Analysis of a Laser-induced Instructions Replay Fault Model in a 32-bit Microcontroller

Vanthanh Khuat, Jean-Max Dutertre, Jean-Luc Danger

Analysis of a Laser-induced Instructions Replay Fault Model in a 32-bit Microcontroller

Vanthanh Khuat*‡, Jean-Max Dutertre† and Jean-Luc Danger*
*LTCI, Télécom Paris, Institut polytechnique de Paris, France
Email: {khuat, jean-luc.danger}@telecom-paris.fr
†Mines Saint-Etienne, CEA, Leti, Centre CMP, F - 13541 Gardanne France
Email: dutertre@emse.fr
‡Faculty of Information Technology, Le Quy Don Technical University, Hanoi, Vietnam
Email: van-thanh.khuat@lqdtu.edu.vn

***Abstract*—In this paper, we present a method to obtain a new Laser Fault Injection (LFI)-induced fault model: replay of instructions on a 32-bit Microcontroller (MCU). This method allows a potential adversary to replay a block of two or four instructions with a fault rate up to 100%. These faults are induced by laser pulses and cause the instructions updating process of a Flash buffer to fail. As a result, the new instructions failing to be stored in the Flash buffer, the previous ones are replayed. We deeply studied the properties of this replay fault model by many experiments of laser fault injections. We have notably shown that the sensitivity window is proportional to the laser Pulse Width (PW), and that up to 20 instructions in a row were tested to be overwritten due to replaying five times the block of four instructions. The effects of the laser power and cache status (enabled or disabled) are also presented. Finally, we proposed and assessed a simple method to detect the LFI-induced replay faults using a hardware counter with different increments. Our results extend the ability of LFI on MCU, illustrating the accuracy and reproducibility of LFI.**

***Index Terms*—Laser fault injection, Fault models, Characterization, Microcontroller.**

I. Introduction

Thanks to its advantages such as low-cost, compact size, and easy to use, MCU is an outstanding candidate for many Internet of Things (IoTs) applications, in which the MCUs process plenty of valuable information such as password, account number, identity, critical data, etc. The MCUs are subject to many types of attack. As they are physically accessible, physical attacks become possible in addition to cyber attacks. One of the most powerful threat is the Fault Injection (FI).

FI is an active physical attack in which the attacker induces faults into a target to further exploit them in order to obtain information by differential fault analysis (fault vs no-fault). The most common techniques used for Fault Injection Attacks (FIA) are: Clock or voltage tampering [1], Electromagnetic Fault Injection (EMFI) [8], Optical fault injection [4], [13]. The attacks can be classified into invasive, semi-invasive, and non-invasive attacks. LFI is a semi-invasive one because in this technique the device needs to be unpackaged to ensure that the light can reach the circuit layer. Optical fault injection is the method in which the optical source, specifically the laser source, is used to induce ionization in electronic circuit devices hence causing the targets to behave differently. The disadvantages of this method are: (1) the cost of equipment is relatively high (2) and well-trained staffs are needed to operate the laser. However, the laser has the advantage that it has a very high temporal and spatial accuracy. This comes from the fact that the laser pulse can be confined in a very small space (a few μm), and lasts for a very short time.

The optical source had been used for injecting transient localized disturbances into the electronic circuit since 1965 [5], but the first optical attack was only reported in [13], in which the author succeeded to use the laser illumination to change the data stored in memory cells. Then there have been plenty of works focusing on examining the characteristics of the optical attack, understanding the mechanism underneath, and developing the countermeasures against it. Among these, many are dedicated to studying different kinds of laser-induced faults in MCU.

LFI is reported of being able to cause faults on data stored in memories such as volatile type [12], or data being transferred on the communication bus without affecting the data stored in the source memories [2]. Specifically, LFI can target and modify instructions being read from memories to Processor Core (PC), causing instruction corruption, instruction skip, etc. Recently, Dutertre et al. [4] reported a LFI-induced powerful multiple instructions skip fault model, in which the authors were able to achieve an arbitrary number of skipped instructions with a maximum of 300 instructions being skipped by using laser pulse with a relatively long PW.

FI-induced fault in MCU can also prevent the update of a block of several instructions to be executed, resulting into the replay of the previous block of instructions (as if the targeted instructions have been overwritten by the replayed ones). In [11], the authors succeed to use Electromagnetic (EM) pulse to prevent the update of a data buffer during cache read, replaying a block of up to four instructions. However, to our best knowledge, there is no paper reporting such fault model using LFI.

In this paper, we report on the observation of a new fault model induced by LFI: the replay of block of instructions in a 32-bit MCU. The main contributions of this paper are:

- obtaining a new LFI-induced fault model: the replay of block of instructions in a 32-bit MCU;

- analyzing the impact of the laser power on the replay fault rate and fault window;
- investigating the impact of laser PW on the replay fault model and extending the fault model to multiple times replay of a block of instructions;
- achieving the replay fault with cache enabled and cache disabled, comparing them to each other;
- proposing a countermeasure at instruction level to detect the replay fault.

The rest of the paper is organized as follows. Section II describes the experimental setup, target, test code, fault definition, and methodology. Section III discusses the fault models of replay of block instructions achieved with LFI. Section IV discusses the dependency of the number of instructions being overwritten as the function of the laser PW. Section V characterizes the impact of the laser power on the replay fault. Section VI investigates the replay fault model as the cache is enabled and compares it with result obtained as the cache is disabled. Section VII describes a proposed detection method for LFI-induced replay fault and provides experimental results. Finally, section VIII provides the main conclusions and some perspectives.

II. Experimental Setup and Methodology

A. Laser bench

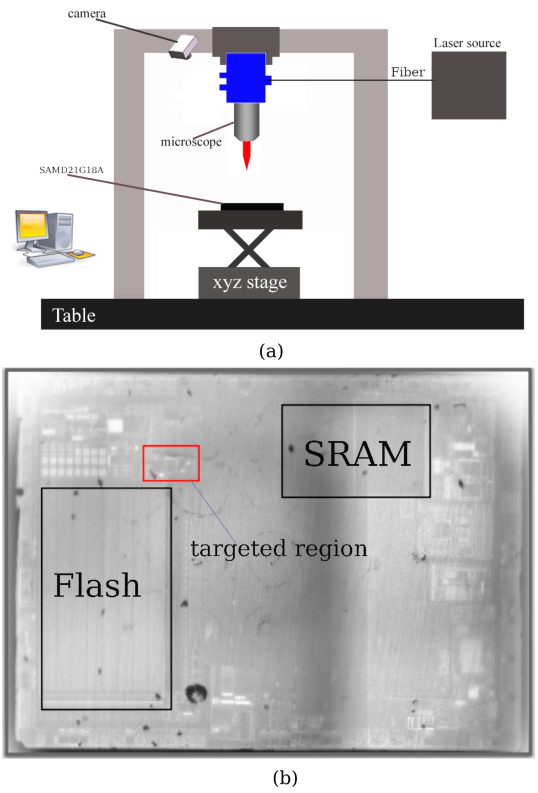

Figure 1. Experimental setup: (a) laser bench schematic, (b) target back-side image taken using an IR camera.

Our laser setup is depicted in Fig. 1(a), it consists in a laser source, a microscope, a XYZ table, a camera, and a computer. The laser source can produce laser pulses with a wavelength of 1,064 nm which allows the light to pass through several hundreds of μm of silicon. The laser PW is tunable in the range from 50 ns to 1 s. Note that in this work we only considered single laser pulse (several laser pulses could also be used as in [4]). In addition, the laser source allows obtaining a programmable delay, and power ranging from 0 to 3 W. The laser light is conducted to and focused by a microscope. In our experiments, an objective of 20x was used to focus the pulse on the device under test. The diameter of the laser spot was 5 μm (single-mode laser fiber). More details on the laser bench can be found in [4], [10]. The device under test was mounted on a XYZ stage which allows controlling the position of the laser spot with an accuracy of 0.1 μm. The IR camera was used to see the surface of the device. The computer was used to control the laser pulse parameters and communicate with the device under test.

B. Device under test and targeted region

Our target is the 32-bit MCU SAMD21G18A [6], embedding an ARM Cortex-M0+ core (2-stage pipeline), that implements the ARMV6 thumb instruction set of which most of the instructions are 16-bit length [7]. Fig. 2 shows the block

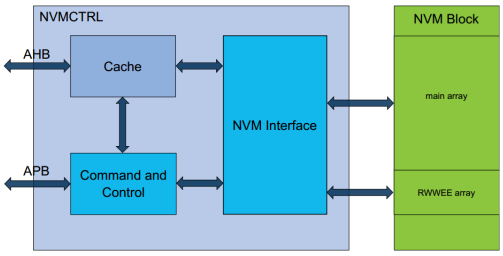

Figure 2. SAMD21 Nonvolatile Memory (NVM) interface [6]

schematic of the NVM of the SAMD21G18A. Instructions, normally stored in the NVM such as the Flash, are loaded into the Flash interface buffer before being transferred to the AHB bus. The MCU is equipped with 256 Kb Flash, and 32 Kb SRAM. The data transfer between processor and memories is performed via a 32-bit AHB and APB bus. A cache, of which the size is 8×64-bit lines, is added to improve the performance of the MCU.

The MCU was unpackaged from the backside to ensure that the light is able to reach the transistor layer. Notice that laser power is strong enough to reach the circuit layer of the MCU without requiring it to be thinned down. Fig. 1(b) shows the image taken with an IR camera from the backside of the MCU. The positions of the Flash and SRAM memories are marked with black rectangular shapes. The other structures in the circuit layer can also be seen. The region where the replay fault model is obtained is highlighted as the **targeted region** marked with a red rectangular shape which is located near the Flash region. After being opened, the MCU was soldered on a minimum circuit board designed based on [6]. For all the experiments, the MCU was configured to work at 12 MHz,

with zero wait states which guarantees there is no delay during the data read operation from the Flash memory. The MCU was debugged using an Atmel-ICE Debugger which allows stopping the program at a breakpoint and collecting registers data for further analysis.

C. Test code and fault definition

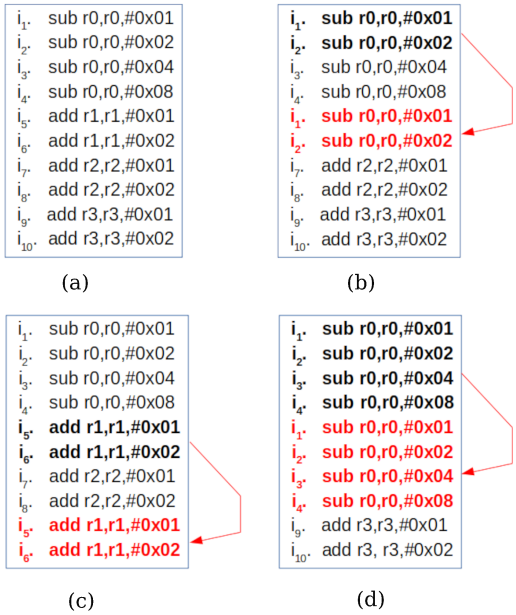

Figure 3. Test code and fault definition, (a) testcode, (b) replay i_1i_2, (c) replay i_5i_6, (d) replay $i_1i_2i_3i_4$.

The main part of our test code which consists of ten instructions is shown in Fig. 3(a). For convenience, the instructions are denoted as (i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9, i_{10}). As in cache disable mode, for every two clock cycles, 32-bit data corresponding to two 16-bit instructions are loaded from the Flash, the laser pulse was used to target the load of each two instructions. It should be pointed out that to differentiate the replay and skip of instructions, the replayed instructions shall not be no-operation (or *nop* instruction), because it would make the replay being equivalent to skip of instructions. Here, for convenience of post processing, instructions: `sub r0,r0,#value` (with value being `0x01`, `0x02`, `0x04`, `0x08`) were used for (i_1, i_2, i_3, i_4); and (i_5, i_6, i_7, i_8, i_9, i_{10}) are simply the operations to `add #value` to a register, with `value` being `0x01` and `0x02`. The initial values of the related register are as follows: `R0:0x7F8000FF`, `R1:0x01FE0000`, `R2:0x3FC00000`, `R3:0x001FE000`. And the expected values of the related registers at the end of the program are: `R0:0x7F8000F0`, `R1:0x01FE0003`, `R2:0x3FC00003`, `R3:0x001FE003`. The following fault definitions are used in this paper.

- **replay** i_1i_2: (i_5, i_6) being overwritten by (i_1, i_2) as shown in Fig. 3 (b). In this case, the final values of the related registers are: `R0:0x7F8000ed`, `R1:0x01FE0000`, `R2:0x3FC00003`, `R3:0x001FE003`;
- **replay** i_5i_6: (i_9, i_{10}) being overwritten by (i_5, i_6) as shown in Fig. 3 (c). In this case, the final values of the related registers are: `R0:0x7F8000F0`, `R1:0x01FE0006`, `R2:0x3FC00003`, `R3:0x001FE000`;
- **replay** $i_1i_2i_3i_4$: (i_5, i_6, i_7, i_8) being overwritten by (i_1, i_2, i_3, i_4) as shown in Fig. 3 (d). In this case, the final values of the related registers are: `R0:0x7F8000E1`, `R1:0x01FE0000`, `R2:0x3FC00000`, `R3:0x001FE003`;
- **other**: Instruction(s) modification, system faults, and other.

It is worth noting (as reported in section III) that in replay of two instructions: (i_5, i_6) are overwritten by (i_1, i_2) but (i_3, i_4) which are right above them, can not be replayed. While in replay $i_1i_2i_3i_4$, (i_5, i_6, i_7, i_8) are overwritten by (i_1, i_2, i_3, i_4).

D. Test procedure

We first performed a scan of all the chip surface to identify its laser-sensitive regions. Several regions were found to be sensitive to the laser pulse such as the Flash and several other regions. For the Flash region, the fault tends to be the corruption on some specific bits of the opcode, resulting into instruction modification or skip as reported in [3]. The laser spot was then fixed at an optimal position that provides the highest replay fault rate. Notice that the same fault behavior can still be achieved as we moved the laser spot around this position with a diameter around tens μm, however the fault rate deceases along the distance to the optimal position. All registers were initialized to a known value at the beginning of all the tests to ensure fault traceability. One test iteration follows three main steps: (1) the target is reset to initialize all systems including memories and registers; (2) the trigger for laser pulse generator is set and the test code is then executed (as a result of the trigger setting the laser source is fired); (3) registers content harvesting is performed as the program reaches the configured break-point. During the campaign, 100 tests were performed for each set of fault injection parameters. Before each test, we collected the contents of all registers at the configured break-point to confirm that the program functions correctly in the normal condition and used it as a reference to detect the fault after each LFI.

III. Replay Fault Model

A. First experimental results

At first, the MCU was configured to work in cache disabled mode. We set the laser PW to 50 `ns`, and its power to 0.75 `W`. The delay, or time elapsed between the trigger onset and the actual firing of a laser pulse, was chosen to target the loading of instructions (i_5, i_6), (i_7, i_8), (i_9, i_{10}) into the Flash buffer to see how often the fault happens. Fig. 4 depicts the LFI replay fault rate as a function of time on a 400 `ns` delay range (from 1,650 `ns` to 2,050 `ns`). For a delay around 1,680 `ns` and 2,000 `ns`, the replay i_1i_2 and replay i_5i_6 fault models were respectively obtained with fault rates up to 100%. The replay

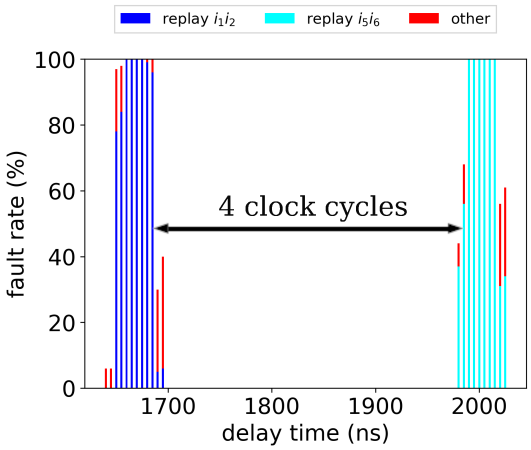

Figure 4. Fault rates of replay i_1i_2 and replay i_5i_6.

i_1i_2 happens from 1,645 ns to 1,695 ns, while the replay i_5i_6 happens from 1,980 ns to 2,025 ns. The corresponding fault windows for each were almost equal to the laser PW. It proved impossible to obtain a replay i_3i_4 fault for any value of the delay. The time interval between replay i_1i_2 and replay i_5i_6 is 4 clock cycles (i.e. ≈333 ns). Further investigation confirms that the above behavior occurs every 4 clock cycles. Therefore we focused our experiments on the time interval of four clock cycles during which the loads of instructions (i_5, i_6) and (i_7, i_8) were performed.

The laser power was set to 0.75 W, the PW was set to different values from 50 ns to 200 ns. LFI delay was adjusted to target the loads of (i_5, i_6) and (i_7, i_8) from Flash memory by changing the pulse delay time. The obtained replay fault rates are shown in Fig. 5. First, we can see that the fault sensitivity window is proportional to the laser PW and that it ends around a delay time of 1,695 ns for each value of PW. For PWs of 50 ns, 80 ns and 120 ns, a replay of two instructions with a fault rate of 100% is observed, as shown in Fig. 5(a), (b) and (c). Starting from a PW of 160 ns, a replay of four instructions is seen at the end of the fault interval as shown in Fig. 5(d). Then, as the laser PW increases, the replay of four instructions interval widens accordingly as shown in Fig. 5(e) and (f).

As the clock period is approximately 83.3 ns, the replay of four instructions fault model (i_1, i_2, i_3, i_4) starts to appear as the laser PW reaches two clock cycles (i.e. 160 ns). The replay (i_3, i_4) fault model was never observed, it is only faulted together with (i_1, i_2).

B. Fault mechanism hypothesis

For analysis purposes, we drew the scenarios of replay faults in Fig. 6 where we depicted the sensitivity window of the replay (i_1, i_2) and (i_3, i_4) fault models with green arrows marked load buffer 1 and 2 respectively. For (i_1, i_2) as long as the pulse covers the first sensitive point, a replay of these two instructions is observed as shown in Fig. 6(a). Note that the delay sensitivity window in Fig. 5 (a) to (d)

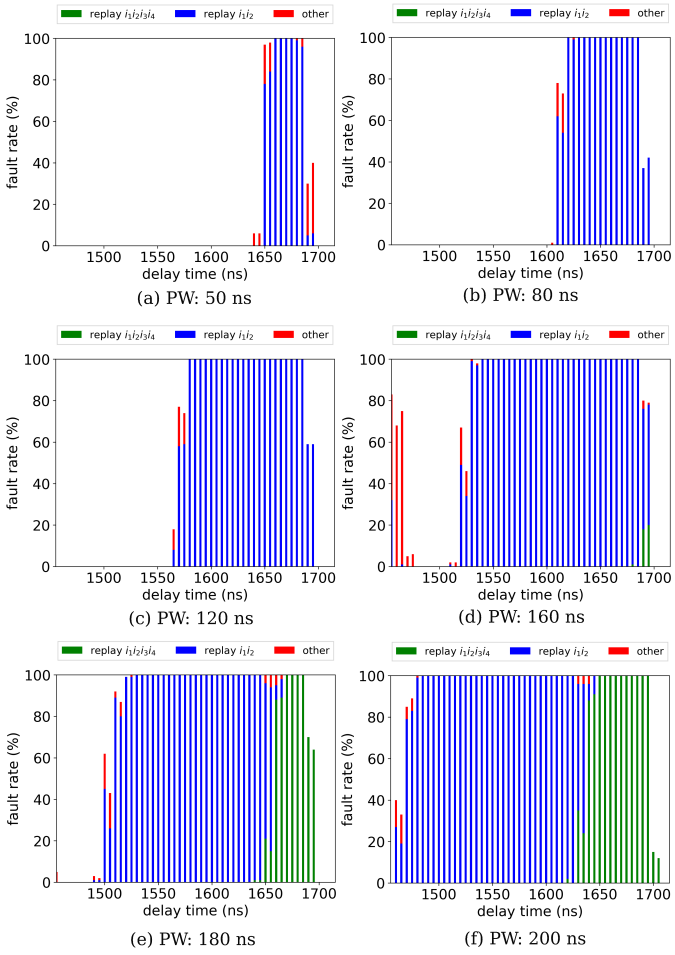

Figure 5. Obtained replay fault models and fault rates for laser PW set to: (a) 50 ns, (b) 80 ns, (c) 120 ns, (d) 160 ns, (e) 180 ns, (f) 200 ns.

extends leftward as the laser PW increases; this is because a greater PW accommodates with a lower delay to obtain a laser perturbation able to reach the instruction load process at 1,700 ns.

(i_3, i_4) can not be replayed independently as illustrated in Fig 6 (b). However, as the pulse duration is long enough to cover both sensitive points at 1,700 ns and 1,860 ns, a replay $i_1i_2i_3i_4$ is observed as exemplified in Fig. 6(c), which corresponds to the experimental results reported in Fig. 5 (e) and (f).

Because the SAMD21G18A implements an ARM Cortex-M0+ core (which has a 32-bit bus, and a 2-stage pipeline) and that the processor fetches 32 bits data (i.e. two 16-bit instructions) from the Flash memory every two clock cycles, we draw the assumption that it uses two 32-bit buffers: buffer1 and buffer2, at the Flash interface. Each of them is updated with new data every four clock cycles (with a relative phase shift of two clock cycles between them). This process is illustrated in Fig. 7. In a normal execution process, at clock cycle 2, the content of buffer1 is (i_1, i_2), and will be updated with (i_5, i_6) as shown in Fig. 7(a) (and then with (i_9, i_{10}) four

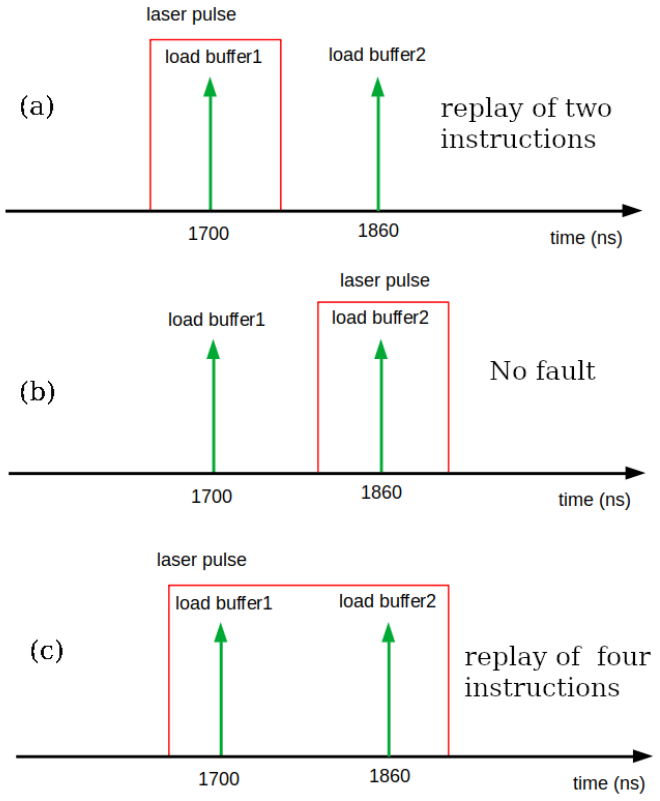

Figure 6. Different replay fault scenarios with different PWs and delay time, (a) replay $i_1 i_2$, (b) nofault, (c) replay $i_1 i_2 i_3 i_4$.

clock cycles later).

Fig. 7(b) depicts how a replay $i_1 i_2$ is induced. Buffer1 is disturbed by a laser pulse during clock cycle 2, it fails to update: its content remains (i_1, i_2): as a result (i_1, i_2) are replayed, and (i_5, i_6) are never executed (as if they were overwritten by (i_1, i_2)). The same principle is applied for replay $i_5 i_6$ in which (i_9, i_{10}) are not updated properly as shown in Fig. 7(c).

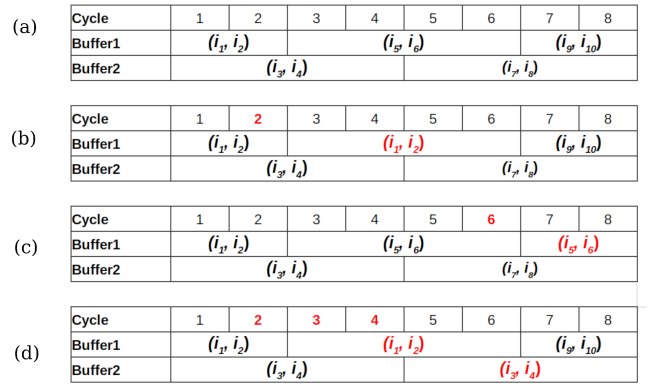

Figure 7. Replay fault hypothesis, (a) normal execution process, (b) replay $i_1 i_2$ caused by a laser pulse during clock cycle 2, (c) replay $i_5 i_6$ caused by a laser pulse during clock cycle 6, (d) replay $i_1 i_2 i_3 i_4$ caused by a laser pulse covering clock cycles 2 to 4.

Fig. 7(d) illustrates the case of a longer laser pulse (corresponding to a laser PW greater or equal to 160 ns as reported in Fig. 5(e) and (f)) induced during clock cycles 2 to 4 and able to cover the sensitivity windows of both buffer 1 and buffer 2. As a result, the update of both buffers fails and a replay of four instructions is obtained (i.e. replay $i_1 i_2 i_3 i_4$). Our experiments did not revealed why it is impossible to replay instructions (i_3, i_4) independently of instructions (i_1, i_2). It is probably due to the hardware architecture of buffers 1 and 2 that is unknown to us and to the local effect of LFI.

IV. Impact of the PW on the number of instructions being overwritten

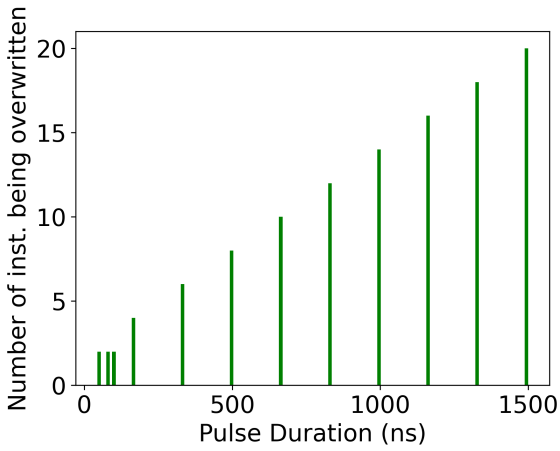

Figure 8. Impact of PW on the number of instructions being overwritten.

We then checked the ability to replay a block of instructions several times by increasing the laser PW. The laser power was set to 0.75 W, and the laser PW was increased from 160 ns to 1,500 ns with an increment step of 166 ns corresponding to a duration of two clock cycles. Fig. 8 reports the number of instructions being overwritten as a function of the laser PW. For every increase of the laser PW by two clock cycles, the number of instructions being overwritten increases by two. The same four instructions were replayed several times (with a granularity of two replayed instructions). We tested until the block (i_1, i_2, i_3, i_4) was replayed for five times corresponding to 20 instructions being overwritten (not executed). It is believed that the number of instructions being overwritten can be even more, however, we did not test with longer laser pulse because there was a risk of burning the MCU.

To summarize, this laser-induced fault injection mechanism is linked to a failure of the update processes of two 32-bit buffers at the Flash memory interface. As a result, the instructions stored into the targeted buffer(s) are replayed while the instructions failing to be fetched from the Flash are discarded (as if they are overwritten by the replay instructions). This faulty behavior has a periodicity of four clock cycles and may affect both buffers (though buffer 2 can only be affected after buffer 1 was). This fault can be reproduced several times

by increasing the laser PW, replaying the same instructions stored in the two buffers and overwritten those failing to be fetched). A fault rate of 100% was obtained experimentally.

V. Impact of the Laser Power

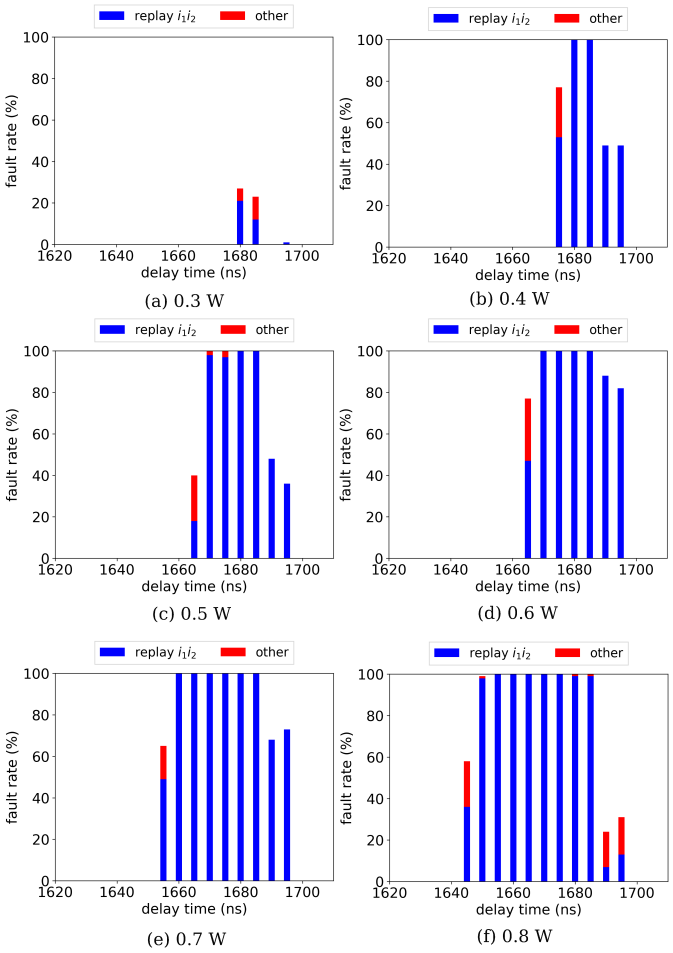

Figure 9. Impact of laser power on replay fault rate as PW is 50 ns, (a) 0.3 W, (b) 0.4 W, (c) 0.5 W, (d) 0.6 W, (e) 0.7 W, (f) 0.8 W.

To understand the impact of the laser power on the fault rate of the replay fault model, we set the PW at 50 ns and the laser power was progressively increased from 0.1 W to 0.8 W with an increment of 0.1 W. The corresponding fault rates are shown in Fig. 9. For laser power of 0.1 W and 0.2 W, no fault is detected. With the power of 0.3 W, replay i_1i_2 is observed, however the maximum fault rate is only 20% and the fault sensitivity window is very small as shown in Fig. 9(a). Starting from laser power of 0.4 W, the fault rate can reach up to 100% as shown in Fig. 9(b). As the power goes higher, the fault interval becomes wider as shown in Fig. 9 (c), (d), (e) and (f). As can be seen, the laser power has a direct impact on the replay fault rate, it should be high enough to cause the replay fault.

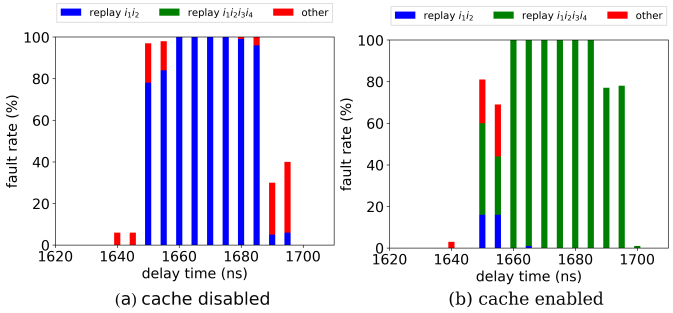

Figure 10. Comparison of replay faults obtained with cache disabled and cache enabled: (a) cache disabled, (b) cache enabled.

(a)

Cycle	1	2	3	4	5	6	7	8
Buffer (64 bits)	(i_1, i_2, i_3, i_4)			(i_5, i_6, i_7, i_8)			$(i_9, i_{10}, i_{11}, i_{12})$	
Cache	(i_1, i_2, i_3, i_4)			(i_5, i_6, i_7, i_8)				

(b)

Cycle	1	2	3	4	5	6	7	8
Buffer (64 bits)	(i_1, i_2, i_3, i_4)			(i_1, i_2, i_3, i_4)			$(i_9, i_{10}, i_{11}, i_{12})$	
Cache	(i_1, i_2, i_3, i_4)			(i_1, i_2, i_3, i_4)				

Figure 11. Hypothesis of Replay of four instruction as the cache is enabled, (a) normal execution process, (b) replay of four instruction caused by LFI-induced 64-bit buffer update prevention during clock cycle 2.

VI. Cache Disabled and Enabled Comparison

We also studied the replay fault model as the cache is enabled. The replay fault rates with cache disabled and enabled are shown in Fig. 10. The laser PW was 50 ns and the laser power was 0.75 W. It is clear that as the cache is enabled we still achieved the replay fault. And the fault interval is quite the same as that of the case when the cache is disabled. However, the replay is with a block of four instructions (i.e. replay $i_1i_2i_3i_4$) instead of two instructions. And the fault rate is up to 100% as shown in Fig. 10(b). Further investigation also showed that the fault repeats itself every four clock cycles. Therefore, it is also ascribed to the laser-induced prevention on the Flash buffer update. However unlike the case with cache disabled, as the cache is enabled, the 64 bits data corresponding to four 16-bit instructions are updated every four clock cycles. This is because the cache is 8 lines of 64 bits. Fig. 11 demonstrates how the Flash buffer is updated as the cache is enabled. As shown in Fig. 11(a), in normal execution process, at clock cycle 2 the content of the buffer is (i_1, i_2, i_3, i_4) and is supposed to update with (i_5, i_6, i_8, i_8). However, as it is disturbed by a laser pulse during clock cycle 2 the update process is prevented as illustrated in Fig. 11(b), therefore four instructions (i_1, i_2, i_3, i_4) are re-executed instead of (i_5, i_6, i_7, i_8), resulting in the replay $i_1i_2i_3i_4$ (the corresponding experiments are reported in Fig. 10(b)). It should be noticed that with cache disabled, replay $i_1i_2i_3i_4$ can also be observed as discussed in the previous section however the PW needs to be as long as two clock cycles duration to cover the update of four instructions. In addition, the replay of instruction as cache enabled here is similar to the result of Riviere [11], in which the author ascribed replay

of instructions to cache read failure as subjected to EM pulse. However, the result here clearly demonstrates that the replay of blocks of instructions occurs in both cases for cache disabled or cache disabled. In addition, it is more likely to happen at the Flash interface buffer, where the instructions are loaded periodically.

VII. METHOD FOR DETECTION OF INSTRUCTIONS REPLAY FAULT

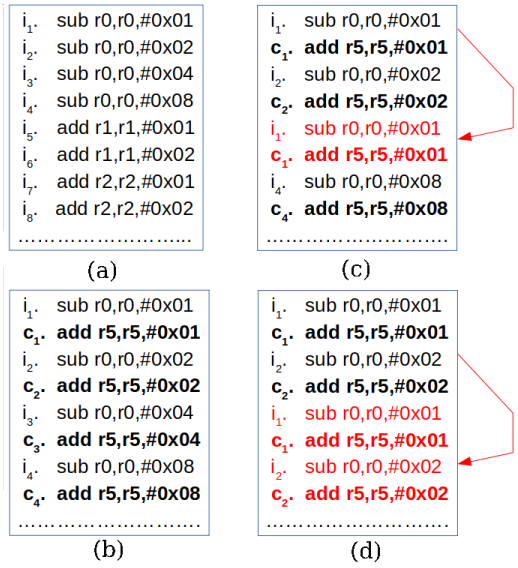

Figure 12. Detection of replay fault model (a) test code , (b) protected code, (c) replay of two instruction with protected code, (d) replay of four instructions with protected code.

We also developed a method to detect the LFI-induced instructions replay fault in a MCU. Fig. 12 illustrates the implementation of the method and its principle. Here, we consider the case of a replay that targets the block of instructions (i_1, i_2, i_3, i_4), see in Fig. 12(a). The idea behind this method is to insert a hardware counter (register `R5` in our case) to be incremented after each instruction and use it to detect the fault as highlighted in bold in Fig. 12(b). `R5` is used as the counter, and the inserted instructions are denoted as (c_1, c_2, c_3, c_4), which are simple `add` operations. Notice that each time the counter is incremented with a different value using instruction `add R5,R5, #value` with the value being `0x01,0x02,0x04,0x08`. Fig. 12(c) demonstrates the case in which replay fault happens with two instructions (i_1, c_1). While Fig. 12(d) demonstrate the case in which replay fault happens with four instructions (i_1, c_1, i_2, c_2).

As a consequence of the laser-induced instructions replay, the value of `R5` will be modified. Therefore, in the program by checking the value of `R5`, the fault can be detected. For example, consider that an initial value of `R5` is `0x00`. At the end of the program, the expected value of `R5` is `0x0F`. However, if the replay of two instructions illustrated in Fig. 10(c) happens, (i_3, c_3) are overwritten by (i_1, c_1), and as a result the value of `R5` would be `0x0C`. In case of a replay of four instructions as illustrated in Fig. 10(c) (where (i_3, c_3, i_3, c_4) are overwritten by (i_1, c_1, i_2, c_2)), the value of `R5` would be `0x06` which would be detected.

The advantages of this detection method are: (1) easy to implement, (2) relatively small code size overhead. To clarify this, let us consider the countermeasure proposed in [9] against single skip instruction fault model. The implementation of this method includes two steps: (1) transforming the instructions into the idempotent ones, (2) duplicating all the idempotent instructions obtained in first step. It should be noticed that in the first step, the transformation of a non-idempotent instruction may result into several idempotent instruction which are further duplicated in the second step. This leads to very heavy code size overhead and longer code execution time. In addition, the process is quite complex because there is no common formula for such instruction transformation. In comparison, our detection method is quite easy to implement. The designers only need to insert a counter in the program and increase it after each instruction. There is no need to transform or revise the instruction. The protected code suffers from an overhead in both code size and execution time that are doubled (a little more than doubled in fact, taking into account the code part in charge of checking the counter).

We experimentally tested this method against LFI-induced replay instructions faults. In the following, in order to evaluate the effectiveness of the detection method, the detected faults are classified into fault in `R5` (i.e. a replay fault is detected thank to an incorrect value readback from `R5`) and other fault (fault occurs without affecting value of `R5`). The results obtained with different laser powers, while the PW was 50 `ns` are shown in Fig. 13. It can be seen that in most of the cases where faults occur, the value of `R5` is also faulted. No replay fault was observed without being detected by a faulty value of `R5`. The proposed method was proved to be effective at detecting laser-induced replay faults with a 100% success rate. It should be pointed out that the detection method only works if one of the instructions c_n (with n being 1, 2, ..) is included in the block being replayed or overwritten. In case of single instruction updating, if the laser illumination prevents updating the instruction i_n but does not prevent updating the counter c_n, the replay will be not detected because the value of the counter is correct.

VIII. CONCLUSION & PERSPECTIVES

In this paper, we reported, on experimental basis, the ability for an attacker to induce an original LFI fault model: the replay of blocks of instructions in a 32-bit MCU. These replay faults are induced by laser pulses which cause the instructions updating process of two instruction buffers at the interface of the Flash memory to fail. It leads to the replay of the already stored instructions. In addition, and as a result, the instructions failing to be stored in the buffers are overwritten by those replayed. When the target's cache is enabled, it makes it possible to replay a block of four instructions with settings relatively easy to find to ensure a fault rate of 100%. The fault injection process has then a periodicity of four clock

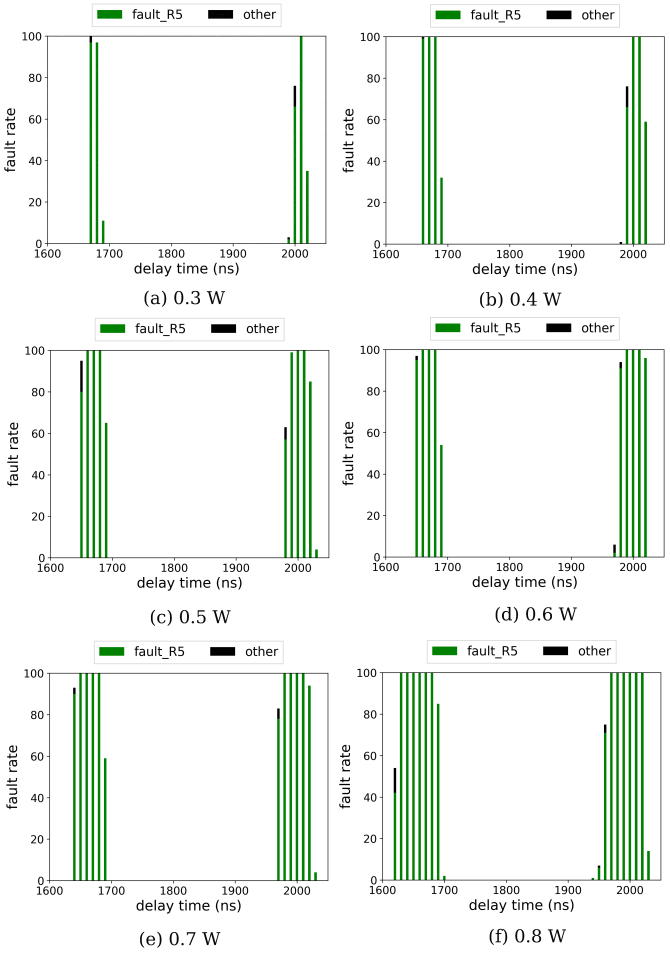

Figure 13. Experimental result using a hardware counter to detect replay fault, PW 50 ns (a) 0.3 W, (b) 0.4 W, (c) 0.5 W, (d) 0.6 W, (e) 0.7 W, (f) 0.8 W.

cycles. Increasing the laser PW makes it possible to repeat the process and to replay successively the same block of replayed instructions while overwriting several instructions in a row (we ascertained this behavior for up to twenty instructions in a row and draw the assumption that this number can be increased further similarly to what is reported in [4] for the instruction skip fault model). This phenomenon was observed with the target's cache enable or disabled with a difference in the granularity of the number of replayed faults that drops to two for the latter case (which was also very helpful to analyze and understand the underlying process and its relation with the two buffers).

This laser-induced new fault model appears to be very powerful for an attacker as it makes it possible to overwrite several instructions in a row, mimicking the instruction skip fault model of [4] that was only demonstrated on an 8-bit MCU. It may be used to erase (i.e. overwrite) a whole section of a program for attack purposes. A perspective work would be to study if the ability to replay some instruction while erasing other may be linked to an increased threat w.r.t. to repeated fault skips.

We also introduced a first simple software detection method for the replay fault model. It uses a register as a hardware counter that is incremented by dedicated *canary* instructions inserted in-between instructions to be protected. The register content is checked against a predicted value after the protected section of the program. Hence, any difference in the counter value due to the overwriting of a *canary* instruction shall be detected if a replay fault is induced. This method comes at a high price, it is associated to an overhead in both code size and execution time that are approximately doubled. However, our experiments proved its effectiveness.

Other, more elaborate countermeasures are prospective research work.

ACKNOWLEDGMENT

This work was partly funded by the SPARTA project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 830892.

REFERENCES

[1] 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 *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 105–114. IEEE, 2011.

[2] Arthur Beckers, Josep Balasch, Benedikt Gierlichs, Ingrid Verbauwhede, Saki Osuka, Masahiro Kinugawa, Daisuke Fujimoto, and Yuichi Hayashi. Characterization of em faults on atmega328p. In *International Symposium on Electromagnetic Compatibility. IEEE*, 2019.

[3] 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. *IACR Cryptology ePrint Archive*, 2018:1042, 2018.

[4] Jean-Max Dutertre, Timothé Riom, Olivier Potin, and Jean-Baptiste Rigaud. Experimental analysis of the laser-induced instruction skip fault model. In *Nordic Conference on Secure IT Systems*, pages 221–237. Springer, 2019.

[5] Donald H Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *IEEE Transactions on Nuclear Science*, 12(5):91–100, 1965.

[6] Microchip Technology Inc. SAM D21/DA1 Family. https://ww1.microchip.com/downloads/en/DeviceDoc/SAM_D21_DA1_Family_DataSheet_DS40001882F.pdf.

[7] ARM Limited. Arm®v6-m architecture reference manual. In *ARM Limited*. ARM Limited, 2017.

[8] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 77–88. IEEE, 2013.

[9] Nicolas Moro, Karine Heydemann, Emmanuelle Encrenaz, and Bruno Robisson. Formal verification of a software countermeasure against instruction skip attacks. *Journal of Cryptographic Engineering*, 4(3):145–156, 2014.

[10] PULSCAN. Innovative Solutions for Testing and Failure Analysis. https://www.pulscan.com/pages/pulsbox.php/.

[11] Lionel Riviere, Zakaria Najm, Pablo Rauzy, Jean-Luc Danger, Julien Bringer, and Laurent 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)*, pages 62–67. IEEE, 2015.

[12] Cyril Roscian, Alexandre Sarafianos, Jean-Max Dutertre, and Assia Tria. Fault model analysis of laser-induced faults in sram memory cells. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 89–98. IEEE, 2013.

[13] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *International workshop on cryptographic hardware and embedded systems*, pages 2–12. Springer, 2002.