symmetry

MDPI

Article

Instruction-Fetching Attack and Practice in Collision Fault Attack on AES

Huilong Jiang [1,2], Xiang Zhu [1,2,*] and Jianwei Han [1,2,*]

[1] State Key Laboratory of Space Weather, National Space Science Center, Chinese Academy of Sciences, Beijing 101499, China
[2] School of Astronomy and Space Science, University of Chinese Academy of Sciences, Beijing 100049, China
* Correspondence: zhuxiang@nssc.ac.cn (X.Z.); hanjw@nssc.ac.cn (J.H.)

Abstract: A Fault Attack (FA) is performed mainly under the data corruption model and poses a threat to security chips. Instruction corruption can enact the same purpose at the behavioral level, which is produced by interfering with the instruction system. Laser Fault Injection (LFI) on program memory during the instruction-fetching process, which we refer to as an instruction-fetching attack, is studied in this paper. This process bears the ability to produce a controllable instruction-fetching fault. Our work shows the implementation of the attack and its specific application case on an 8-bit microcontroller. The main contributions of this paper include: (1) We have mapped the sensitive areas precisely to the faulted instructions via laser injection and implemented controllable instruction tampering. (2) A Collision Fault Attack (CFA) scheme based on instruction-fetching fault is proposed. (3) The impacts of the faulted instructions are fully explored, including the influence on subsequent operations and key recovery. (4) The fault mechanism of the on-chip Flash is further investigated. Instruction-fetching fault means that the controller fetches a tampered instruction from the program memory under external interference, which likely gives rise to an invalid or incorrect operation. The experiment confirms that this specific fault can induce particular types of faults that are different to realize, e.g., the byte-fault model in CFA. The realization, application and mechanism of instruction-fetching fault are discussed in detail.

Keywords: collision fault attack; instruction-fetching fault; AES; laser injection; microcontroller; Flash

Citation: Jiang, H.; Zhu, X.; Han, J. Instruction-Fetching Attack and Practice in Collision Fault Attack on AES. *Symmetry* **2022**, *14*, 2201. https://doi.org/10.3390/sym14102201

Academic Editors: Kuo-Hui Yeh and Christos Volos

Received: 31 August 2022
Accepted: 13 October 2022
Published: 19 October 2022

1. Introduction

Security chips are usually a microcontroller or an ASIC embedded with a cryptographic algorithm, which is crucial for trusted computing and data confidentiality. Since the Side Channel Attack (SCA) was proposed, the security of physical implementation becomes as important as the cryptographic algorithm itself. There are various types of SCAs, some are passive attacks such as Timing Attack [1], Electromagnetic (EM) Analysis [2] and Power Analysis Attack (PA) [3], and others are active attacks such as the FA [4]. FA means that the attacker performs Fault Injection (FI) on the device to obtain more information to recover the key. FI can be realized by a voltage or clock glitch [5,6], an EM pulse [7], or a laser beam [8]. Internal temporary memory, such as register file, Flip-Flop, and Static Random Access Memory (SRAM) is usually the priority target. In addition, FI can also be achieved by interfering with the logic circuit or instruction system, and this is the mechanism behind voltage or clock glitches. Recent studies have demonstrated researchers' interests in instruction corruption, and some focus on disturbing the instruction-fetching process of a chip. As a result, the normal instruction to be stored in the Instruction Register (IR) may be replaced with an illegal instruction. A variety of methods have been proven to be capable of instruction corruption, including voltage glitches [6], clock glitches [9], EM pulses [10,11] and laser injection [12–15]. Research shows that laser injection could have an effect with better controllability, as shown in Table 1, rather than clock glitch and EM pulse.

The effects of an instruction-fetching fault may be diverse: if the tampered instruction is invalid, the effect is equivalent to instruction skip; if the tampered instruction is illegal, it may lead to an unpredictable result.

Table 1. Comparison between several instruction-fetching attack methods.

Methods	Temporal & Spatial Accuracy	Performance of Instruction Tampering
Voltage glitch	High & Low	Not controllable [6]
Clock glitch	High & Low	Not controllable [9]
Electromagnetic pulse	High & Ordinary	Ordinary [10,11]
Laser injection	High & High	Controllable [12], ours

In this paper, we execute an instruction-fetching attack on an 8-bit AVR microcontroller, characterize the fault model of the on-chip Flash, and propose a practical CFA scheme on AES. For this purpose, firstly, AVR assembly instructions are used to implement specific operations (i.e., AddRoundKey), and possible target instructions are analyzed. In this model, a critical evaluation surrounds whether the introduced faulted byte can lead to a collision, and then several instructions are chosen as the target. At this stage, the instruction skip model is the primary consideration. The CFA model was challenging to execute in past attack practice since the traditional fault injection model is predominantly random bit or byte changes. Instruction operations are primarily in words or bytes, which is more suitable for certain particular scenarios. Next, we conduct the attack on a microcontroller to demonstrate the actual effect of the instruction-fetching attack. Prior to this, we first investigate the fault characteristics of the Flash during data reading under laser injection. An accuracy mapping between the attack-sensitive areas and the faulted readback data is established with the help of the small laser spot and high-precision test bench. With fault characterization of the on-chip Flash, controllable instruction tampering is finally realized. The execution process and results brought by the tampered content of the instruction bear significant differences, and key recovery is valid for some content. Therefore, the faulted instructions are classified according to the impacts. Finally, we provide further discussion regarding the fault mechanism. At present, there exists no conclusive point of view to explain the physical mechanism of the instruction-fetching fault. Based on the existing works, we have discussed more sensitive area distributions to explain the current fault model.

The rest of this article is organized as follows: Section 2 summarizes some previous works and the progress of related research. Section 3 briefly introduces the AES algorithm and details the proposed CFA model. In Section 4, We characterize the laser injection fault model during data transmission of the on-chip Flash memory. Section 5 introduces results of the instruction-fetching attack on the microcontroller and discusses it in different modes. In Section 6, the mechanism behind the laser-induced fault model of Flash is discussed. Finally, Section 7 provides a summary of the paper.

2. Related Works

An FA based on physical implementation requires defining an abstract fault model of erroneous device behaviors, and fault analyses need to consider both the mathematic structures of the algorithm and the status of the working device [13]. FA usually consists of two steps: attack practice and key recovery. In the first step, attackers require the ability to make a slight alteration to the device to perform fault injection. It is necessary to observe the status of the device and collect data in this phase. In the next step, the key is calculated through a specific fault analysis model. These analysis models include Differential Fault Attack (DFA) [16], Key-Expansion Attack [17], Invalid Fault Attack [18], Algebraic Fault Attack (AFA), Persistent Fault Attack (PFA) [19], CFA [20,21], etc. DFA is a traditional and efficient fault analysis model, and currently the most commonly used model. DFA against

the eighth round of AES requires only one ciphertext and a small search volume to recover the complete key, representing the most efficient FA model at present [22].

Most of the above attack models are based on operand errors, which can be reflected in the tampering of operands directly or the indirect disturbing of the logical circuit and instruction system. The latter can be divided into *instruction-executing fault* and *instruction-fetching fault* as shown below.

- *Instruction-executing fault*: This kind of fault is induced by disturbing the normal execution of instructions, e.g., making the executed operation not complete normally by underpowering or providing a clock glitch, which is widely used in actual attacks. The core idea of this technology is that reducing the supply voltage of the device can temporarily extend the key-path of the circuit, or the rising edge of the clock glitch will cause an error data to be stored in the Flip-Flop in advance [23,24].
- *Instruction-fetching fault*: This article focuses only on this type of fault. The core idea is to modify the program data to be stored in the instruction register by introducing interference to the on-chip Flash during instruction-fetching, and then the microcontroller executes a wrong instruction. There is no uniform name for this type of attack. Some works only focused on Instruction Skip [25,26], and some other works summarized it as Instruction Replacement [12] or Instruction Corruption [13]. To describe this concept more accurately, the term *instruction-fetching fault* is applied in this paper.

Microcontrollers widely adopt the Harvard architecture, i.e., the design of independent access to program flow and data flow. For attackers, active attacks can be used to threaten secret data stored in SRAM or interfere with instructions accessed from the Flash memory. Early research on instruction-fetching fault mainly relied on voltage glitches or clock glitches. The primary purpose at this point was to skip specific instructions [6]. Ref. [9] showed that controlling the generation timing of glitches can lead to different types of instruction-fetched data errors, although the changes are limited. In later research, EM pulse and laser injection were applied. In [10], Moro et al. found that certain instructions were more sensitive to EM pulses. Ref. [11] demonstrated that the accuracy of the EM attack could be improved by pipeline architecture analysis and equipment spatial and temporal parameter improvement. Another study investigated the performance of EM pulse on an ARM processor, and subsequently found that multiple instruction skip could also be achieved [27]. In [28], Kumar et al. set an ATmega328p microcontroller as the target, and their research showed that laser injection can lead to bit-reset faults of instruction flow. Our research also shows a similar result regarding this feature. Ref. [13] carries out instruction corruption attacks on a 32-bit microcontroller platform and proposes a possible explanation for the fault model of Flash. In [12], researchers implement fully controllable instruction replacement and put forward a corresponding explanation for the fault mechanism by irradiating sense amplifier in Flash memory with laser. To achieve this goal, extremely expensive equipment is needed. In addition to the single instruction, a series of laser pulses can be used to skip multiple instructions [26]. Khuat et al. reported an instruction replay fault model caused by buffer update failure [29]. Their other work characterized the instruction transmission pipeline with the faulted instructions [14]. Ref. [30] reports instruction skip attack on a more complex ARM cortex A9 Microprocessor, and another work is also carried out in the mobile phone processor to obtain higher permissions [31]. One of the thorny problems of an instruction-fetching attack concerns the production of controllable instruction tampering, especially for an EM pulse even with the improved scheme proposed in [28]. A better fault model can be obtained by laser injection, but little work has established accurate mapping between the attack area and the fault model. Fault characterization depends on a special target architecture, and requires a high cost [12]. In this paper, we introduce interference into the execution of AVR Flash access instruction using an irradiating laser to finally achieve this goal. The fault characterization and the controllable instruction tampering scheme will be detailed.

3. Attack Model

In this part, we detail the CFA model based on instruction skip, an application case of the instruction-fetching fault proposed in this paper.

3.1. The Advanced Encryption Standard

AES is a symmetric encryption algorithm based on the Substitution-Permutation Network (SPN) structure, which was released in 2001 to replace the Data Encryption Standard (DES) [32]. The operand data named State Matrix is fixed to 128 bits, and the key size is variable at 128, 192 or 256 according to the security requirement. Further, the iterations are 10, 12 or 14, respectively. The operations of AES are built in the finite field, and the basic operations of each round are SubBytes (SBox operation), ShiftRows, MixColumns (except the last round) and AddRoundKey. AES has high security and performance and is widely applied in various confidentiality scenarios.

3.2. CFA Model Based on Instruction Skip

The collision of the first AddRoundKey operation can be used to recover the complete key. Let $p_i (0 \leq i \leq 15)$ and k_i denote ith byte plaintext and key, respectively, then the result c_i of this operation satisfies:

$$p_i \oplus k_i = c_i \tag{1}$$

If c_i is set to a known or fixed value v_i, the key can be calculated by

$$k_i = p'_i \oplus v_i \tag{2}$$

It needs to traverse from 0 to 255 to find the p'_i that leads to a collision of ciphertexts. If $i = 0$, collision detection of ciphertexts is shown in Figure 1, and $c'_0 = v_0$ is satisfied when the collision occurs.

The crucial operation is to set c_i to a fixed value. Listing 1 shows a possible implementation of AddRoundKey operation. Register R20 and R21 are loaded with plaintext byte and key byte, respectively, and they are refreshed with the operations. One possible speculation is that tampering with the instructions LD, EOR or ST may result in a fixed value of the operation.

Listing 1. An assembly case of the AddRoundKey operation on the AVR platform.

```
#Initialize the Y, Z and R24 register.
  MOVW  R28,&m
  MOVW  R30,&k
  CLR   R24
loop:
#Load a plaintext byte indirectly addressed by the Y register
  LD    R20,Y
#Load a key byte indirectly addressed by the Z register
  LD    R21,Z
#Perform the XOR operation
  EOR   R20,R21
#Write result back to the original address
  ST    Y,R20
#Address offset
  ADIW  R28,0x01
  ADIW  R30,0x01
#Iterative round control
  ADIW  R24,0x01
  CPI   R24,0x10
  BRNE  loop
```

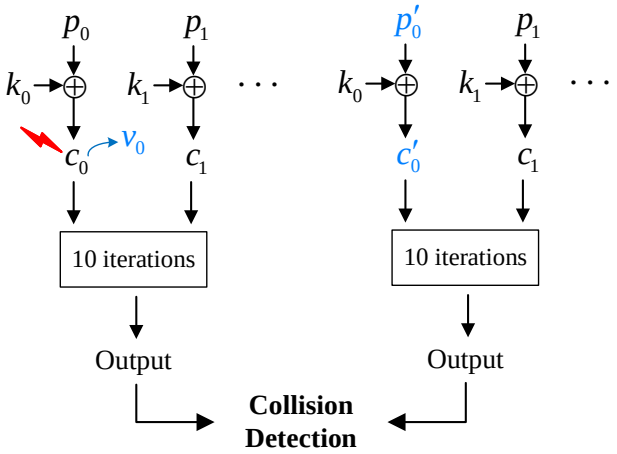


Figure 1. Collision detection of the first AddRoundKey operation of AES.

3.2.1. Skiping the LD Instruction

Assuming that the instruction LD is skipped, the initial value (maybe zero or another value) is still stored in the R21 register, and then an incorrect value is written to the address pointed to by Y after EOR and ST operation. Suppose the initial value of register R21 is v_0, then the register value will be updated with the operation process, as shown in Table 2. To calculate the first key byte, one should to traverse v_0 from 0 to 255 and make collision detection for each value, and then the corresponding key byte can be calculated by Equation (2), respectively. p'_0 is the byte value satisfying collision. The second key byte can be calculated in a similar manner, which is equivalent to replacing v_0 with k_0. The calculation scheme of the complete key is shown in Equation (3), and it should be noted that there are 255 schemes for the combination of all key bytes. Algorithm 1 presents a key recovery algorithm based on the skip of the LD instruction.

$$\begin{aligned} k_0 &= p_0 \oplus v_0 \oplus p'_0 \\ k_1 &= p_1 \oplus k_0 \oplus p'_1 \\ k_2 &= p_2 \oplus k_1 \oplus p'_2 \\ &\cdots \\ k_{15} &= p_{15} \oplus k_{14} \oplus p'_{15} \end{aligned} \tag{3}$$

Table 2. An update of R20 and R21 with the operation process by skipping LD.

Index of the Key Byte	Register	Instruction Skip	LD	EOR	ST
1st	R20	Before	p_0	$p_0 \oplus k_0$	$p_0 \oplus k_0$
		After	p_0	$p_0 \oplus v_0$	$p_0 \oplus v_0$
	R21	Before	k_0	k_0	k_0
		After	v_0	v_0	v_0
2nd	R20	Before	p_1	$p_1 \oplus k_1$	$p_1 \oplus k_1$
		After	p_1	$p_1 \oplus k_0$	$p_1 \oplus k_0$
	R21	Before	k_1	k_1	k_1
		After	k_0	k_0	k_0
...	...	...	...	...	...
15th	R20	Before	p_{15}	$p_{15} \oplus k_{15}$	$p_{15} \oplus k_{15}$
		After	p_{15}	$p_{15} \oplus k_{14}$	$p_{15} \oplus k_{14}$
	R21	Before	k_{15}	k_{15}	k_{15}
		After	k_{14}	k_{14}	k_{14}

Algorithm 1: An algorithm of key calculation for LD skipping model.

Input: $p = (p_0, p_1, \ldots, p_{15})$, $c = (c_0, c_1, \ldots, c_{15})$
Output: $k = (k_0, k_1, \ldots, k_{15})$
Some descriptions: p, c and k represent plaintext, ciphertext and key, respectively. $En(p)$ indicates normal encryption operation, and $En'(p|p_i)$ indicates the fault encryption targeting the ith byte.
for $0 \leq i \leq 15$ **do**
    for $0 \leq p'_i \leq 255$ **do**
        if $En(p|p_i \leftarrow p'_i) = En'(p|p_i)$ **then**
            $tmp_i \leftarrow p_i \oplus p'_i$;
            break;
for $0 \leq v_0 \leq 255$ **do**
    $k_0 \leftarrow v_0 \oplus tmp_0$;
    for $1 \leq i \leq 15$ **do**
        $k_i \leftarrow k_{i-1} \oplus tmp_i$;
    $k \leftarrow (k_0, k_1, \ldots, k_{15})$;
    if *k is correct* **then**
        out k;
        break;

3.2.2. Skippping EOR or ST Instruction

It can be seen that skipping EOR or ST can produce a similar effect, and as a result, the address pointed to by the Y register still holds the original plaintext byte. That is, for the ith plaintext byte, the value held in the original address is p_i after the ST operation. The key can be calculated in a similar way as before. The attacker needs to traverse p'_i from 0 to 255 and use $k_i = p_i \oplus p'_i$ to calculate the key byte k_i when a collision occurs. Algorithm 2 shows the key recovery algorithm under the fault model, and it can be seen that this scheme is more efficient than the previous model.

Algorithm 2: An algorithm of key calculation for EOR or ST skipping model.

Input: $p = (p_0, p_1, \ldots, p_{15})$, $c = (c_0, c_1, \ldots, c_{15})$
Output: $k = (k_0, k_1, \ldots, k_{15})$
for $0 \leq i \leq 15$ **do**
    for $0 \leq p'_i \leq 255$ **do**
        if $En(p|p_i \leftarrow p'_i) = En'(p|p_i)$ **then**
            $k_i \leftarrow p_i \oplus p'_i$;
            break;
out $k \leftarrow (k_0, k_1, \ldots, k_{15})$;

3.3. Summary

In this section, we describe the CFA model based on instruction skip, and in the assembly case of the model, three instructions are set as targets. In the following sections, the attack will be implemented on a microcontroller.

4. Experimental Setting

4.1. The Pulsed Laser Fault Injection Platform

The experimental platform we designed, as shown in Figure 2, mainly includes the following elements:

- A pulsed laser and optical path system
- A 3D mobile station
- Synchronous control system (responsible for timing control of different elements)

- An oscilloscope
- A control PC

The laser can produce 1064 nm infrared light, and the width of a single pulse is about 15 ps. Narrow pulse width can ensure that the energy of a single pulse is absorbed by Si material and reduces the influence of the thermal effect. We can trigger it at any time with a 5V square wave, and it will emit a laser beam after a delay of about 1 µs. The maximum accuracy of the 3D mobile station is 0.1 µm and the mobile station can be programmed to realize automatic scanning.

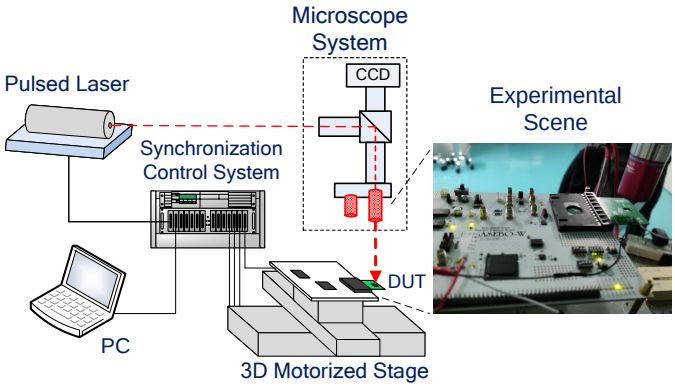


Figure 2. The experimental system of pulsed laser fault injection.

4.2. Test Chip

Our target chip is an 8-bit AVR microcontroller ATmega163L, designed as a Harvard architecture with a 2-stage fetch-execute pipeline. It contains 1K bytes of data memory (SRAM), 16K bytes of program memory (Flash) and 32 general registers (R0: R31). AVR controller allows the CPU to concurrently use both the data bus and instruction bus in a clock cycle, therefore, it can complete the current operation and obtain the next instruction in one clock cycle [9].

For the convenience of communication and power consumption, as shown in Figure 3, we design it into a smart card matching the SASEBO-W development board and set its working frequency to 3.57 MHz. In Figure 3, some important modules of the microcontroller have been highlighted, and the focus of the attack is the Flash. An oscilloscope is used to monitor the execution state of instruction in real-time, and the laser trigger signal produced by a Spartan-6 FPGA in the SASEBO-W board can also be monitored, as shown in Figure 4.

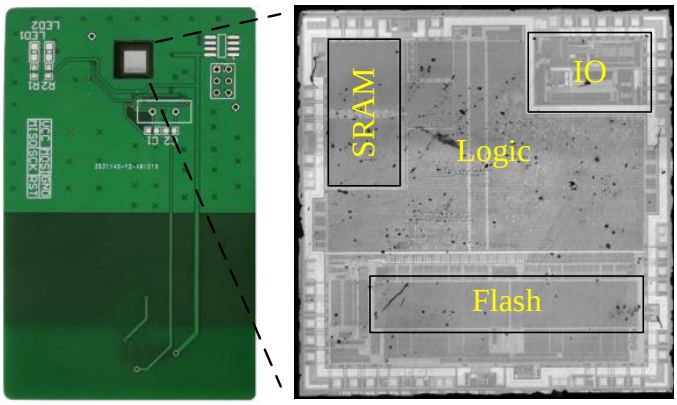


Figure 3. The ATmega163L microcontroller and its back-side layout. The back photo is taken via infrared imaging, and certain important parts have been highlighted.

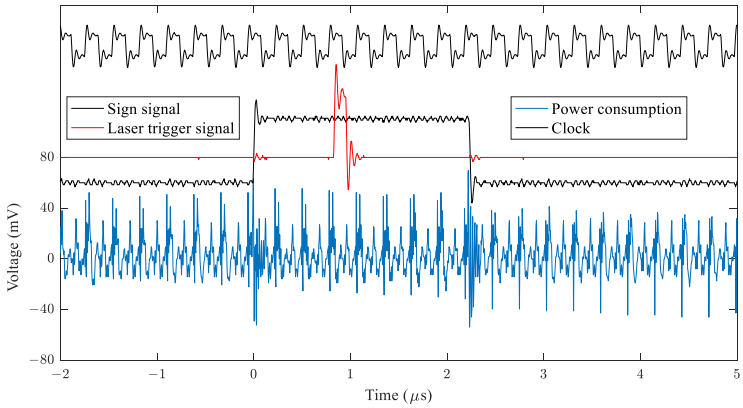


Figure 4. The power curve of instruction execution (blue) and the laser trigger signal (red). The oscilloscope is used to detect the working state, so as to accurately control the laser signal timing.

5. Experimental Result

In this section, we target a special instruction LPM to clarify the data organization features and the fault models of Flash, and then we conduct specific attack practices to several instructions.

5.1. Laser-Induced Flash-Data-Accessing Error

In order to clarify the influence induced by the laser of the on-chip Flash, we chose a specific instruction, LPM (Load form Program Memory), which is applied to load one-byte data from program memory pointed to by the Z register (stores 16-bit address data) into any of general registers in the MCU, as shown in Table 3. Since the instruction is associated with Flash access, it will help to characterize the failure model.

Table 3. Opcodes and three usage formats of the LPM instruction.

Format	Opcodes
LPM	1001 0101 1100 1000
LPM Rd *, *Z*	1001 000d dddd 0100
LPM Rd, *Z+*	1001 000d dddd 0101

* Rd is one of the general purpose registers.

5.1.1. Target the LPM instruction

Figure 5a shows the power trace of the LPM execution process represented in Listing 2, and the red pulse signal reveals the trigger signal of the laser. Since there is a delay of about 1 µs between the pulse signal and the laser emission, the pulse signal needs to be set in advance. The LPM requires three clock cycles to execute, as shown in Figure 5a, and needs an additional clock cycle to process the address data, that is, the data to be saved in the Z register. As mentioned above, the process of instruction fetching is completed in the last clock cycle of the previous instruction execution. Therefore, the Flash read-operation is performed in the second clock cycle. To verify this conjecture, we calculate the correlation between the Hamming Weight of storage data and the power consumption when executing the LPM instruction. The selected memory address ranged from 0x00 to 0xFF and a total of 600 power consumption points were sampled. Let h_i denote the Hamming Weight of data stored in ith address and t_i denotes the corresponding power consumption, the jth $(1 \leq j \leq 600)$ column correlation coefficient r_j satisfies:

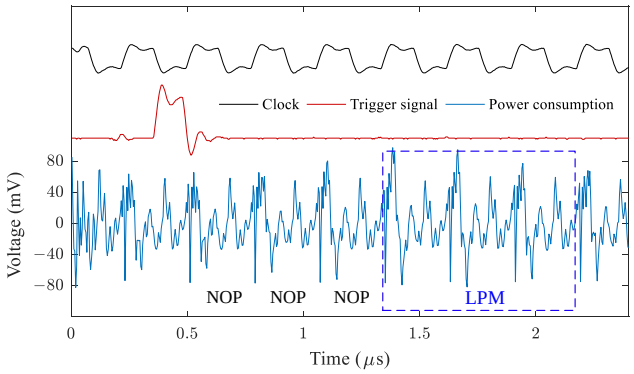
$$r_j = corr(H, T) = \frac{\sum_{i=1}^{n} (h_i - \bar{h})(t_i - \bar{t})}{\sqrt{\sum_{i=1}^{n} (h_i - \bar{h})^2} \cdot \sqrt{\sum_{i=1}^{n} (t_i - \bar{t})^2}} \quad (4)$$

where $H = [h_1, h_2, \cdots, h_n]$ and $T = [t_1, t_2, \cdots, t_n]$.

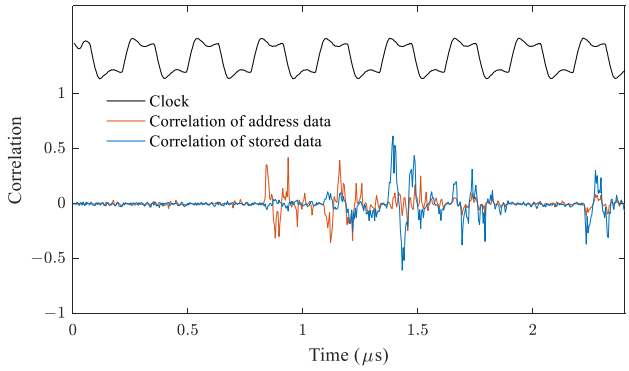
As shown in Figure 5b, the correlation of address data is also calculated. Subsequently, the result shows consistency with our expectations. The blue curve indicates the correlation of stored data. It can be seen that there is a strong correlation in the second clock cycle of the LPM instruction, indicating that storage data are loaded from Flash memory into the register at this time. The orange curve indicates the correlation of address data, and there are several spikes in the previous and first clock cycle of the LPM instruction. This may indicate that the two spikes should correspond to the two-stage of LPM instruction fetching.

Listing 2. Test code for attacking the LPM instruction.

```
#Pull up the trigger signal
  SBI PORTB, 7
#Delay 3 NOPs to allow sufficient time for laser trigger
  NOP
  NOP
  NOP
#Execute the LPM instruction and increment the Z register by 1
  LPM R12, Z+
#Pull down the trigger signal
  CBI PORTB, 7
```



(**a**)



(**b**)

Figure 5. Implementing the attack on the LPM instruction by analyzing timing and power leakage. (**a**) Reveals the power curve of instruction execution and the laser trigger signal. The timing of the LPM instruction should be confirmed before the attack. (**b**) Shows the correlation analysis of stored data and address data with power consumption. These peaks are associated with points in time for instruction and data processing.

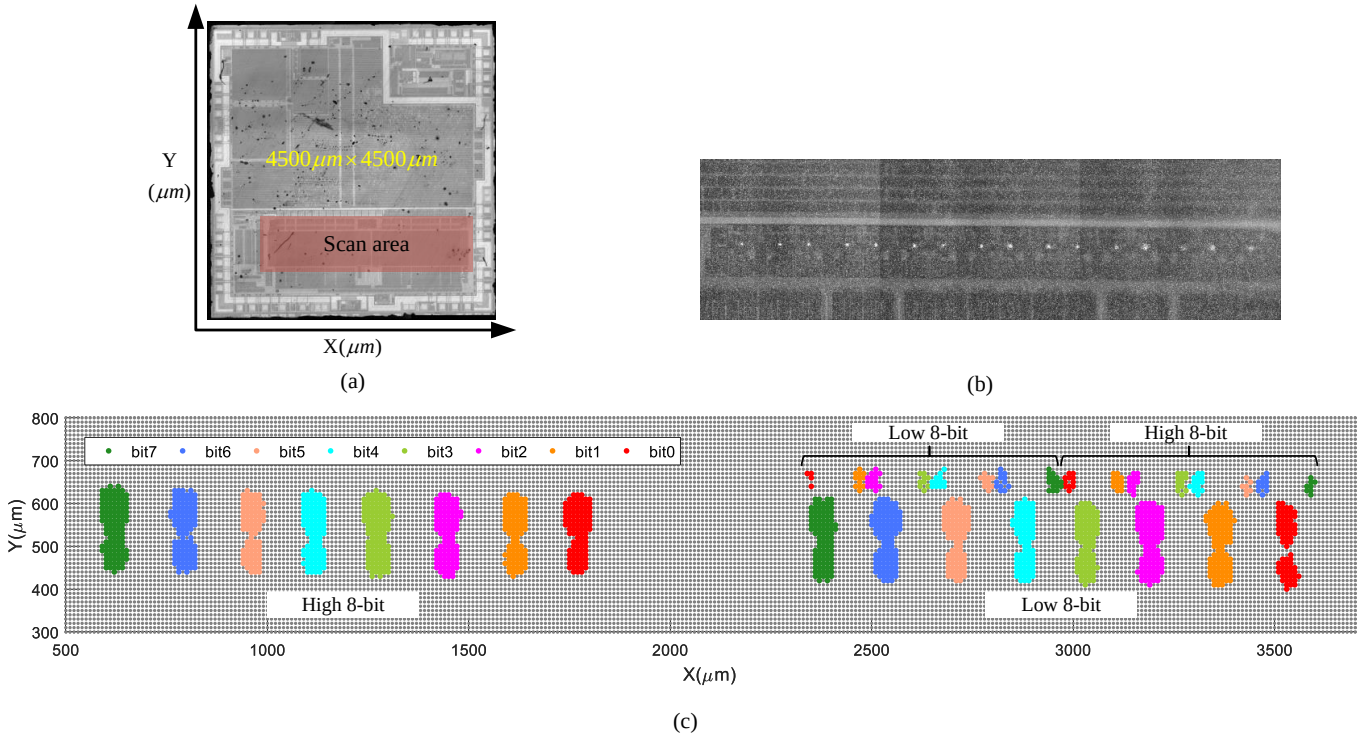


Figure 6. Experimental results of laser attack during Flash data-accessing operation. (**a**) Shows the scanning scheme, and the bottom left corner is set as the coordinate origin. (**c**) Shows a scanning area covering the whole Flash, and the sensitive areas of all bits have been highlighted. In addition, (**b**) also shows the photon leak imaging of the Flash area. The 16 bright spots may correspond to 16 sensitive positions on the upper right of (**c**), respectively.

Our scanning scheme is shown in Figure 6a. The size of the chip is about 4500 × 4500 (μm^2), and we set the lower left corner as the coordinate origin. We select a suitable area to cover the whole Flash, and then scan the area at 10 μm interval, as shown in Figure 6c. After storing specific data in an address space, LPM is executed to read back these data, and the laser is triggered during the reading operation. By detecting whether the readback data match the known value, we finally obtain the attack-sensitive areas of all data.

5.1.2. Sensitive Areas of Laser Injection

Figure 6 shows all the sensitive areas that lead to readback data error. Through careful analysis of the experimental results, we offer the following conclusions:

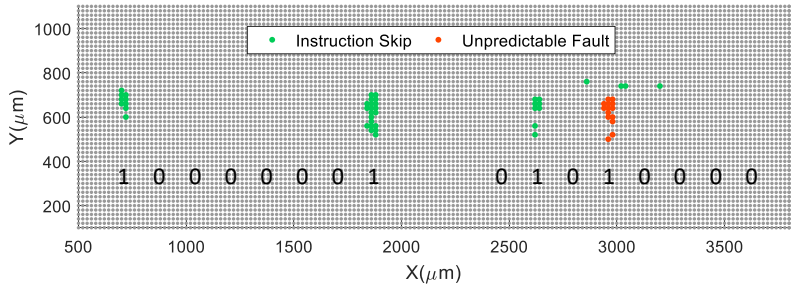
- **All error data bits are changed from 1 to 0 (bit-reset fault), and no bit changes from 0 to 1.** This conclusion is consistent with those from previous works, and the asymmetric fault model related to the specific structure and encoding scheme of Flash [13,28].
- **Sensitive areas of laser injection of the same bit of all words are distributed in clusters in the same area.**
- **There are 16 highlighted areas in the upper right corner of the Flash, which also correspond to 16-bit sensitive areas, respectively.** The attack effect against these areas is consistent with the above. We further track the photon radiation of the Flash region and find 16 conspicuous photon leakage points in the upper right corner, as shown in Figure 6b, which indicates that there are frequent transistor switching operations in this area. These areas may indicate the control register location in the Flash read–write control circuit.

- **These faults are not permanent.** These faults only correspond to the error of read-out data and do not change data stored in Flash cells.
- **There is a large effective time window for triggering the laser, which is about 840ns.** This may be due to the long duration of Flash data access.

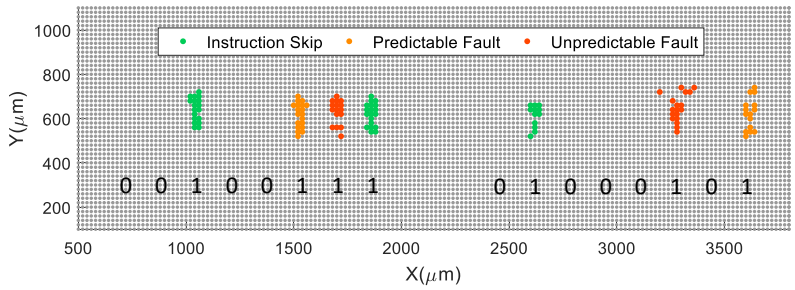
Our further experiments show that these fetched instructions could be tampered with by focusing the laser to a special bit-1 position.

5.2. Attacks on LD, EOR and ST Instructions

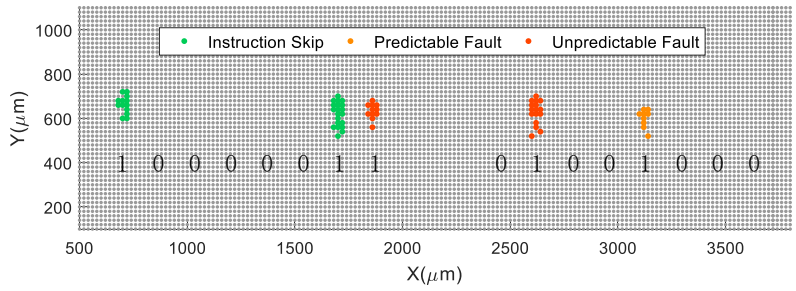
We attempted to focus the laser in special areas that correspond to the sensitive areas of bit-1 of instructions, and experiments show that not all bit-reset faults can produce the excepted error ciphertexts. Only modifying special bits can produce the appropriate fault. Figure 7 shows the actual attack effect against three instructions, and every sensitive area corresponds to a bit-1 of the instruction. In order to detail the effect of instruction corruption, we divide it into three levels: Instruction Skip, Unpredictable Fault and Predictable Fault, as shown in Tables 4–6.



(**a**)



(**b**)



(**c**)

Figure 7. Sensitive areas of laser-induced instruction-fetching fault for three instructions: (**a**–**c**) shows the sensitive areas of the LD, EOR and ST instructions separately. Due to the scanning granularity, some sensitive areas may not be fully covered. Each sensitive area corresponds exactly to a bit-1. Note that not all bits of each instruction can produce effective modifications by laser injection.

Table 4. Laser-induced Instruction-Fetching Fault of LD.

Modified Bit-1 Position	Opcodes	Operation Instruction	Effect
None	1000 0001 0101 0000	*LD R21, Z*	Normal
1st	0000 0001 0101 0000	*MOVW R10, R0*	Skipping
2nd	1000 0000 0101 0000	*LD R5, Z*	Skipping
3rd	1000 0001 0001 0000	*LD R17, Z*	Skipping
4th	1000 0001 0100 0000	*LD R20, Z*	Unpredictable

Table 5. Laser-induced Instruction-Fetching Fault of EOR.

Modified Bit-1 Position	Opcodes	Operation Instruction	Effect
None	0010 0111 0100 0101	*EOR R20, R21*	Normal
1st	0000 0111 0100 0101	*CPC R20, R21*	Skipping
2nd	0010 0011 0100 0101	*AND R20, R21*	Predictable
3rd	0010 0101 0100 0101	*EOR R20, R5*	Unpredictable
4th	0010 0110 0100 0101	*EOR R4, R21*	Skipping
5th	0010 0111 0000 0101	*EOR R16, R21*	Skipping
6th	0010 0111 0100 0001	*EOR R20, R17*	Unpredictable
7th	0010 0111 0100 0100	*EOR R20, R20*	Predictable

Table 6. Laser-induced Instruction-Fetching Fault of ST.

Modified Bit-1 Position	Opcodes	Operation Instruction	Effect
None	1000 0011 0100 1000	*ST Y, R20*	Normal
1st	0000 0011 0100 1000	*FMUL R20, R16*	Skipping
2nd	1000 0001 0100 1000	*LD R20, Y*	Skipping
3rd	1000 0010 0100 1000	*ST Y, R4*	Unpredictable
4th	1000 0011 0000 1000	*ST Y, R16*	Unpredictable
5th	1000 0011 0100 0000	*ST Z, R20*	Predictable

5.2.1. Instruction Skip

Instruction skip is our expected fault and will generally not lead to an additional impact on other operations. For the instruction *LD R21, Z* (opcode is 0x8150), after tampering with the highest bit, it becomes *MOVM R10, R0* (0x0150). Since the latter instruction operates on other unrelated registers instead of R20 or R21, the effect at this time is equivalent to that when the instruction is skipped. The key recovery schemes, in this case, were mentioned earlier.

5.2.2. Unpredictable Fault

This kind of fault is caused by executing an illegal instruction and tampering with related registers. The intermediate data stored in registers could not be determined, and the calculation result may also be random at this time. If the fifth bit of the instruction *LD R21, Z* is tampered with, the instruction executed actually is *LD R20, Z*. Since the key byte is not properly loaded into the R21 register, the next operation *EOR R20, R21* will produce an uncertain result, and the final ciphertext may also be an uncertain value. The key cannot be recovered from this case.

5.2.3. Predictable Fault

Although this fault does not behave like an instruction skip, the result can be predicted. For example, after tampering with the least bit of *EOR R20, R21* (0x2745), it becomes *EOR R20, R20* (0x2744), which causes the value loaded into the R20 register and written back to Y address to be zero. Key recovery for this case will be discussed in detail in the next section.

5.3. Key Recovery for the Predictable Fault

Here, we focus on key recovery for the predictable faults. A predictable fault can cause the register to store in a value that can be inferred and key recovery can be achieved by adopting a similar strategy of instruction skip. The experiment shows three kinds of predictable faults, which will be discussed in the following part.

5.3.1. EOR R20, R21 → AND R20, R21

Similar to the above, for the ith plaintext byte p_i and its collision byte p'_i satisfy:

$$p_i \wedge k_i = p'_i \oplus k_i \tag{5}$$

To solve this equation, one should traverse k_i from 0 to 255, but there will be multiple candidate values that satisfy this. Therefore, attackers should perform multiple fault injections to determine the unique correct key byte. Key recovery can be achieved by following these steps: (1) Generate a plaintext randomly and then perform encryption with fault injection. (2) Collect the fault ciphertext, traverse the relevant plaintext byte from 0 to 255 and then perform normal encryptions to detect a collision. (3) Calculate the candidate space of the key byte according to Equation (5). (4) Check the size of the candidate space: If the size has not decreased to 1, return to step (1); Otherwise, output the unique key byte. The main part of the calculation process can be described by Algorithm 3.

Algorithm 3: Key recovery algorithm for the fault instruction *AND R20, R21.*

Input: A certain number (N) of ciphertexts $\{p^0, p^1, \ldots, p^{N-1}\}$, and $p^i = (p^i_0, p^i_1, \ldots, p^i_{15}), 0 \le i \le N-1$, is a 16-byte ciphertext
Output: The jth key byte $k_j (0 \le j \le 15)$

```
Λ_k ← {0, 1, ..., 255};
for 0 ≤ i ≤ N − 1 do
    Γ ← ∅;
    for 0 ≤ g ≤ 255 do
        if En(p^i | p^i_j ← g) = En'(p^i | p^i_j) then
            Γ ← Γ ∪ {g};
            break;
    Λ_k ← Λ_k ∩ Γ;
    if the size of Λ_k decreases to 1 then
        out k_j ← the only element in Λ_k;
        break;
```

We further investigated the average number of fault injections required for key recovery. The possible key byte values (0~255) are fully considered and loaded in turn into the chip for 100 times of encryptions with laser injections in each experiment separately. The sequence of the 100 fault pairs of plaintexts and ciphertexts is used for key recovery. After selecting a plaintext as the starting position for the calculation, the subsequent plaintexts are also used to calculate the candidate key bytes until the correct one is obtained, and then the number of plaintexts required is recorded. The offset after each complete calculation is 1, and the sequence connection is constructed as a cycle. Finally, 100 calculation results for each key byte and 256×100 for all possible key bytes are obtained.

The running speed of encryptions exceeds four times per second on the microcontroller, and the experiment took more than 1.5 h. Figure 8 shows the attack result of the first-key-byte position, and the average number of candidate key bytes will decrease to 1 with about 4.54 (5 for actual) fault encryptions. Hence, approximately 73 fault encryptions are required to recover the complete key.

5.3.2. EOR R20, R21 → EOR R20, R20

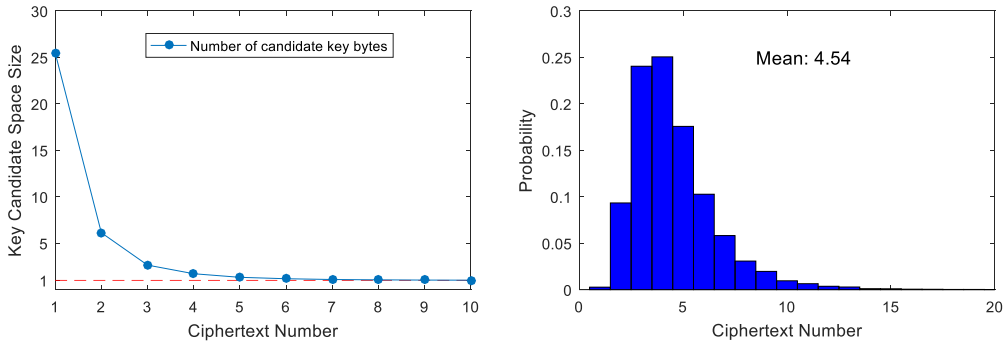
In this case, the result of the XOR operation is zero, and the value to be stored in R20 after the ST operation is also zero. Here, the key byte and collision plaintext byte satisfy $p'_i \oplus k_i = 0$, that is, $k_i = p'_i$. The number of candidate keys will decrease to 1 after performing 16 fault injections.

5.3.3. ST Y, R20 → ST Z, R20

This situation is very interesting because the address pointed to by the Z register stores the key and the key may possibly be modified at this time. We assume that this type of fault will occur and the correct key can only be recovered after a restart. It can be inferred that p_i is still stored in the address pointed to by Y, and the key stored in the address Z becomes $p_i \oplus k_i$. When a collision occurs, the following shall be satisfied:

$$p'_i \oplus p_i \oplus k_i = p_i \tag{6}$$

that is, $k_i = p'_i$ when collision occurs. Therefore, the complete key can be recovered after 16 fault injections.



(**a**) (**b**)

Figure 8. The number of ciphertexts to obtain the first key byte under the fault model. The figure shows the attack results on all possible key byte values (0~255) by performing a total of 25,600 fault injections: (**a**) Shows that the key byte candidate space decreases rapidly with the increase in available fault ciphertexts produced by fault injections. As the number of fault injections increases, only the unique correct key bytes will be retained. (**b**) Shows the probability of the number of ciphertexts required to obtain the correct key byte under multiple experiments. The number of plaintexts or ciphertexts corresponding to a higher blue bar is more likely to reduce the candidate space to 1, and the average number required is about 4.54.

6. Discussion on the Fault Model of Flash

The experiment shows the bit-reset faults cluster, which is most likely related to the Flash shared bit-lines [15]. Ref. [13] pointed out that these transistors connected to the raised bit line could be sensitive to laser irradiation. However, we have observed some sensitive positions that were not in the storage array area. Here, we further analyze the fault model based on the experiment, mainly discussing the possible sensitive areas under laser injection.

Figure 9 shows the structure of the NOR Flash storage array. By applying a voltage to the control gate, the working status of the cell transistor depends on the number of electrons in the floating gate. When the Flash executes read-operation, the bit-line and the word-line are pre-charged to a high level, and the storage bit is generally determined by detecting the current from the bit-line. The transistor in the normal OFF state failed to produce a current in the bit-line, but the laser-induced carriers lead to a photocurrent that provides compensation. The compensation of the photocurrent no longer demonstrates efficacy when the transistor is in the ON state. The reverse biased PN-junctions are the

sensitive nodes of laser injection [33]. The photocurrent is transmitted along the bit-line to the sense amplifier, where it can then also be amplified hundred-fold.

According to the experimental results, three sensitive nodes could be inferred as the laser-sensitive areas. The first sensitive nodes are the transistors belonging to storage cells. The bit-line is directly connected to the drain of any storage transistor, therefore, the photocurrent generated in the drain region of each transistor could pull down the level of the whole bit-line, whether there are electrons in the floating gate or not. The second sensitive nodes could be in the column decoder. The local and global bit-line architecture is widely used in the Nor Flash to achieve high-access performance [34]. These switch transistors concerned with the local bit-lines in the column decoder could become sensitive areas for laser injection because the photocurrent generated in the nodes will also be collected and amplified by the sense amplifier. The third sensitive nodes could be the transistors belonging to the sense amplifier, and the input or inside disturbances of the photocurrent could also change the logic output. Note that the sensitive areas in the upper right corner of Figure 6c may correspond to those nodes. The reverse biased PN-junctions of the transistor exist in the green-marked regions in Figure 9b, and these nodes are the sensitive areas of laser injection. All these faults seem to correspond to a transition from the OFF state to the ON state. This asymmetric fault model of the Flash is determined by its storage mechanism of the floating gate structure, and is also a temporary fault with a short time window of attack.

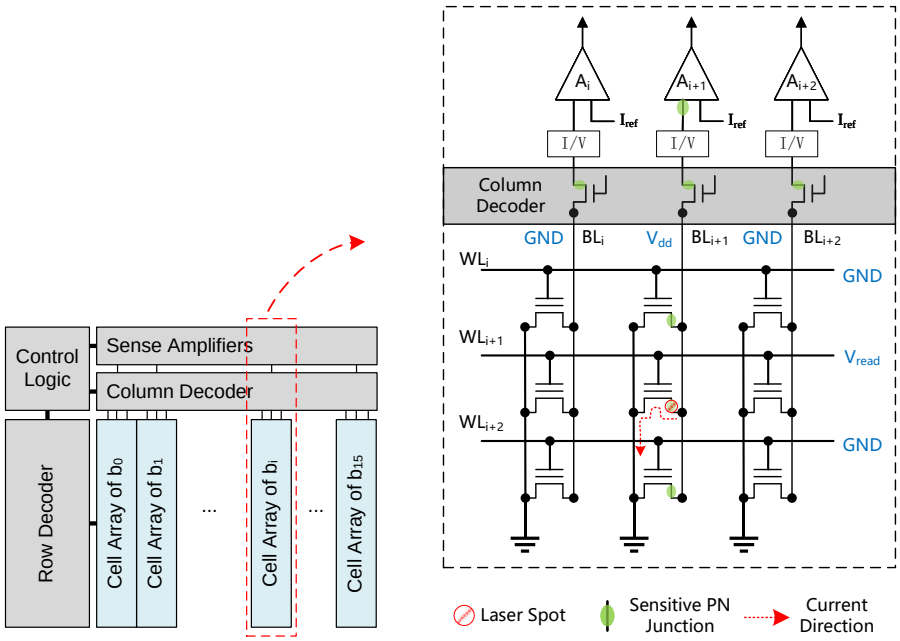


Figure 9. Flash memory structure and its sensitive areas under laser injection when accessing the *i*th bit cell. (**a**) Shows the main modules of NOR Flash, including storage array, decoder, sensitive amplifier and control circuit. (**b**) Shows the sensitive areas, and a total of three areas are included: areas near the storage cells, decoder and sense amplifier.

In the experiment, we adopt the scanning scheme to characterize the fault model of the Flash memory and attempt to establish an attack template for microcontrollers of the same type or series. In this process, these sensitive areas can be mapped to the faulted readout data, which will reveal the data organization characteristics of the Flash memory. Meanwhile, the instruction system of the microcontroller is also analyzed (as a white box), and then the impacts of fault instructions will also be mapped to the sensitive areas. Thus, in an actual attack, focusing the laser beam on a specific location can lead to an expected fault for the same type of equipment.

7. Conclusions

In this paper, we have demonstrated our contribution to instruction-fetching Attack. These works mainly focus on the characterization of fault models, the application of instruction-fetching fault in the CFA model and the exploration of fault mechanisms. In order to establish an accurate physical mapping between fault characteristics and sensitive areas, we target a data transfer instruction (LPM), employ a small-size laser spot and appropriate parameter settings to interfere with the on-chip Flash memory, and finally obtain a controllable fault model. The instruction system mainly operates in words or bytes, and thus, it is more suitable for the CFA model, which is critical for special scenarios such as [35]. The storage structure of Flash memory results in an asymmetric fault model embodied as the bit-reset fault in this work. The fault mechanism is discussed, and the areas of the clustered fault are revealed. However, the process from instruction fetches to the execution pipeline can still be further divided into several phases, and additional mechanisms still remain to be explored [14]. In the next work, we will also explore additional potential applications of the instruction-fetching fault.

Author Contributions: Conceptualization, H.J. and X.Z.; methodology, H.J. and X.Z.; software, H.J.; validation, H.J. and X.Z.; formal analysis, H.J. and X.Z.; resources, X.Z. and J.H.; data curation, X.Z. and J.H.; writing—original draft preparation, H.J. and X.Z.; writing—review and editing, H.J., X.Z. and J.H.; supervision, X.Z. and J.H.; project administration, X.Z. and J.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Foundation Enhancement Planning Technology Field Fund Project (2021JCJQJJ0926).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kocher, P.C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 1996; Springer: Berlin/Heidelberg, Germany, 1996; pp. 104–113.
2. Gandolfi, K.; Mourtel, C.; Olivier, F. Electromagnetic analysis: Concrete results. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Paris, France, 14–16 May 2001; Springer: Berlin/Heidelberg, Germany, 2001; pp. 251–261.
3. Kocher, P.; Jaffe, J.; Jun, B. Differential power analysis. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 15–19 August 1999; Springer: Berlin/Heidelberg, Germany, 1999; pp. 388–397.
4. Boneh, D.; DeMillo, R.A.; Lipton, R.J. On the importance of checking cryptographic protocols for faults. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Konstanz, Germany, 11–15 May 1997; Springer: Berlin/Heidelberg, Germany, 1997; pp. 37–51.
5. Schmidt, J.M.; Herbst, C. A practical fault attack on square and multiply. In Proceedings of the 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography, Washington, DC, USA, 10–10 August 2008; pp. 53–58.
6. Choukri, H.; Tunstall, M. Round reduction using faults. *FDTC* **2005**, *5*, 13–24.
7. Schmidt, J.M.; Hutter, M. Optical and em Fault-Attacks on Crt-Based Rsa: Concrete Results. In Proceedings of the Austrochip 2007, 15th Austrian Workhop on Microelectronics, Graz, Austria, 11 October 2007; pp. 61–67.
8. Skorobogatov, S.P.; Anderson, R.J. Optical fault induction attacks. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Redwood Shores, CA, USA, 13–15 August 2002; Springer: Berlin/Heidelberg, Germany, 2002; pp. 2–12.
9. Balasch, J.; Gierlichs, B.; Verbauwhede, I. An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, Nara, Japan, 28 September 2011; pp. 105–114.
10. Moro, N.; Dehbaoui, A.; Heydemann, K.; Robisson, B.; Encrenaz, E. Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, 20 August 2013; pp. 77–88.
11. Trabelsi, O.; Sauvage, L.; Danger, J.L. Characterization of electromagnetic fault injection on a 32-bit microcontroller instruction buffer. In Proceedings of the 2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Kolkata, India, 15–17 December 2020; pp. 1–6.

12. Sakamoto, J.; Fujimoto, D.; Matsumoto, T. Laser-induced controllable instruction replacement fault attack. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **2020**, *103*, 11–20. [CrossRef]
13. Colombier, B.; Menu, A.; Dutertre, J.M.; Moëllic, P.A.; Rigaud, J.B.; Danger, J.L. Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. In Proceedings of the 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 5–10 May 2019; pp. 1–10.
14. Khuat, V.; Danger, J.L.; Dutertre, J.M. Laser Fault Injection in a 32-bit Microcontroller: from the Flash Interface to the Execution Pipeline. In Proceedings of the 2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC), Milan, Italy, 17 September 2021; pp. 74–85.
15. Menu, A.; Dutertre, J.M.; Rigaud, J.B.; Colombier, B.; Moellic, P.A.; Danger, J.L. Single-bit laser fault model in NOR flash memories: analysis and exploitation. In Proceedings of the 2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC), Milan, Italy, 13 September 2020; pp. 41–48.
16. Biham, E.; Shamir, A. Differential fault analysis of secret key cryptosystems. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 1997; Springer: Berlin/Heidelberg, Germany, 1997; pp. 513–525.
17. Kim, C.H.; Quisquater, J.J. New differential fault analysis on AES key schedule: Two faults are enough. In Proceedings of the International Conference on Smart Card Research and Advanced Applications, London, UK, 8–11 September 2008; Springer: Berlin/Heidelberg, Germany, 2008; pp. 48–60.
18. Blömer, J.; Seifert, J.P. Fault based cryptanalysis of the advanced encryption standard (AES). In Proceedings of the International Conference on Financial Cryptography, Guadeloupe, French West Indies, 27–30 January 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 162–181.
19. Zhang, F.; Zhang, Y.; Jiang, H.; Zhu, X.; Bhasin, S.; Zhao, X.; Liu, Z.; Gu, D.; Ren, K. Persistent fault attack in practice. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**, *2020*, 172–195. [CrossRef]
20. Hemme, L. A differential fault attack against early rounds of (triple-) DES. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Cambridge, MA, USA, 11–13 August 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 254–267.
21. Blömer, J.; Krummel, V. Fault based collision attacks on AES. In Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography, Yokohama, Japan, 10 October 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 106–120.
22. Tunstall, M.; Mukhopadhyay, D.; Ali, S. Differential fault analysis of the advanced encryption standard using a single fault. In Proceedings of the IFIP International Workshop on Information Security Theory and Practices, Heraklion, Crete, Greece, 1–3 June 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 224–233.
23. Selmane, N.; Guilley, S.; Danger, J.L. Practical setup time violation attacks on AES. In Proceedings of the 2008 Seventh European Dependable Computing Conference, Kaunas, Lithuania, 7–9 May 2008; pp. 91–96.
24. Bhasin, S.; Selmane, N.; Guilley, S.; Danger, J.L. Security evaluation of different AES implementations against practical setup time violation attacks in FPGAs. In Proceedings of the 2009 IEEE International Workshop on Hardware-Oriented Security and Trust, San Francisco, CA, USA, 27 July 2009; pp. 15–21.
25. Yuce, B.; Ghalaty, N.F.; Santapuri, H.; Deshpande, C.; Patrick, C.; Schaumont, P. Software fault resistance is futile: Effective single-glitch attacks. In *Proceedings of the 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*; IEEE: Toulouse, France, 2016; pp. 47–58.
26. Dutertre, J.M.; Riom, T.; Potin, O.; Rigaud, J.B. Experimental analysis of the laser-induced instruction skip fault model. In Proceedings of the Nordic Conference on Secure IT Systems, Santa Barbara, CA, USA, 16 August 2016; Springer: Berlin/Heidelberg, Germany, 2019; pp. 221–237.
27. Elmohr, M.A.; Liao, H.; Gebotys, C.H. EM fault injection on ARM and RISC-V. In Proceedings of the 2020 21st International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 25–26 March 2020; pp. 206–212.
28. Kumar, D.S.; Beckers, A.; Balasch, J.; Gierlichs, B.; Verbauwhede, I. An in-depth and black-box characterization of the effects of laser pulses on atmega328p. In Proceedings of the International Conference on Smart Card Research and Advanced Applications, Montpellier, France, 12–14 November 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 156–170.
29. Khuat, V.; Dutertre, J.M.; Danger, J.L. Analysis of a laser-induced instructions replay fault model in a 32-bit microcontroller. In Proceedings of the 2021 24th Euromicro Conference on Digital System Design (DSD), Palermo, Italy, 1–3 September 2021; pp. 363–370.
30. Vasselle, A.; Thiebeauld, H.; Maouhoub, Q.; Morisset, A.; Ermeneux, S. Laser-induced fault injection on smartphone bypassing the secure boot. In Proceedings of the 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), Taipei, Taiwan, 25 September 2017; pp. 41–48.
31. Gaine, C.; Aboulkassimi, D.; Pontié, S.; Nikolovski, J.P.; Dutertre, J.M. Electromagnetic fault injection as a new forensic approach for SoCs. In Proceedings of the 2020 IEEE International Workshop on Information Forensics and Security (WIFS), New York, NY, USA, 6–11 December 2020; pp. 1–6.
32. Daemen, J.; Rijmen, V. Reijndael: The Advanced Encryption Standard. *Dr. Dobb's J. Softw. Tools Prof. Program.* **2001**, *26*, 137–139.
33. Roscian, C.; Sarafianos, A.; Dutertre, J.M.; Tria, A. Fault model analysis of laser-induced faults in sram memory cells. In Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, 20 August 2013; pp. 89–98.

34. Richter, D. Fundamentals of non-volatile memories. In *Flash Memories*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 5–110.
35. Amiel, F.; Villegas, K.; Feix, B.; Marcel, L. Passive and active combined attacks: Combining fault attacks and side channel analysis. In Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), Vienna, Austria, 10 September 2007; pp. 92–102.