any sensors since the target is unpowered.

Reverse-engineering of the memory mapping Based on the previous experiments, it is possible to fully reverse engineer the mapping between logical addresses and physical locations of data in the Flash memory. In Figure 9 we can see the physical organization

of the 128 pages that make up the Flash memory. From the programming manual of the device [STM10], we can obtain the range of addresses contained in each page. For example, page 0 ranges from `0x08000000` to `0x080003FF`.

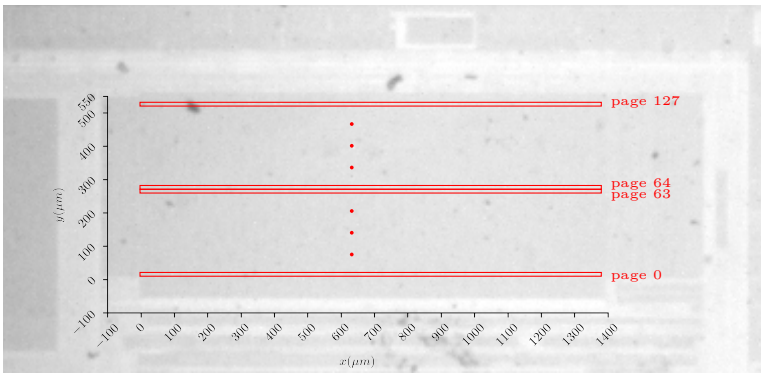


Figure 9: Reverse engineering of the Flash memory mapping : page-level

Reverse engineering can also be performed at a bit level. In Figure 10 we can see that bits are stored in columns by index. The first column contains all bits at index 31, the second column contains all bits at index 30, and so on until the bits at index 0 which are stored in the last column.

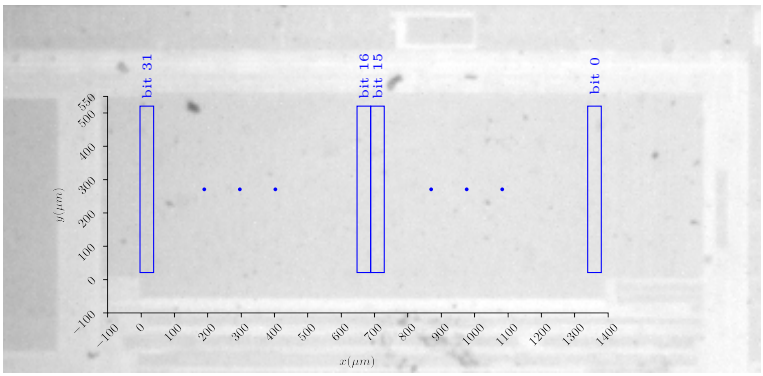


Figure 10: Reverse engineering of the Flash memory mapping: bit-level.

To conclude, the organization of words and bits within a memory page can be seen in Figure 11. Reverse-engineering the Flash memory mapping enables an attacker to target a specific bit in the Flash memory by knowing its address.

5 Practical application : Persistent Fault Analysis

Previous sections have demonstrated that laser fault injection can be used to inject persistent faults in the floating gate transistors of unpowered Flash memories. A fault model for a single bitset has been described, whereby the targeted floating gate transistor

bit 31				bit 30					bit 0			
$b_{31,0}$	$b_{31,1}$	⋯	$b_{31,64}$	$b_{30,0}$	$b_{30,1}$	⋯	$b_{30,64}$	⋯	$b_{0,0}$	$b_{0,1}$	⋯	$b_{0,64}$
$b_{31,65}$	⋯	⋯	$b_{31,127}$	$b_{30,65}$	⋯	⋯	$b_{30,127}$		$b_{0,65}$	⋯	⋯	$b_{0,127}$
$b_{31,128}$	⋯	⋯	$b_{31,191}$	$b_{30,128}$	⋯	⋯	$b_{30,191}$		$b_{0,128}$	⋯	⋯	$b_{0,191}$
$b_{31,192}$	⋯	⋯	$b_{31,255}$	$b_{30,192}$	⋯	⋯	$b_{30,255}$		$b_{0,192}$	⋯	⋯	$b_{0,255}$

Figure 11: Physical position of words and bits within a Flash memory page. $b_{i,j}$ is the i^{th} bit of the j^{th} 32-bit word of a page.

is erased. Most cryptanalysis techniques fault injection are based on transcient faults. However, the injection of persistent faults can still be effectively exploited through the Persistent Fault Analysis [ZLZ+18], a recent technique exploiting this fault model. This section presents the application of the fault model described above in the PFA framework.

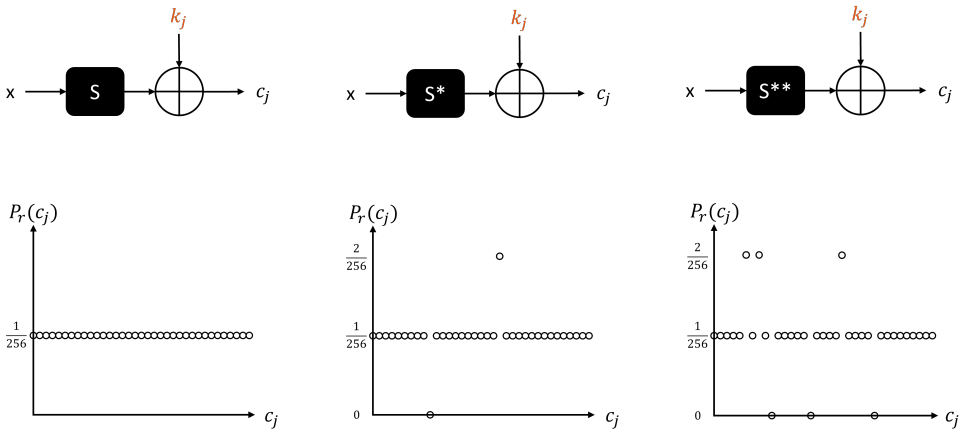
5.1 PFA principles

In 2018, Zhang *et al.* [ZLZ+18] introduced a new attack called *persistent fault analysis* or *PFA*. It requires the injection of persistent faults *i.e.* permanent fault that force the target to behave incorrectly every time the faulty code or data is executed. In the original publication, Zhang *et al.* applied this analysis to an FPGA non-volatile memory on which an AES had been implemented. It differs from a DFA which requires both faulty and unfaulty execution.

5.1.1 Overview

The attack principle involves injecting a fault on one or multiple bytes of the AES S-box used in the `SubBytes` transformation. A statistical study is then carried out on the bytes of the resulting ciphertexts.

The probability distribution of ciphertext bytes c_j, through an unfaulty S-box S and adding the key k_j, is shown in the graph of Figure 12a. Statistically, all the S-box output values have an equal probability of appearing. Therefore, the probability distribution is uniform for all values.



(a) Without fault. **(b)** With one faulty byte. **(c)** With multiple faulty bytes.

Figure 12: Probability distribution of ciphertexts bytes.

In contrast, the graph in Figure 12b shows a probability distribution that arises from

an S-box S^* in which one byte is faulty. One value, denoted as v^*, appears twice as often, while another value, denoted v is no longer present. This fault leads to a value appearing twice as often in the bytes of the ciphertexts noted c_j^{max} and a value no longer appearing at all noted c_j^{min}.

5.1.2 Attacker model

In the original PFA publication [ZLZ+18], the authors assumed that the S-box was not faulty for the key schedule. The S-box was only faulty for the encryption.

The attacker captures several ciphertexts and plots the probability distribution of the appearance of the various possible values of c_j. This allows to distinguish c_j^{min} and to use this information to recover the key with Equation 4.

$$k_j = v \oplus c_j^{min} \tag{4}$$

For the distribution to be accurate enough, and therefore to be able to find with certainty which byte no longer appears, an average of 2,272.9 ciphertexts are required. In 2020, Zhang *et al.* [ZZJ+20] proposed an enhanced version of the PFA using the maximum likelihood estimation (MLE). This improvement allows an attacker to retrieve c_j^{min} with only 1,640.7 ciphertexts in average. To ensure the success of this method, it is crucial to have only one faulty byte. Based on the experimental characterization described in Subsection 4.3, it was determined that an average of 2.2 bits are faulty. Therefore, the decision was made not to implement this version.

The analysis can also be performed when multiple faults are injected. When multiple faults are injected, the probability distribution is not uniform, and so the PFA can be used. Figure 12c shows these probabilities, with several values appearing more often and several values not appearing at all. Note that in this figure, three bytes are faulty, so three values no longer appear. However, even if in this figure three values appear twice as often, it is also possible that the different faults injected lead to a value appearing more than twice, if several faulty values are equal. In any case, analysis is still possible.

Even if the analysis is still possible, the increase in the number of faults is linked to the decrease in the efficiency of the analysis and the increase in the complexity of the analysis. Indeed, with several faults injected in the S-box, several c_j are associated to c_j^{min} and c_j^{max}. The probability-based Equation 4 is therefore less efficient. Several v and several c_j^{min} exist.

The analysis is then possible if one or several bytes are altered in the S-box. However, according to this number of injections, the computations performed are slightly different. This section presents the application of the analysis with one or several altered bytes.

5.1.3 1-byte fault

In the first analysis by Zhang *et al.* [ZLZ+18], the attacker knows where the fault was injected and what is the fault value. They therefore know the v and v^* values. After eliminating all the c_j that appear at least once, they can then directly use Equation 4 on probabilities to determine the key byte k_j.

However, the analysis can still be conducted even without knowledge of the fault value. Indeed, the authors show that this value can be found in 623 ciphers in average [ZZJ+20]. In fact, by making assumptions about this fault value, it is possible to find out what the true value is. Similarly, the location i where the fault was injected into the S-box may also be unknown, and the analysis leads to find it at the same time as calculating c_j^{min}, because around 300 messages are needed to find i. Therefore on average 2,200 ciphertexts are needed to find the key, the values of f and i have more than enough ciphers to be found. Table 1 summarises these different amounts of ciphertexts required to recover

the information used by the PFA. To ensure that the key is found with each PFA use, a threshold of 3,000 ciphertexts has been chosen.

Table 1: Average number of ciphertexts needed to recover a given information in the PFA setting.

Information	Average number
Fault location (i)	300
Fault value (f)	600
c_j^{min}	2,200

In practice, the initial step is to recover c_j^{min}. As a large number of ciphertexts are required during this step, we consider that the fault value is the result of the `Xor` operation between c_j^{min} and c_j^{max}. However, since the plaintexts are considered random, it is possible that the c_j^{max} does not represent the value of the S-box that appears twice. An error may occur. However, the fault is consistent across all 16 bytes of the ciphertext. Therefore, the probability of each of the 16 bytes producing the same error is very low. The actual fault injected is thus the value which is in the majority between all the hypotheses of faults.

To determine the fault location, an additional analysis must be conducted. An hypothesis on this fault location is done. That means that this value must never appear at any output of the `SubBytes` transformation. The hypothesis of the fault location, with the actual knowledge of the fault value, gives an hypothesis on the last round key k_{10}. It is then possible to rewind the algorithm to the output of the 9^{th} round `SubBytes`, and to check if, with any of the received ciphertexts produce an impossible S-box output. The presence of the impossible value in a ciphertext refutes the hypothesis. Statistically, the correct hypothesis is the only one for which the impossible value does not appear.

5.1.4 n-byte fault

The main distinction between the analysis of 1-byte and n-byte faults is the uncertainty surrounding the number of altered bytes into the S-box. As a result, a `while` loop cannot be used to obtain ciphertexts until n values do not appear for each byte. In this case, a threshold must be selected for the number of ciphertexts obtained to ensure confidence on the distribution of byte appearance probabilities. We chose a threshold of 3000. With a quick simulation on the number of appearances for the first byte of the ciphertexts (with a manually altered byte in the S-box), Figure 13 shows that 3000 ciphertexts are sufficient to distinguish between the value that never appears and the value that appears multiple times.

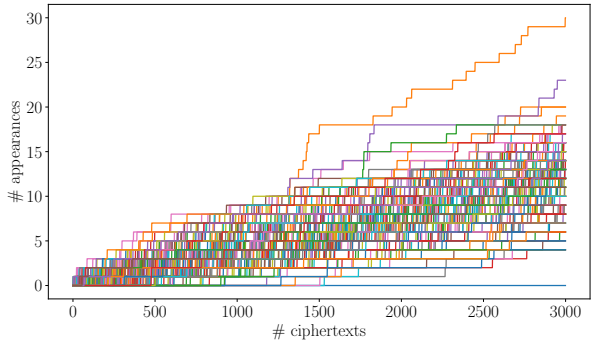


Figure 13: Number of appearance of byte values.

When the fault value was easy to find with a 1-byte fault, we have here multiple pairs

of c_j^{min} and c_j^{max}. For example, with two altered bytes, we have 4 different values for the faults injected, when only two of them are correct. Indeed, with $c_j^{min_1}$ and $c_j^{min_2}$ the two byte values that never appear and $c_j^{max_1}$ and $c_j^{max_2}$ the two byte values that appear more often, the fault candidates are : $c_j^{min_1} \oplus c_j^{max_1}$, $c_j^{min_1} \oplus c_j^{max_2}$, $c_j^{min_2} \oplus c_j^{max_1}$ and $c_j^{min_2} \oplus c_j^{max_2}$.

The next step is then to find the right pairs of fault location and fault value. However, n altered bytes on the S-box lead to n different c_j^{min} for each byte of the ciphertexts. Then, n^{16} different combinations are possible for the key. The analysis is thus more and more complicated with the increase of the number of faulty bytes.

The injection must be precise in order to apply the analysis effectively. The fault location analysis is also performed in three steps:

- make an hypothesis on the location (S_{min})
- make an hypothesis on the combination of c_j^{min} (the two hypotheses lead to hypothesis on the last round key)
- rewind to the output of the penultimate `SubBytes` operation and verify if the impossible value (S_{min}) appears for any of the ciphertexts stored

The correct key is the only combination of assumptions for which the impossible output of the S-box does not appear once for any of the threshold number of ciphertexts.

5.2 Attack implementation

5.2.1 Target

The hardware target is identical to that described in Subsection 4.2. We decided to implement a tiny-AES with a 128-bit key as described in the GitHub repository[3] with a few modifications. Following the standard PFA attacker model described by Zhang *et al.* in [ZLZ+18], uses of the S-box in the key expansion are not considered.

The AES S-box consists of 256 bytes indexed from 0 to 255. One page of the Flash memory contains 1 kB thus the S-box is stored in a quarter of a page commonly called a wordline (if the data are aligned in Flash memory). The wordline is divided in 32 columns, each column stores 64 bits. The S-box used is stored in the Flash memory as described in Figure 14 where $S_{i,j}$ is the i^{th} bit of the j^{th} value of the S-box.

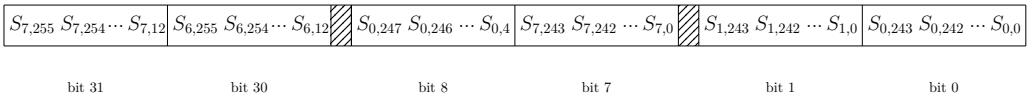


Figure 14: Physical S-box implementation in Flash Memory. $S_{i,j}$ is the i^{th} bit of the j^{th} byte value of the S-box.

Thus, we can see that the 256 bytes of the S-box are organized in 32-bit words, corresponding to the concatenation of 4 bytes of the S-box, as following:

$$\underbrace{(S_{*,255}S_{*,251}S_{*,247}S_{*,243})}_{w_{64}}...S_{*,3}S_{*,254}S_{*,250}...S_{*,2}S_{*,253}...S_{*,1}S_{*,252}...\underbrace{(S_{*,12}S_{*,8}S_{*,4}S_{*,0})}_{w_0}.$$

The rightmost memory cell of each column stores the 32 bits of the first word w_0, while the second rightmost memory cell of each column stores the 32 bits of the following word

[3] https://github.com/kokke/tiny-AES-c

w_1. This process is repeated until the 64^{th} word of the S-box is stored. This characteristic enables us to determine that if a single fault injection occurs and affects multiple bits, those bits will be located horizontally adjacent to each other.

In the scenario we have selected, we made the hypothesis that the firmware is known to the attacker. If not, several methods have been proposed in [VOC19, GLZ+19, BH22] to extract the firmware. Thus, being aware of the firmware, an attacker knows the logical address of the S-box. This is the only information about the firmware that the attacker needs. The physical location can be retrieved by performing a mapping on a clone device as described in Subsection 4.3. A clone device is a similar component on which an attacker has read and write access.

Section 4 results reveal the precise location of the S-box in the Flash memory and where to inject faults. The experimental protocol described in Figure 15 was followed. After an initialization (programming, setting up the serial communication, etc.), the target is then unpowered. We performed 1,000 laser fault injections on the Flash memory of the target. The target is then powered. 3,000 ciphertexts are acquired. This process is repeated until we find one or multiple c_j^{min} values of bytes no longer appearing in the ciphertexts. Once this step is reached, the fault value f and the fault location can be computed. These data are necessary to estimate the S_{min} value. Combinations of c_j^{min} give the ciphertext c_{min} and lastly, the last round key can be computed using both S_{min} and c_{min}. The AES true key can be computed by following the de-expansion key algorithm.

5.2.2 Attacker model in practice

As the Persistent Fault Analysis can be performed offline, an attacker only needs to access the circuit during fault injection and not during encryption. An attacker only needs to know the ciphers, which are public data, which is a significant advantage of this attack.

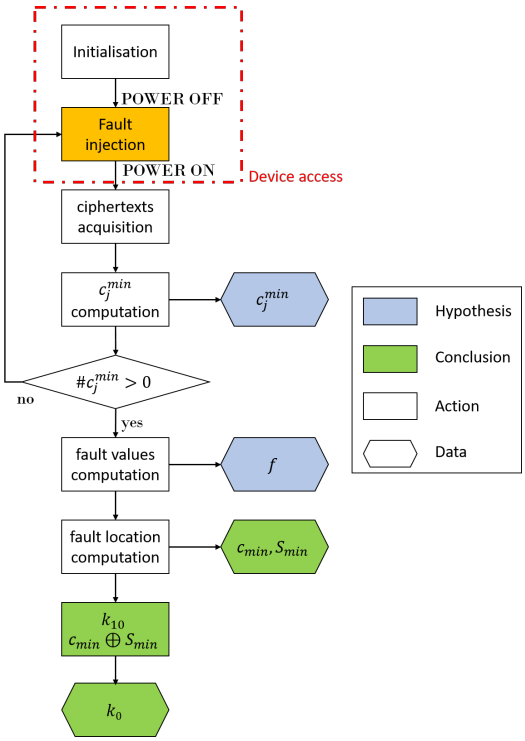


Figure 15: Protocol of the attack.

The protocol presented in Figure 15 outlines an analysis that is divided into three steps. The first step is the computation of c_j^{min}. During this step, presented in Algorithm 2, the 3,000 ciphertexts are studied, and the number of appearance of each value for each byte is stored. The resulting c_j^{min} list is composed of the values that, for each byte, never appear. If the list is empty, that means that no fault is injected in the S-box (as presented before, the threshold of 3,000 ciphertexts is enough to guarantee this result) and that we must repeat the fault injection step. If not, the number of values for each byte is the number of faulty bytes in the S-box.

The plaintexts are chosen at random. This choice is made to get closer to a very realistic adversary. Indeed, random plaintexts is the case where the adversary does not have access to the plaintexts. The only information needed is then the ciphertexts produced. Other versions of the PFA exist, in which the adversary can choose the plaintexts [SBH+22]. In such a case, by fixing all the plaintext bytes but one at 0, and by varying that one byte from 0 to 255, the key byte can be recovered with at most 256 ciphertexts [ZHF+23]. With this version of the PFA, it is needed to perform $16 \times 256 = 4096$ encryptions to recover the complete key.

Algorithm 2 c_j^{min} computation.

Require: 3000 ciphertexts (C_{LIST})
Ensure: List of impossible values for each byte (c_j^{min})

```
c_j ← [[0,...,0],...,[0,...,0]]                ▷ 16 lists of 256 zero value
for c ∈ C_LIST do
    for j ∈ [0,...,15] do
        c_j[j][c[j]] ← c_j[j][c[j]] + 1        ▷ Add 1 to the appearance of the byte value
c_j^min ← [[ ],...,[ ]]                        ▷ 16 empty lists
for j ∈ [0,...,15] do
    for i ∈ [0,...,256] do
        if c_j[j][i] = 0 then
            c_j^min[j].append[i]
return c_j^min
```

The second step is the fault computation. For each byte of the ciphertexts, we compute the values that are most likely to appear. This includes n c_j^{min} values and n c_j^{max} values for each byte when n bytes of the AES S-box are altered. The fault candidates are the results of the `Xor` operation for each pair of c_j^{min} and c_j^{max}. However, as the inputs of the cipher are considered random, the values that appear more frequently are not always linked to the faulty values of the S-box (c_j^{max} is not always related to S_{max}). This problem is solved by the fact that the faults are the same for each byte of the ciphertexts, and that all of the n faults injected are equal (this result is explained by the physical implementation of the AES S-box in the Flash memory described in Figure 14). In practice, there are $16 \times n^2$ fault candidates present (n^2 for each byte), and the actual fault value is the candidate that appears most frequently.

Lastly, the third step is the computation of the fault location. The objective is to identify the value of the S-box that never appears and the impossible ciphertext that is associated with this faulty value. By combining these two pieces of information, the attacker can recover the last round key k_{10}. This step is presented in Algorithm 3.

5.2.3 PFA with a unidirectional fault model

To efficiently apply the PFA, some modifications to the original analysis are applied. The first improvement is based on the knowledge of the fault injection mechanism. The

Algorithm 3 S_{min} computation.

Require: List of the c_j^{min}, list of the S_{min} candidates, list of the ciphertexts C_{LIST}
Ensure: One fault location s_{min} and its impossible ciphertext associated c_{min}

```
for s_min ∈ S_min do                          ▷ Hypothesis on the fault location
    for c_min ∈ c_j^min do                    ▷ Hypothesis on the c_j^min combination
        k_10 ← s_min ⊕ c_min
        k_9 ← reverse_key_schedule_k9(k_10)
        for c_0 ∈ C_LIST do
            c_1 ← shiftRowsInv(...
            mixColumnsInv(addRoundKey(...
            invSubBytes(ShiftRowsInv(...
            addRoundKey(c_0, k_10))), k_9)))
            if s_min ∈ c_1 then
                next s_min
        return k_10 ▷ The pair of valid c_min and s_min gives the right last round key k_10
```

advantage of this attack is that we know the impact of the fault on the NOR Flash memory. Indeed, a fault injected on an S-box value can only inject *bitsets* in the value. The fault location hypothesis is initially made among 256 candidates, and with the knowledge of the fault model, some fault location candidates can be eliminated. Furthermore, this improvement is also correlated with the number of faults that have been detected. Indeed, the number of impossible values for each byte of the ciphertexts is equal to the number of altered bytes in the S-box. The Flash memory is organized in such a way that is multiple faults are injected into the Flash memory, these faults will alter the same bit in different bytes of the AES S-box (as we perform the fault injection at only one position, the multiple altered bytes are linked to adjacent bits under the laser spot). The S-box is stored in the memory in such a way that two consecutive floating gate transistors save the same bit of two S-box values 4-byte away from each other (as presented in Figure 14, two consecutive bits in the Flash memory, for the same column location, store the same bits of two S-box values 4-bytes away). At this instant of the analysis, the fault value and the fault number are already known, then some S_{min} candidates can be deleted. Indeed, for n faults injected, if one candidate is valid, the other candidates must be in the neighbourhood according to the flash memory organization, unless the candidate is eliminated.

In practice, we compare the `Or` and the `Xor` operations between the fault value and the S_{min} candidate. If both of the operations yield the same result, then the S_{min} candidate is considered valid. The method is applied to all S-box values to create a map of valid and invalid values. Then, with the known number n of injected faults, we verify that a valid candidate is actually surrounded by $n-1$ valid candidates. If not, the candidate is eliminated. Table 2 presents on average the number of candidates eliminated according to the number of faults injected (note that all possible faults are 1-bit faults, and that the number of eliminated candidates is very close for any fault value). This improvement enables the analysis to be applied to less than 50 % of the S-box, resulting in a analysis that is at least twice as fast.

Table 2: Number of remaining candidates according to the number of injected faults.

Number of injected faults	Left candidates
1	50 %
2	36 %
3	25 %
4	14 %

The second improvement is based on the elimination of pairs of c_{min} and S_{min} candidates. Indeed, the original PFA proposed in [ZLZ+18] explains that to eliminate an S_{min} candidate, the attacker has to count the number of different byte values at the output of the penultimate `SubBytes` operation. If the number of those values is 256, then the candidate is eliminated. However, we rather just look for the S_{min} hypothesis inside the output of the penultimate `SubBytes` operation. Indeed, rather than waiting for 256 values to appear, we just have to check one precise value. With this improvement, a candidate is eliminated on average 6 times faster.

The last improvement is the use of the T-table version of `mixColumnsInv` in the software implementation of Algorithm 3, which is more than 3 times faster than the original one.

The source code used for the PFA and the ciphertexts will be published if *the submission is accepted*.

5.3 Experimental results

From the *map* file or the *hex* file generated during compilation or during the firmware extraction, we know that the S-box is stored between the addresses `0x080012F4` and `0x080013F3` which are located within page 4 of the Flash memory. To find the physical location of these addresses, we set the XY origin on the top left corner of the Flash memory. We then perform a small laser fault injection mapping on the clone device around the physical location of the S-box in the hardware target until we find the precise position of the spot. We obtain the fault `0x00000002` at the address `0x08001310` for the position `(x,y)=(44.3,300)`. This address is indeed in the S-box range.

The target device can now be attacked by placing the laser spot at the same position. We managed to inject faults in the S-box by performing a few series of 1,000 laser shots. Then, only 3,000 ciphertexts are required to perform the attack. For instance, it was found that the 32^{th} value of the S-box was faulty (`0xC2` instead of `0xC0`) which means that the fault value is `0x02`. As we only have one fault, the calculation of k_{10} and the reverse of the key expansion is instantaneous.

In the case where two bytes of the S-box are faulty, the attack is completed in less than an hour (the worst case being the case where the faulty bytes are at the end of the S-box). This value is obtained by running simulation as in experimental results we obtain only one byte of the AES S-box is altered.

6 Discussion

In this paper, we put forth a novel attack vector. As with the advent of any new attack vector, the question of countermeasures inevitably arises.

In the present case, the physical target is the Flash memory of a common microcontroller without any memory protection mechanism. It is true that most cryptographic devices include memory protection mechanisms, including error correction codes, error detection codes, memory scrambling, memory encryption and secure boot. These protections will be effective against the attack we propose. However, in the context of the Internet of Things (IoT), where manufacturing costs represent a significant factor, memory protection mechanisms are not typically implemented in addition to encryption algorithms. For example, the component we chose (STMicroelectronics STM32F1) is dedicated to IoT applications and has no security features. It is therefore recommended that the new attack technique described in this paper be further studied in conjunction with the implementation of memory protection features. Consequently, the attack technique we propose shall constrain semiconductor manufacturers and firmware designers to implement memory protection mechanisms.

To date, very few PFA-specific countermeasures have been proposed. One of them [ZZJ+20] consists in storing a reference pair of plaintext and ciphertext, encrypted with the secret key, and verifying the correctness of the ciphertext from time to time. It is also possible to implement other countermeasures, which have been specifically designed for the PFA. One such proposal is that put forth by Tissot *et al.* [TBG23], in which the integrity of the S-box is verified by traversing the loops it contains. In conclusion, this paper reinforces the interest of the PFA and encourages the hardware security community to propose new countermeasures that are less costly.

7 Conclusion and future work

This paper demonstrates for the first time the possibility of performing laser fault injection on unpowered devices. It has been reported that a laser spot with a diameter of 5 µm can be used to inject faults in the Flash memories with a bit-level resolution. We obtained a bitset fault model and developed the fault model from the physical level up to the software. We also assume that the fault injection is due to a local thermal effect. By heating the floating gate transistors of the Flash memory, electrons stored in the floating gate get enough energy to escape. We obtain a bitset fault model (for the most-common convention). It is possible to encounter the opposite convention, we will then obtain a bitreset fault model. The laser fault injection on unpowered target is characterized by the discharge of the charges stored in the floating gate and an unidirectional fault model.

This article demonstrates that powerful attacks as PFA can be conducted with a realistic attacker model. The attack allows us to recover a 128-bit AES key based on the ciphertexts alone.

There are several potential attacks that could be developed with this new attack path in the future. Using this injection technique to corrupt a firmware or change access rights (set all the bits of password to a `1` value) can present interesting challenges. Also, other hardware targets or cryptographic algorithms could be investigated.

Furthermore, there is still scope for improvement in the PFA. For instance, it is conceivable to use the knowledge of the fault model and use the maximum a posteriori estimation in order to determine the attack parameters with fewer ciphertexts.

Acknowledgments

This work was supported by research grants of the projects POP ANR-21-CE39-0004 and PROPHY ANR-22-CE39-0008 from the french Agence Nationale de la Recherche.

References

[ADM+10] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *16th IEEE International On-Line Testing Symposium (IOLTS 2010), 5-7 July, 2010, Corfu, Greece*, pages 235–239. IEEE Computer Society, 2010.

[ADN+10] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When clocks fail: On critical paths and clock faults. In Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny, editors, *Smart Card Research and Advanced Application, 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany, April 14-16, 2010. Proceedings*, volume 6035 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2010.

[ARM12] ARM. *ARM Architecture Reference Manual: ARMv7-A and ARMv7-R edition*, July 2012.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, pages 37–51, 1997.

[BFP19] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. Shaping the glitch: Optimizing voltage fault injection attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019:199–224, 2019.

[BGH+14] Noemie Beringuier-Boher, Kamil Gomina, David Hély, Jean-Baptiste Rigaud, Vincent Beroulle, Assia Tria, Joel Damiens, Philippe Gendrier, and Philippe Candelier. Voltage glitch attacks on mixed-signal systems. In *17th Euromicro Conference on Digital System Design, DSD 2014, Verona, Italy, August 27-29, 2014*, pages 379–386. IEEE Computer Society, 2014.

[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. In Luca Breveglieri, Sylvain Guilley, Israel Koren, David Naccache, and Junko Takahashi, editors, *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*, pages 105–114. IEEE Computer Society, 2011.

[BH22] Daehyeon Bae and Jaecheol Ha. Implementation of disassembler on microcontroller using side-channel power consumption leakage. *Sensors*, 22(15), 2022.

[BMPM13] Stephen Buchner, Florent Miller, Vincent Pouget, and Dale McMorrow. Pulsed-laser testing for single-event effects investigations. *IEEE Transactions on Nuclear Science*, 60(3):1852–1875, 2013.

[CBD+15] Clement Champeix, Nicolas Borrel, Jean-Max Dutertre, Bruno Robisson, Mathieu Lisart, and Alexandre Sarafianos. SEU sensitivity and modeling using pico-second pulsed laser stimulation of a D flip-flop in 40 nm CMOS technology. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFTS 2015, Amherst, MA, USA, October 12-14, 2015*, pages 177–182. IEEE Computer Society, 2015.

[CGOZ99] Paulo Cappelletti, Carla Golla, Piero Olivo, and Enrico Zanoni. *Flash Memories*. Springer New York, NY, 1999.

[CGV+21] Brice Colombier, Paul Grandamme, Julien Vernay, Émilie Chanavat, Lilian Bossuet, Lucie de Laulanié, and Bruno Chassagne. Multi-spot laser fault injection setup: New possibilities for fault injection attacks. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 2021.

[CMD+19] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, May 5-10, 2019*, pages 1–10. IEEE, 2019.

[CMN05] Giovanni Campardo, Rino Micheloni, and David Novosel. *VLSI-Design of Non-Volatile Memories*. Springer Berlin, Heidelberg, 2005.

[DBC+18] Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan De Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hély, Régis Leveugle, Paolo Maistri, Giorgio Di Natale, Athanasios Papadimitriou, and Bruno Rouzeyre. Laser fault injection at the CMOS 28 nm technology node: an analysis of the fault model. In *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2018, Amsterdam, The Netherlands, September 13, 2018*, pages 1–6. IEEE Computer Society, 2018.

[DDRT12] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic transient faults injection on a hardware and a software implementations of AES. In Guido Bertoni and Benedikt Gierlichs, editors, *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*, pages 7–15. IEEE Computer Society, 2012.

[ERM16] David El-Baze, Jean-Baptiste Rigaud, and Philippe Maurine. A fully-digital EM pulse detector. In Luca Fanucci and Jürgen Teich, editors, *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pages 439–444. IEEE, 2016.

[GBD23] Paul Grandamme, Lilian Bossuet, and Jean-Max Dutertre. X-ray fault injection in non-volatile memories on power off devices. In *2023 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pages 1–7, 2023.

[GLZ+19] Chao Gao, Lan Luo, Yue Zhang, Bryan Pearson, and Xinwen Fu. Microcontroller based IoT system firmware security: Case studies. In *IEEE International Conference on Industrial Internet, ICII 2019, Orlando, FL, USA, November 11-12, 2019*, pages 200–209. IEEE, 2019.

[HNT+13] Clemens Helfmeier, Dmitry Nedospasov, Christopher Tarnovsky, Jan Starbug Krissler, Christian Boit, and Jean-Pierre Seifert. Breaking and entering through the silicon. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 733–744. ACM, 2013.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996.

[LVD+22] Rodrigo Silva Lima, Raphael Viera, Jean-Max Dutertre, Anne-Lise Ribotta, Matthieu Pommies, and Anthony Bertrand. Target preparation methodology for semi-invasive attacks on microcontrollers. In *2022 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pages 1–7. IEEE, 2022.

[MDC+20] Alexandre Menu, Jean-Max Dutertre, Brice Colombier, Jean-Baptiste Rigaud, Pierre-Alain Moëllic, and Jean-Luc Danger. Single-bit laser fault model in NOR flash memories: Analysis and exploitation. In *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*, pages 41–48. IEEE, 2020.

[Men21] Alexandre Menu. *Sécurité matérielle des objets connectés*. Theses, Université de Lyon, November 2021.

[NRV+06] Egas Henes Neto, Ivandro Ribeiro, Michele Vieira, Gilson Wirth, and Fernanda Lima Kastensmidt. Using bulk built-in current sensors to detect soft errors. *IEEE Micro*, 26(5):10–18, 2006.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 243–260. Springer, 2014.

[O'F16] Colin O'Flynn. Fault injection using crowbars on embedded systems. *IACR Cryptol. ePrint Arch.*, page 810, 2016.

[RSDT13] Cyril Roscian, Alexandre Sarafianos, Jean-Max Dutertre, and Assia Tria. Fault model analysis of laser-induced faults in SRAM memory cells. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 89–98. IEEE Computer Society, 2013.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[San11] David Sands. Pulsed laser heating and melting. In *Heat Transfer*. IntechOpen, 2011.

[SBH+22] Hadi Soleimany, Nasour Bagheri, Hosein Hadipour, Prasanna Ravi, Shivam Bhasin, and Sara Mansouri. Practical multiple persistent faults analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):367–390, 2022.

[SBHS15] Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise laser fault injections into 90 nm and 45 nm SRAM-cells. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2015.

[SGS+13] Alexandre Sarafianos, Olivier Gagliano, Valérie Serradeil, Mathieu Lisart, Jean-Max Dutertre, and Assia Tria. Building the electrical model of the pulsed photoelectric laser stimulation of an nmos transistor in 90nm technology. In *2013 IEEE International Reliability Physics Symposium (IRPS)*, pages 5B.5.1–5B.5.9, 2013.

[SH07] Jörn-Marc Schmidt and Michael Hutter. Optical and EM fault-attacks on crt-based rsa: Concrete results. In *Austrochip 2007, 15th Austrian Workhop on Microelectronics, 11 October 2007, Graz, Austria, Proceedings*, pages 61–67. Verlag der Technischen Universität Graz, 2007.

[SHP09] Jörn-Marc Schmidt, Michael Hutter, and Thomas Plos. Optical fault attacks on AES: A threat in violet. In Luca Breveglieri, Israel Koren, David Naccache, Elisabeth Oswald, and Jean-Pierre Seifert, editors, *Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC*

2009, Lausanne, Switzerland, 6 September 2009, pages 13–22. IEEE Computer Society, 2009.

[Sko09] Sergei P. Skorobogatov. Local heating attacks on flash memory devices. In Mohammad Tehranipoor and Jim Plusquellic, editors, *IEEE International Workshop on Hardware-Oriented Security and Trust, HOST 2009, San Francisco, CA, USA, July 27, 2009. Proceedings*, pages 1–6. IEEE Computer Society, 2009.

[Sko10] Sergei Skorobogatov. Optical fault masking attacks. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2010, Santa Barbara, California, USA, 21 August 2010*, pages 23–29. IEEE Computer Society, 2010.

[SLD+12] Alexandre Sarafianos, Roxane Llido, Jean-Max Dutertre, Olivier Gagliano, Valerie Serradeil, Mathieu Lisart, Vincent Goubier, Assia Tria, Vincent Pouget, and Dean Lewis. Building the electrical model of the photoelectric laser stimulation of a PMOS transistor in 90 nm technology. *Microelectron. Reliab.*, 52(9-10):2035–2038, 2012.

[SRD+13] Alexandre Sarafianos, Cyril Roscian, Jean-Max Dutertre, Mathieu Lisart, and Assia Tria. Electrical modeling of the photoelectric effect induced by a pulsed laser applied to an SRAM cell. *Microelectron. Reliab.*, 53(9-11):1300–1305, 2013.

[SSAQ02] David Samyde, Sergei P. Skorobogatov, Ross J. Anderson, and Jean-Jacques Quisquater. On a new way to read data from memory. In *Proceedings of the First International IEEE Security in Storage Workshop, SISW 2002, Greenbelt, Maryland, USA, December 11, 2002*, pages 65–69. IEEE Computer Society, 2002.

[STM10] STMicroelectronics. *Programming Manual : STM32F100XX value line Flash programming*, October 2010. Rev. 2.

[TBG23] Pierre-Antoine Tissot, Lilian Bossuet, and Vincent Grosso. Baloo: First and efficient countermeasure dedicated to persistent fault attacks. In Alessandro Savino, Michail Maniatakos, Stefano Di Carlo, and Dimitris Gizopoulos, editors, *29th International Symposium on On-Line Testing and Robust System Design, IOLTS 2023, Crete, Greece, July 3-5, 2023*, pages 1–7. IEEE, 2023.

[VDDM21] Raphael Andreoni Camponogara Viera, Jean-Max Dutertre, Mathieu Dumont, and Pierre-Alain Moëllic. Permanent laser fault injection into the flash memory of a microcontroller. In *19th IEEE International New Circuits and Systems Conference, NEWCAS 2021, Toulon, France, June 13-16, 2021*, pages 1–4. IEEE, 2021.

[VOC19] Sebastian Vasile, David Oswald, and Tom Chothia. Breaking all the things—a systematic survey of firmware extraction techniques for IoT devices. In *Smart Card Research and Advanced Applications*, pages 171–185. Springer International Publishing, 2019.

[ZDCR14] Loïc Zussa, Jean-Max Dutertre, Jessy Clédière, and Bruno Robisson. Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2014, Arlington, VA, USA, May 6-7, 2014*, pages 130–135. IEEE Computer Society, 2014.

[ZHF+23] Fan Zhang, Run Huang, Tianxiang Feng, Xue Gong, Yulong Tao, Kui Ren, Xinjie Zhao, and Shize Guo. Efficient persistent fault analysis with small number of chosen plaintexts. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(2):519–542, 2023.

[ZLZ+18] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. Persistent fault analysis on block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):150–172, 2018.

[ZZJ+20] Fan Zhang, Yiran Zhang, Huilong Jiang, Xiang Zhu, Shivam Bhasin, Xinjie Zhao, Zhe Liu, Dawu Gu, and Kui Ren. Persistent fault attack in practice. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):172–195, 2020.