IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2020, No. 2, pp. 172–195. DOI:10.13154/tches.v2020.i2.172-195

Persistent Fault Attack in Practice

Fan Zhang[1,2,4], Yiran Zhang[1,3,4], Huilong Jiang[5], Xiang Zhu[5], Shivam Bhasin[6], Xinjie Zhao[7], Zhe Liu[2,8(✉)], Dawu Gu[9] and Kui Ren[1,4]

[1] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[2] State Key Laboratory of Cryptology, P.O.Box 5159, Beijing, China
[3] College of Information Science & Electronic Engineering, Zhejiang University, Hangzhou, China
[4] Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies, Hangzhou, China
[5] Chinese Academy of Sciences, Beijing, China
[6] Nanyang Technological University, Singapore
[7] Institute of North Electronic Equipment, Beijing, China
[8] Nanjing University of Aeronautics and Astronautics, Nanjing, China
[9] Shanghai Jiaotong University, Shanghai, China
fanzhang@zju.edu.cn;zhe.liu@nuaa.edu.cn

Abstract. *Persistence fault analysis* (PFA) is a novel fault analysis technique proposed in CHES 2018 and demonstrated with rowhammer-based fault injections. However, whether such analysis can be applied to traditional fault attack scenario, together with its difficulty in practice, has not been carefully investigated. For the first time, a persistent fault attack is conducted on an unprotected AES implemented on ATmega163L microcontroller in this paper. Several critical challenges are solved with our new improvements, including (1) how to decide whether the fault is injected in SBox; (2) how to use the maximum likelihood estimation to pursue the minimum number of ciphertexts; (3) how to utilize the unknown fault in SBox to extract the key. Our experiments show that: to break AES with physical laser injections despite all these challenges, the minimum and average number of required ciphertexts are 926 and 1641, respectively. It is about 38% and 28% reductions of the ciphertexts required in comparison to 1493 and 2273 in previous work where both fault value and location have to be known. Furthermore, our analysis is extended to the PRESENT cipher. By applying the persistent fault analysis to the penultimate round, the full PRESENT key of 80 bits can be recovered. Eventually, an experimental validation is performed to confirm the accuracy of our attack with more insights. This paper solves the challenges in most aspects of practice and also demonstrates the feasibility and universality of PFA on SPN block ciphers.

Keywords: PFA · Fault Injection · SRAM · MLE · AES · PRESENT

1 Introduction

Fault attack (FA) on cryptographic systems is an active attack with two phases. In Phase one, the adversary disturbs the operation of the target device, which is known as *fault injection* (FI). In Phase two, he analyzes the faulty ciphertexts to recover the key, which is known as *fault analysis*.

Fault injection techniques can include changing the power supply voltage, the external clock, or the temperature, etc, which are low-cost but low-precision. Laser is one of the most popular methods due to its precision. Laser injection was first developed in the field of radiation resistance. The specific manifestation of *Single Event Effect* (SEE) [WM62] is that the incident high-energy particles can generate a large number of ionization charges inside the chip and then generate current pulses, which may lead to changes of storage

Received: 2019-10-15 Accepted: 2019-12-15 Published: 2020-03-02

information or errors in logic operation [BSH75]. In 1965, Habing first proposed a method to simulate the SEE of heavy ions by using pulsed laser [Hab65]. Later, laser has been used in the field of security evaluation as a fault injection method. The laser fault injection was first proposed by P. Skorobogatov and J. Anderson in 2002 [SA02], which targets at the 68 byte *Static Random-Access Memory* (SRAM) of a microcontroller (μC). In addition to SRAM, the registers of μC are also vulnerable to the laser [CLMFT14].

After fault analysis was firstly published by Boneh et al. [DDL97] in 1996, a huge amount of related work about fault analysis has been carried out. *Differential fault analysis* (DFA) is one of the most well-studied methods which was proposed by Biham and Shamir in 1997 [BS97]. It is a very powerful analysis method and has been successfully applied to block ciphers such as DES [BS97], AES [PQ03, DLV03, KQ08, TMA11], PRESENT [WW10] etc. In CHES 2007, *ineffective fault analysis* is proposed by Clavier et al. to exploit ineffective injections. They show that ineffective faults, i.e., faults that have no effect at all, are informative. Later in 2013, *statistical fault analysis* (SFA) is proposed to exploit statistics of faulty ciphertexts [FJLT13]. In CHES 2018, *statistical ineffective fault analysis* (SIFA) is proposed to combine IFA and SFA [DEK+18]. They exploit statistics of ciphertexts when the fault is ineffective to recover the key. Later Dobraunig et al. show that SIFA can break masked AES with fault countermeasures [DEG+18].

Most of the previous fault analyses are based on transient faults, which means these can require an adversary to perform several injections ranging from two to millions. In CHES 2018, *persistent fault analysis* (PFA) was proposed by Zhang et al. to take advantage of persistent faults [ZLZ+18]. Under the proposed model of persistent faults, a fault targeting long term constants in memory like Sbox can be exploited with only one injection. PFA was proposed to break SPN block ciphers, and it can bypass some countermeasures like module redundancy. In their work [ZLZ+18], the persistent fault is induced through rowhammer techniques and it persists in the DRAM of general processors. In [PZRB19], Pan et al. shows that PFA can break higher-order masking countermeasures at any masking order with only one fault injection. In [MBD+19], Menu et al. demonstrates PFA by electromagnetic fault injections on data transfers on microcontroller. However, these attacks are based on the assumption that the adversary knows the location and value of the persistent fault injected by the attacker.

In this paper, we demonstrate a practical persistent fault attack on an unprotected AES-128 cipher on ATmega163L microcontroller without the knowledge about the fault location and fault value. The main contributions of this work include:

- We demonstrate the first practical persistent fault analysis in the traditional laser-based fault attack setting. The target is a common microcontroller used in many existing works, showing that PFA is not a difficult practice to achieve.
- We propose methods to verify whether the desired persistent fault was injected in the SBox or not.
- We propose an improved analysis technique for PFA to handle cases when both fault value and location within a SBox are unknown.
- We extend PFA on the lightweight block cipher PRESENT, which has a small 4×4 SBox that is difficult to target and analyze with persistent fault.
- We investigate the mechanism, distribution, and impacts for bit flips in SRAM.

The rest of the paper organised as follows. Sec.2 gives some background. An overview of our attack is described in Sec.3. Sec.4 and 5 elaborate our fault injection and analysis method, respectively. Sec.6 extends our attack to PRESENT cipher. Sec.7 shows a more thorough investigation on the bit flips in SRAM, and finally we conclude the paper in Sec.8.

2 Background

In this section, we recall general background on target microcontroller, AES cipher and laser fault injection on microcontrollers.

2.1 ATmega163L Microcontroller

Microcontrollers (μC) are widely used for endpoint encryption in various commodities, such as commercial IC cards, usb-keys, etc. A typical μC mainly consists of CPU(usually containing ALUs, controllers and registers), RAMs, ROM/FLASHs and I/O peripherals. In this paper, an 8-bit commercial microcontroller, ATmega163L from Microchip [Mic03] is used for demonstration. It is with 1K bytes SRAM, 16K bytes Flash ROM and 4MHz maximal frequency. The TQFP-44 package version was selected for this work.

2.2 AES-128 and S-Box Implementation

The Advanced Encryption Standard (AES) is a block cipher that published as a standard for symmetric encryption by NIST in 2001. AES has three versions. In this paper, we focus on the AES-128 where the block size and key size are all 128 bits, i.e., 16 bytes.

During encryption, the 16-byte plaintext is copied into a 4×4 matrix called state. Then it goes through 10 rounds, and the final state is copied into ciphertext. Each round consists of four major operations: (1) **SubBytes**. It is a simple table lookup to the so-called SBox. Each byte is substituted by the corresponding byte in SBox. (2) **ShiftRows**. (3) **MixColumns**. (4) **AddRoundKey**. The state is XORed with round keys which are generated by **KeyExpansion** function with the master key. Note that KeyExpansion is an invertible function, which means getting any of round keys is equivalent to extracting the master key. Besides, there is an additional AddRoundKey at the beginning of encryption and the MixColumns operation is skipped in the last round.

Two types of AES implementation are frequently used. The first one is SBox implementation which uses a compact lookup table of 256 bytes. The InvSbox used in decryption also has 256 bytes. The other one is called TBox implementation which combines multiple operations into one substitution. In this paper, we consider the SBox implementation.

2.3 Laser Fault Injection to μC and SRAM

Semiconductors are sensitive to light when photons have sufficient energy to free bound electrons. A reverse PN junction can be turned on when collecting these additional charges, which could lead to an upset or other faults. A pulsed and focused laser can be injected into some micron-scale cells of a chip within nanoseconds and it is possible to create specific and accurate faults with a desired output.

Laser injection on a large scale integrated circuit may cause different effects, depending on the energy and injection location of the chip. Some effects as latch-ups (a type of short circuit which disrupts functionality and even leads to destruction due to overcurrent) are destructive and may not be proper for fault analysis. Some effects as upsets (a type of state change that happens in memory or register) are soft-errors, suffering from some data or instruction errors while the functionality and structure of the chip are maintained. As laser fault injection implemented in appropriate time and position, various effects including register or memory modification, key zeroing, and instruction bypass can be achieved.

For user data storage in μC, SRAMs are critical for the validity and integrity of the data. SRAM is deployed for data storage in μC because it has faster read-write speed than DRAM due to its cell structure. However, most components of μC such as SRAMs or registers are sensitive to laser and could be an attack target.

3 Overview of Persistent Fault Attack on SRAM

A general overview of the proposed attack at a high level is described in this section. The attack flow is dedicated to SRAM in this paper, showing the feasibility of the so-called persistent fault attack with existing equipment, technologies and analysis methods. Details of fault injection and fault analysis will be addressed in Sec.4 and Sec.5, respectively.

3.1 Fault Model

The target of PFA is a serialised software implementation running on the μC of AES-128 with one common SBox for all bytes and rounds. The fault model can be summarized as follows: (1) Before the injection, the data of SBox is already in SRAM and not updated for several encryption, ex. loaded during a boot-up sequence; (2) Before the encryption, the adversary can inject faults into the SRAM region of the micorcontroller; (3) The injected faults are persistent, i.e., they will not be refreshed during the encryption, and faulty elements of SBox will be used in the encryption; (4) The injected faults are at byte level. For a single injection, only one byte is altered; (5) The adversary can control the input plaintexts and collect output ciphertexts.

3.2 Core idea

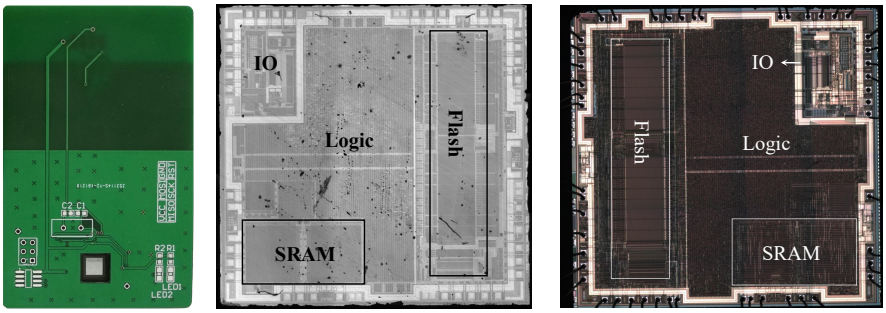
The security application together with the AES cipher is compiled and burned into the Flash of the microcontroller. Once the target μC is powered on, some variables and data will be either loaded from Flash or pre-configured with initial values directly in SRAM. The adversary has normal expertise of launching a fault attack, which generally includes two phases. In Phase one, there is possibly a quite long time window between the power-on of the device and the start of cipher execution, which gives enough time for the adversary to setup the laser. Then the laser pulses are repeatedly performed until a laser pulse inject a so-called persistent fault into SBox. In Phase two, the adversary collects several ciphertexts for random plaintexts encrypted by AES-128 operation using the faulty SBox. Persistent fault analysis can be utilized to break the cipher. However, it requires quite some adjustment in practical attacks.

3.3 Phase One: Online Fault Injection

Stage 1: Decapsulation of μC. Decapsulation of ATmega163L in our experiment is done by third-party partners. For the plastic package of the chip, the epoxy resin and other packaging materials on the back need to be removed before the experiment. Fig.1a shows the card that is the target of the experiments. The PCB board is designed with a small window to expose the backside of μC which is already decapsulated.

Stage 2: Detection of SRAM. Since the target will be SRAM, it is interesting to know where the physical SRAM region is located. Fig.1b shows the actual μC layout information we captured with an infrared camera after the decapsulation from the backside. In order to visualize the inside much more clearly, an auxiliary picture in Fig.1c is shown in comparison which is taken after a frontside decapsulation. In Fig.1c, the main components are labeled. SRAM and Flash storage arrays are well organized and their regions are relatively easy to distinguish. As shown in the left part of Fig.1c, two flash arrays are divided into 16 sectors. Comparing with the 16KB flash, the SRAM of μC has only 1024 bytes, its region is relatively smaller and located in the right bottom. In this μC, the SRAM is divided into two areas which can be recognized by direct observations on Fig.1b.

Stage 3: Injection to SRAM. A high precision laser facility is used to inject byte level faults. Laser parameters such as the energy must be carefully designed according to the μC. With proper parameters, the laser has the capability of injecting persistent faults



(a) Smartcard **(b)** Back layout **(c)** Front layout

Figure 1: The backside of decapsulated μC with its layouts.

into the SRAM. However, as the adversary has no access to the SRAM, two challenges must be addressed. The first challenge for the adversary is to detect and verify, if the laser injection created a fault in the SBox or not. The second challenge is the location and value of the fault are unknown. Lack of such information makes the analysis difficult.

Stage 4: Encryption without synchronization. In traditional fault attacks, in order to corrupt the target intermediate value of a cipher, the adversary needs to determine the precise timing, i.e., in which round and in which operation the laser beam must be shot. This needs a very tightly-coupled synchronization between the timing of cipher encryption and fault injection, which needs high expertise to achieve. Thanks to the relaxed time window, the persistent fault is already injected in Stage 3 and well prepared for subsequent steps. Then the adversary can feed the plaintext to μC and start the encryption, which is under a favorable fashion that requires no synchronization with the cipher execution.

3.4 Phase Two: Offline Fault Analysis

Our fault analysis method mainly follows the pioneer work proposed in [ZLZ+18], however, lots of improvements are required to make it practical. In Persistent Fault Analysis (PFA), the fault will persist for a relatively long time, but not permanent. The persistent fault used in this paper is a modification to AES SBox in SRAM. It will last over different encryptions and will not disappear until a reset of μC. The offline PFA exploits the statistical distribution of the faulty ciphertexts to reveal key-dependent information. However, the original PFA in [ZLZ+18] assumes that the adversary knows exactly which SBox element is corrupted and the corrupted value. In our attack scenario, both of the assumptions are not satisfied. In this paper, the PFA is improved to recover these unknown information from the ciphertexts. In addition, *Maximum Likelihood Estimation* (MLE) is used to take full information from the ciphertext distribution and reduce the attack complexity.

4 Fault Injection in Practice

Our fault injection method will be detailed in this part. The experimental setup and the process for practical injections are described in Sec.4.1, and method for solving the problems is shown in Sec.4.2.

4.1 Setup and Practical Injection

All experiments are conducted on a pulsed laser facility which is originally used for single event effect tests. It is customized to fit the fault attack on microcontrollers in this paper.

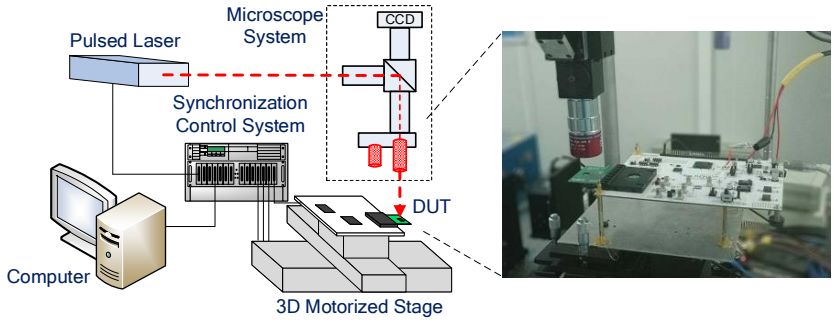


Figure 2: Schematic of pulsed laser facility for our persistent fault attack.

Fig.2 presents the schematic of the experimental setup. The facility is composed of a pulsed laser generator, a microscope system, a three-dimensional motorized stage, a synchronization control system and a computer. The device under test (DUT) consists in the ATmega163L μC whose backside is already decapsulated, as shown in Fig.1a. This DUT is mounted on the 3D motorized stage. The pulsed laser beam is collimated and focused on the backside of μC by the microscope system. Focused laser spot on this μC is imaged via the charge-coupled device (CCD). The facility is coordinated by the synchronization control system which allows synchronization among the movement of the 3D motorized stage, the shot of laser pulse, and the power on/off of μC. The computer can control the facility by sending commands to the synchronization control system which also communicates with μC via a serial port. It will return a signal to the computer after a received command is completed.

The parameters of laser must be carefully chosen. The single photon energy of the laser is set to 1.17eV in order to make it larger than the band gap of silicon. Then the energy of each laser pulse needs to be selected. As mentioned in Sec.2.3, there are three results when laser irradiations are positioned onto SRAM, including no effect, upset and latch-up. In this work, upset is the state of data corruption which can be recovered by reset, and latch-up is the state of lock which can only be restored by power off and on. Usually, the upset threshold is lower than latch-up and the proper energy can be chosen in the range between them. After repeated experiments, the upset threshold of SRAM is about $100pJ$. The latch-up threshold is $250{\sim}300pJ$, so the energy of $200pJ$ is chosen for the subsequent experiment. The laser spot size is about 2 μm which is precise enough for our μC and each laser pulse lasts about 17 ps.

With these setup, the adversary can randomly shoot laser pulses into the coarse area of the SRAM, hoping one of them can inject a fault into the SBox. After each injection, the adversary needs to determine if a real SBox fault is injected, which can be termed as an *effective injection*. For ineffective ones, the adversary can reset the μC and try to shoot laser at a different location. For effective ones, the adversary can collect faulty ciphertexts for the following analysis. The major challenge for the fault injection is how to identify the effective injections.

4.2 Identification on Effective Injections

For an effective injection, i.e., with a fault inside SBox, the ciphertext will become faulty if and only if the corrupted element is accessed during the encryption. Due to the randomized plaintexts, only part of the ciphertexts will become faulty. In other conditions, the ciphertexts will be either all correct (e.g., no data corruption) or all faulty (e.g., the round constants are corrupted). Therefore, the adversary can firstly collect a set of correct ciphertext without activating the laser to have a reference set. Then the same set of

plaintext can be used in subsequent runs. The injection will be judged as an effective one if both correct and faulty ciphertexts exist.

However, if too few ciphertexts are collected, they can be all correct or faulty even with an SBox fault, and misjudgment will happen. Meanwhile, we want to reduce the amount of required ciphertexts as little as possible. So the number of ciphertexts must be carefully evaluated and determined to balance the cost and the misjudgment rate.

For AES-128, the SBox table of 256 bytes is accessed 160 times during the encryption, and the probability that the faulty SBox is not accessed can be computed as $P_F = (1 - \frac{1}{256})^{160} \approx 53.46\%$. That means about half of the ciphertexts should be correct. With N_c ciphertexts, the probability that the ciphertexts are all correct or all wrong, i.e., the misjudgment rate P_{mj1}, can be computed as $P_{mj1} = P_F^{N_c} + (1 - P_F)^{N_c}$. The misjudgment rate drops exponentially with N_c. With 20 ciphertexts, it can be lowered to 3.86×10^{-6} which is negligible. Therefore, 20 ciphertexts will be collected after each injection to decide whether a fault is injected into the SBox or not.

5 Persistent Fault Analysis on AES in Practice

During the fault injection phase, the persistent fault is already injected and verified. In this section, we focus on phase two, where the ciphertexts are carefully analyzed in order to extract the secret key.

The original persistent fault analysis in [ZLZ+18] is a little bit away from practical, where two important factors are ignored: (1) *fault location*, i.e., which byte of SBox is injected with faults; (2) *fault value*, i.e., the fault difference between the corrupted byte and its original value. In their work, the fault location is fixed as the first element of SBox. The fault value is also known to the adversary. For example, the faulty element value is altered from 0x63 to 0x61 via rowhammering, where 0x61 is assumed to be known through a profiling process on the entire memory space [ZLZ+18].

However, as mentioned in Sec.4, in the real attacks these two factors are actually unknown to the adversary. Such information needs to be deduced from the offline cryptanalysis, which is NOT that straightforward.

5.1 Limitations of the Original PFA

The fault location, i.e. the byte index of fault, is denoted as i. The fault value is denoted as f. During the fault injection, the i^{th} byte of S-Box $S[i]$ is changed into $S'[i]$, where $S'[i] = S[i] \oplus f$. Suppose c_j, y_j, k_j denote the j^{th} byte of the ciphertext, the state before the last key addition and the last round key, respectively. Thus $c_j = y_j \oplus k_j$. [1].

Let $Pr(y_j)$ and $Pr(c_j)$ denote the probability distribution of y_j and c_j, respectively. For normal encryptions without faults, due to the avalanche effect, the probability of each value of y_j and c_j should be close to 2^{-8}. But this property would not hold if there is a fault in SBox. Thus the probability distribution for c_j can be calculated as in Eq.(1).

$$Pr(c_j = v) = \begin{cases} 0 & v = S[i] \oplus k_j \\ 2 \times 2^{-8} & v = S'[i] \oplus k_j \\ 2^{-8} & \text{otherwise} \end{cases} \tag{1}$$

Let c_j^{min} denote the value that should never appear in the specific j^{th} byte of final ciphertexts, and $c_j^{min} = S[i] \oplus k_j$. Let c_j^{max} denote the ciphertext value that should appear with doubled frequency where $c_j^{max} = S'[i] \oplus k_j$. In practice, a statistic analysis on the collected ciphertexts can be conducted to check whether such a desired probability

[1] Linear permutations such as ShiftRows in AES are simply ignored for simplification

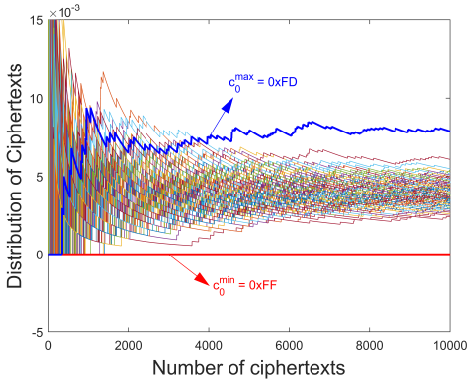


Figure 3: Exemplary PFA on AES with known fault location and value.

distribution really appears or not, which can be used to deduce either c_j^{min} or c_j^{max}, and to extract k_j directly.

Let C_j denote the distribution of the j^{th} byte among all ciphertexts. Specifically, C_j shows different counts of values that appear in N_c ciphertexts. Taking the first byte of all ciphertexts as an example, Fig.3 shows how the distribution C_0 changes with the increasing number of ciphertexts. The red curve at the bottom corresponds to c_0^{min} with a zero probability, and the blue curve on the top corresponds to c_0^{max} which converges to $\frac{2}{256}$ with enough number of ciphertexts. Both curves are apparently getting more and more distinguishable, while other curves are converged to the probability of $\frac{1}{256}$.

For any j, c_j^{min} can be easily recovered with sufficient ciphertexts since it is the only value that never appears. Then the key can be recovered as $k_j = S[i] \oplus c_j^{min}$, which shows that recovering k_j is equivalent to recovering c_j^{min} if $S[i]$ is specified.

In [ZLZ+18], a straight-forward method is proposed to recover c_j^{min}. Since c_j^{min} can never appear, they can eliminate impossible values of those candidates to reduce the search space. More specifically , $c_j^{min} \neq v$ if $Counts(c_j = v) \neq 0$, where $Counts(c_j = v)$ calculates the number of ciphertexts whose $c_j = v$. However, when pursuing as few number of ciphertexts as possible (i.e., a smaller N_c is wanted), such fault analysis has two major drawbacks, which should be overcome in order to improve original PFA.

One drawback is about c_j^{min} itself. In [ZLZ+18], the value of c_j^{min} will be verified ONLY when enough number of ciphertexts are collected. With an increasing number of ciphertexts, more and more impossible values will be identified and then removed from the candidate set for c_j^{min}. When all 255 values are identified, the adversary will be aware that there is only one possible value that remains in the candidate set, which is confirmed as the correct c_j^{min} with a 100% probability. However, the adversary does not need to wait until all impossible values are identified.

The other drawback is about the usage of c_j^{max}. In [ZLZ+18], the fault analysis relies merely on c_j^{min}, which does not take c_j^{max} into consideration. In the original PFA, c_j^{max} will converge to $\frac{2}{256}$ and c_j^{min} remains a zero probability all the time. As shown in Fig.3, the differentiation of such convergence for c_j^{max} requires more ciphertexts when compared to c_j^{min}. That is the reason why c_j^{min} is chosen for filtering those impossible values. However, an analysis on c_j^{max} can possibly assist that on c_j^{min}.

Both drawbacks can be improved with the technique called Maximum Likelihood Estimation (MLE), which can significantly reduce the number of required ciphertexts.

5.2 Improved PFA with Maximum Likelihood Estimation

The core idea of our improved PFA can be sketched as following: Similar to the original PFA, the impossible values of c_j^{min} are firstly eliminated. With sufficient ciphertexts, only one candidate of c_j^{min} will remain, and it must be the correct one. However, with a limited number of ciphertexts, several candidates may remain in the candidate set for c_j^{min}. In this situation, it is hard to decide which candidate is better to be kept if no further information is provided. Recall that the original PFA simply increases the number of ciphertexts until only one candidate remains. However, in an naive case where the fault value f is known, the corresponding c_j^{max} for each remaining candidate of c_j^{min} can be computed as in Eq.(2):

$$c_j^{max} = S'[i] \oplus k_j = (S'[i] \oplus S[i]) \oplus (S[i] \oplus k_j) = f \oplus c_j^{min} \tag{2}$$

As c_j^{max} should appear more frequently, the candidate for c_j^{min} with the most frequent c_j^{max} will be the correct one with a higher probability. The constraint in such naive case can be further relaxed in a further discussion, which will be the real case where neither fault location nor fault value is known.

Let us give an example to facilitate understanding. Considering a set C_j of 1020 ciphertexts:

$$C_j: \quad Counts(c_j = v) = \begin{cases} 8 & v = 0 \\ 4 & v = 1, 2, ..., 253 \\ 0 & v = 254, 255 \end{cases} \tag{3}$$

Both 254 and 255 are candidates for c_j^{min}. Assuming $f = 255$, if $c_j^{min} = 254$, the corresponding $c_j^{max} = 254 \oplus 255 = 1$; If $c_j^{min} = 255$, the corresponding $c_j^{max} = 255 \oplus 255 = 0$. Since the value 0 appeared eight times in C_j while the value 1 only appeared four times, 255 should be more likely to be the correct c_j^{min}.

In fact, our idea is equivalent to estimate c_j^{min} with *Maximum Likelihood Estimation* (MLE). MLE is a method of estimating the parameters of a probability distribution by maximizing a likelihood function, so that under the assumed statistical model the observed data is most probable. With the collected ciphertexts C_j, the adversary can enumerate all possible values (i.e., 0 to 255) for c_j^{min}. Suppose each enumeration θ, the corresponding probability mass function of the ciphertexts is as in Eq.(4):

$$Pr(c_j = v) = \begin{cases} 0 & v = \theta \\ 2 \times 2^{-8} & v = \theta \oplus f \\ 2^{-8} & \text{otherwise} \end{cases} \tag{4}$$

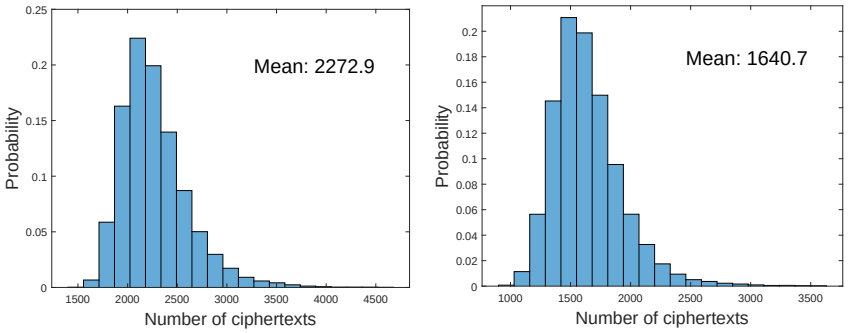
The probability of the event that N_c ciphertexts exactly follow the distribution C_j corresponding to specific θ can be calculated with the probability mass function in Eq.(5).

$$\begin{aligned} Pr(C_j | c_j^{min} = \theta \wedge c_j^{max} = \theta \oplus f) =& \frac{N_c!}{\prod_{v=0}^{255}(n_j^v!)} \cdot \delta_{n_j^\theta} \cdot (\frac{2}{256})^{n_j^{\theta \oplus f}} \cdot (\frac{1}{256})^{N - n_j^\theta - n_j^{\theta \oplus f}} \\ =& Const_0 \cdot \delta_{n_j^\theta} \cdot 2^{n_j^{\theta \oplus f}} \end{aligned} \tag{5}$$

In Eq.(5), n_j^θ is an abbreviation of $Counts(c_j = \theta)$ which is the number of ciphertexts whose $c_j = \theta$. $Const_0$ is a small positive constant. δ_n is the Kronecker delta function with the single argument n where $\delta_0 = 1$ and $\delta_n = 0$ for $n > 0$.

The mathematical proof can be found in Appendix A.

According to Eq.(5), if θ appears in C_j, $Pr(C_j | c_j^{min} = \theta \wedge c_j^{max} = \theta \oplus f)$ will be zero, which means it will be eliminated. For the rest of θs, the one with maximum $c_j^{max} = \theta \oplus f$



(a) Method in [ZLZ+18] **(b)** Method in this paper

Figure 4: Distribution of N_c for $\mathbf{c}^{min}$ with 10000 trials.

will have a maximum $Pr(C_j|c_j^{min} = \theta \wedge c_j^{max} = \theta \oplus f)$, thus it will be selected as the estimation for c_j^{min} and denoted as $\hat{c}_j^{min}$. Note that the explanation of Eq.(5) is consistent to the sketched description aforementioned. $\hat{c}_j^{min}$ can be represented as in Eq.(6).

$$\hat{c}_j^{min} = \arg\max_{\theta} Pr(C_j|c_j^{min} = \theta \wedge c_j^{max} = \theta \oplus f) \tag{6}$$

Such estimation on c_j^{min} $(1 \leq j \leq 16)$ can be applied to all ciphertext bytes in parallel to form all 16 bytes of $\mathbf{c}^{min}$. With $\mathbf{c}^{min}$, the key can be extracted.

To verify our new proposal and compare it with the original PFA, a simulation is repeated for 10000 times to find how many ciphertexts on average are required to recover $\mathbf{c}^{min}$ (all 16 bytes of c_j^{min}). 5000 ciphertexts are collected for each simulation. The statistic results of our proposed enhancement and that method in [ZLZ+18] are shown in Fig.4. On average, only 1641 ciphertexts are required in our method as compared with 2273 ciphertexts in [ZLZ+18]. In addition, the minimum N_c for recovering $\mathbf{c}^{min}$ in [ZLZ+18] is about 1493. In comparison, the minimum number of ciphertexts required in our MLE method is about 926, which is only about 62% of the previous method.

5.3 PFA with Unknown Fault Value *f*

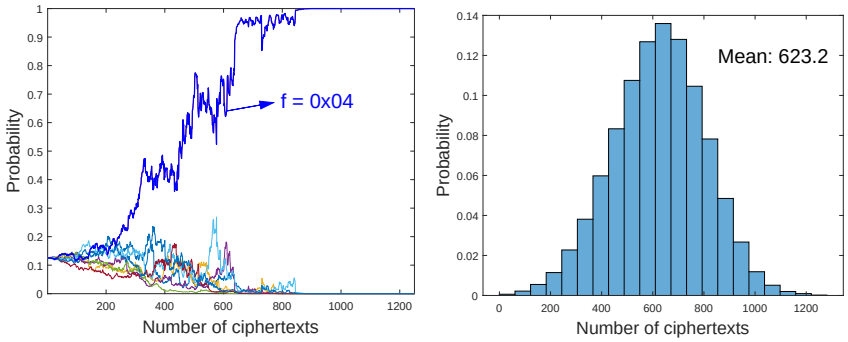
In this section, the constraint on the known fault value f can be further relaxed. That is to say, the fault analysis can be conducted with unknown fault values, which is the actual situation we met in practice.

According to Eq.(2), with sufficient ciphertexts, f can be easily recovered by XORing the most frequently appeared value with the one that never shows in C_j. That is $f = c_j^{min} \oplus c_j^{max}$. However, as shown in Fig. 3, in order to recover c_j^{max}, thousands of ciphertexts are normally required, which does not satisfy our attack goal of using least number of ciphertexts. Thus, a more efficient method needs to be proposed.

In fact, $f = c_j^{min} \oplus c_j^{max}$ can be applied to all 16 bytes rather than only one. Similar to Sec.5.2, when N_c is not enough, MLE is used to estimate the f from all 16 bytes of collected ciphertexts as shown in Eq.(7). The proof will be detailed in Appendix B.

$$\hat{f} = \arg\max_{\theta} Pr(C_0C_1...C_{15}|f = \theta) = \arg\max_{\theta} \prod_{j=0}^{15} (\sum_{l=0}^{255} \delta_{n_j^l} \cdot 2^{n_j^{l \oplus \theta}}) \tag{7}$$

The result is showed in Fig.5a which depicts how the value of likelihood function $Pr(C_0C_1...C_{15}|f = \theta)$ changes along with N_c. When N_c is increased to 686, the probability of one possible value of f will approximate to one. To further verify the number of



(a) $Pr(C_0C_1...C_{15})$ for different candidates of f v.s. the number of ciphertexts

(b) Distribution of N_c required for recovering f

Figure 5: Attack results for f.

ciphertexts that are required in our MLE method, 10000 simulations are conducted. The statistic is shown in Fig.5b. The average of N_c required for recovering f is 623.

In our MLE method, to relax the constraint of the fault value, f has to be determined before the analysis to be conducted in Sec.5.2. The deduction of f actually requires less number of ciphertexts that to recover $\mathbf{c}^{min}$, which is verified by our simulation. This is because all 16 bytes of distribution of ciphertexts can be used for recovering f while only one corresponding byte can be used for recovering each c_j^{min}. Thus, our attack complexity will not be affected by whether f is known or not. In short summary, in the situation that the fault value f is unknown, our MLE method can recover f first and proceed the subsequent fault analysis, which will still keep the number of required ciphertexts at a very small level.

5.4 PFA with Unknown Fault Location i

In this part, we leverage our attack to the complicated scenario that both f and i are unknown. As shown in Sec.5.2 and 5.3, both $\mathbf{c}^{min}$ and f can be recovered in advance, whose analyses are actually independent to the fault location i. However, since $k_j = c_j^{min} \oplus S[i]$, the recovery of the last round key has to rely on that of i.

Recall that the AES SBox table has 256 elements. A simple solution is to enumerate all 256 values of i for the possible locations of the injected fault. For each enumerated value of i, a possible candidate of $\mathbf{k}^{10}$ can be computed, which is the last round key of 16 bytes. Given a correct pair of plaintext and ciphertext, the key candidate can be easily verified. However, in practice, such pair may not be available to the adversary. Therefore an alternative solution needs to be proposed.

Our basic idea can be described as following: For each enumerated value of i, the corresponding $\mathbf{k}^{10}$ can be determined and used to decrypt each ciphertext. With N_c ciphertexts, the distribution of the output of SubBytes in the penultimate round can be computed. If the enumerated guess on i is correct, such outputs will have at least one missing value among all 256 ones, due to the nature of persistent fault which can cross multiple rounds. Otherwise, the guess on i will be incorrect if the output has 256 different values. Note that the calculation can be applied to all the 16 bytes in the 9^{th} round, which can further increase the accuracy of verification and reduce the number of ciphertexts that are required.

Based on this idea, a guess on i will be judged as correct if its corresponding output of SubBytes in the penultimate round has at least one missing value. Similar to previous discussions, the desired goal is to use as few numbers of ciphertexts as possible. When

N_c is small, the distribution of the output of SubBytes in the penultimate round may have missing values even if the guess i is wrong. This will cause some misjudgment on the candidate value of i. However, such misjudgment rate can be proved to be very small in practice. With N_c ciphertexts, as for a specific guess i, a total of $16N_c$ bytes of the output of SubBytes in the penultimate round can be calculated. When the guess on i is wrong, such outputs will approximately follow the uniform distribution due to the avalanche effect. The probability that such outputs have at least one missing value, i.e., the misjudgment rate P_{mj2}, can be calculated as:

$$P_{mj2} = \sum_{i=0}^{256} (-1)^i \cdot \binom{256}{i} \cdot (1 - \frac{i}{256})^{16N_c} \tag{8}$$

P_{mj2} decreases with N_c, the detailed relation between them can be refered to Fig. 13a in Appendix C. With $N_c = 300$ ciphertexts, P_{mj} is already reduced to 1.77×10^{-6}, which can be considered as negligible in practice. With more ciphertexts than 300, the possibility of such misjudgment can be totally ignored. Recall that in Sec.5.2, at least 926 ciphertexts are required to recover $\mathbf{c}^{min}$. Such number of ciphertexts is already enough for our method to find the fault location i with almost 100% probability.

5.5 Summary of Practical PFA on AES

Our proposal of practical and improved PFA on AES can be summarized as these steps: **Step 1** Collect 926 ciphertexts; **Step 2** Estimate the fault value f and $\mathbf{c}^{min}$ with Eq.(7) and Eq.(5); **Step 3** Enumerate i. For each guess on i, calculate the corresponding output of SubBytes in the penultimate round. If the outputs have missing values, the attack is succeeded. Else, collect an additional ciphertext and repeat from Step 2.

A physical experiment is carried out to test our analysis. The fault injection follows what we discussed in Sec. 4, and the 98^{th} injection is judged as an effective one. Then additional ciphertexts are collected continuously. With 1583 ciphertexts collected, a correct i is found, and the recovered master key is identical to the one used in μC.

6 Extended Application on PRESENT

The previously proposed attack is extented to PRESENT in this section.

6.1 PRESENT Block Cipher

PRESENT is an ultra-lightweight SPN block cipher proposed by Bogdanov et al. at CHES 2007 [BKL+07]. Its block size is 64 bits. It has two versions: PRESENT-80 and PRESENT-128. We only focus on PRESENT-80 in this paper whose master key is of 80 bits. It has 31 rounds and each round consists of three steps. (1) **sBoxLayer**. Each nibble (i.e. 4 bits) of the state is substituted through a 4-bit SBox lookup. (2) **pLayer**. The 64-bit state is bitwise permuted. (3) **addRoundKey**. A 64-bit round key is XORed with the current state. The key schedule process uses the 80-bit master key to generate 64-bit round key, which can be denoted as RK_{64}^j for the j^{th} round ($1 \leq j \leq 31$).

6.2 New Challenges for PRESENT

Theoretically, PFA is a generic attacking method and can be applied to PRESENT directly. However, in the practice of fault attacks, two new challenges need to be coped with.

Challenge 1: The SBox used in PRESENT is very small. It has only 16 elements and will be accessed 496 times in the whole encryption, i.e., each of 31 rounds will have

16 SBox lookups. Therefore, during the fault injection phase, the method in Sec.4.2 for identifying effective injections to AES is hard to be adopted to PRESENT. Suppose a fault is injected to SBox, the probability that a ciphertext remains correct (i.e., the faulty SBox element is not accessed in all rounds) can be computed as $P = (1 - \frac{1}{16})^{496} \approx 1.25 \times 10^{-14}$. That means nearly all ciphertexts will become faulty. Therefore, the previous method will no longer work, which only checks whether parts of the ciphertexts are faulty.

Challenge 2: Due to the nature of PFA, only the 64-bit round key that is XORed with the output of the SBox can be recovered. As the master key has 80 bits, additional work is required to recover the remained 16 key bits. Note that an exhaustive search can achieve such goal, however, it is not favored due to two reasons. One is that the exhaustive search needs a pair of correct plaintext/ciphertext, while our analysis can actually be conducted in a ciphertext-only scenario. The other is that the exhaustive search can not be applied to PRESENT-128, since there will be 64 remained key bits whose space is too large to search.

Challenge 1 and 2 will be overcomed in Sec.6.3 and 6.4, respectively.

6.3 Identification on Effective Injections

As mentioned in Challenge 1, it is difficult to determine whether a fault is injected to PRESENT SBox by checking the correctness of the ciphertexts (all ciphertexts are actually faulty). Therefore, a new method is proposed. Our idea is that: whether a persistent fault is really injected can be determined if at least one value of the output nibble of SBox is missing. Note that this property holds for all 16 nibbles simultaneously.

In this method, with limited ciphertexts, an effective injection that already placed a fault into SBox will not be misjudged as ineffective faults, since the corrupted SBox has only 15 possible values. However, it is possible for an ineffective fault to be misjudged as effective ones, if the output nibble of SBox has some missing values due to the lack of enough ciphertexts.

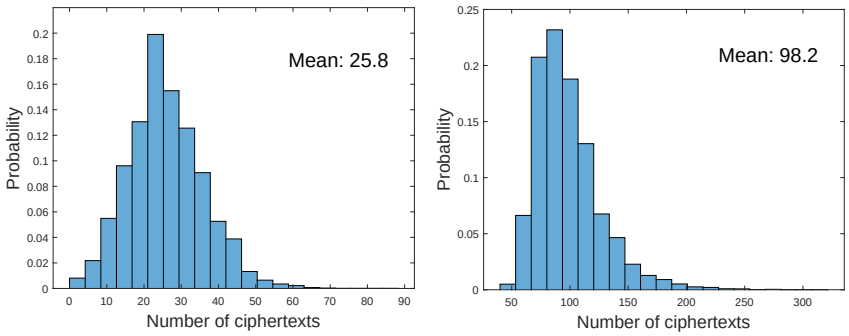
Assuming that N_c ciphertexts are collected and there is no fault injected to SBox of PRESENT, the probability that at least one value does not appear in one nibble of ciphertexts is: $P_N = \sum_{i=0}^{16} (-1)^i \times \binom{16}{i} \times (1 - \frac{i}{16})^{N_c}$, and the probability that all 16 nibbles of ciphertexts have at least one missing value for each nibble, i.e., the misjudgment rate P_{mj3}, can be computed as: $P_{mj3} = (\sum_{i=0}^{16} (-1)^i \times \binom{16}{i} \times (1 - \frac{i}{16})^{N_c})^{16}$.

When N_c increases, P_{mj3} converges to 0 very fast. The detailed relation between P_{mj3} and N_c can be referred to Fig.13b in Appendix C. With about 50 ciphertexts, the misjudgment rate can be reduced to 1.53×10^{-5}, which is already negligible in practice. Thus, after each injection, 50 ciphertexts are collected in order to identify the effective injections. More ciphertexts will be collected for further analysis if the injection is judged as an effective one.

6.4 Persistent Fault Analysis on PRESENT

Although the master key cannot be deduced merely from the last round key RK_{64}^{31}, it can be fully recovered if both RK_{64}^{30} and RK_{64}^{31} are determined. Our general idea for a full attack consists of two steps. The first step is to recover RK_{64}^{31}. The second step is to decrypt the last round with RK_{64}^{31} and get the outputs of the penultimate round. Then an attack similar to the one on the last round can be carried out to recover RK_{64}^{30}. With both RK_{64}^{31} and RK_{64}^{30}, the master key can be recovered without the exhaustive search.

In Step 1, all of the fault location i, the fault value f and the round key RK_{64}^{31} need to be recovered first. Similar to Sec.5, we start our analysis with a known fault location i.



(a) Recover f **(b)** Recover $\mathbf{c}^{min}$

Figure 6: Distribution of N_c for PRESENT, 10000 simulations.

6.4.1 Attack on the last round with known i

Similar to the attack on AES, the fault value f and $\mathbf{c}_j^{min}$ can be recovered with MLE:

$$\hat{f} = \arg\max_{\theta} Pr(C_0C_1...C_{15}|f=\theta) = \arg\max_{\theta} \prod_{j=0}^{15}(\sum_{l=0}^{15} \delta_{n_j^l} \cdot 2^{n_j^{l\oplus\theta}}), \quad 1 \le \theta \le 15 \tag{9}$$

$$\hat{c}_j^{min} = \arg\max_{\theta} Pr(C_j|c_j^{min} = \theta \wedge c_j^{max} = \theta \oplus f) = \arg\max_{\theta} \delta_{n_j^{\theta}} \cdot 2^{n_j^{\theta\oplus f}}, 0 \le \theta \le 15 \tag{10}$$

10000 simulations are carried out to test how many ciphertexts are required for recovering f and $\mathbf{c}^{min}$. The results in Fig.6 show that they can be recovered with 26 and 98 ciphertexts on average, respectively. Compared with PFA on AES, the number of required ciphertexts is dramatically reduced. The reason is that: PRESENT performs on nibbles rather than bytes, thus $\hat{c}_j^{min}$ only has 16 candidates and can be recovered much faster. In addition, f can still be recovered before $\mathbf{c}^{min}$, as its analysis can be conducted among all 16 nibbles to jointly retrieve the result. With a specified fault index i, the 64-bit last round key RK_{64}^{31} can be recovered by XORing $\mathbf{c}^{min}$ with $S[i]$.

6.4.2 Attack on the last round with unknown i

Now we leverage the attack to the real scenario when the fault location i is unknown. The general idea is similar as that in Sec.5.4. The fault index i is enumerated. For each guess on i, the ciphertexts will be traced back to calculate the sBoxLayer outputs in the penultimate (30^{th}) round, and to check if there is any missing value.

Actually, the reverse calculation is not that straightforward for PRESENT. Let SI_j^n and SO_j^n denote the j^{th} nibble of the SBox input and output in the n^{th} round, respectively. Then $\mathbf{SI}^n$ and $\mathbf{SO}^n$ denote all 16 nibbles of SI_j^n and SO_j^n ($1 \le j \le 16$). For a specific guessed i, the last round key can be determined, and $\mathbf{SO}^{31}$ can be calculated. As shown in Fig.7, in order to get $\mathbf{SO}^{30}$ from $\mathbf{SO}^{31}$, all three operations (sBoxLayer, addRoundKey and pLayer) need to be reverse calculated. During these reverse calculation, two specific problems must be coped with.

Problem 1: With a fault inside the SBox, the sBoxLayer becomes irreversible. Thus, part of $\mathbf{SI}^{31}$ cannot be deduced from $\mathbf{SO}^{31}$. For example, suppose $S[0] = 0xc$ is changed into $S'[0] = 0xd$ which is equal to $S[7]$. When the output of SBox is $0xd$, its input has two possibilities 0 and 7. It is hard to decide which one is correct, and we have to skip using such nibbles for fault analysis. Therefore, the ratio of SI_j^{31} that can be used for analysis is $\frac{16-2}{16} = 87.5\%$. Furthermore, as shown in Fig.7, each SO_j^{30} is associated with

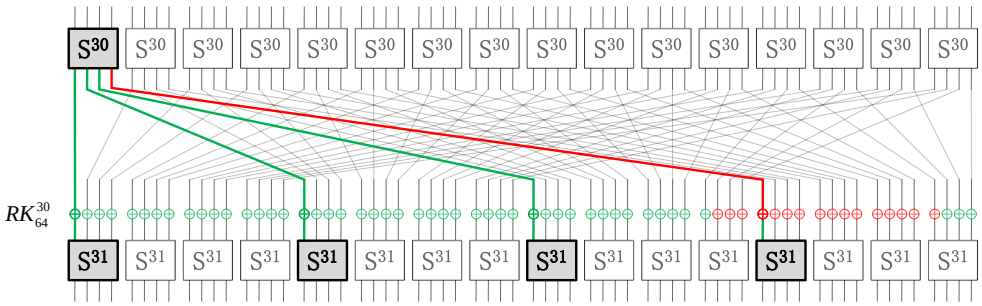


Figure 7: Part of the last two rounds of PRESENT. The relation between SO_{15}^{30} and $\mathbf{SI}^{31}$ are highlighted, and the determined/undetermined bits are colored green/red respectively.

four nibbles of $\mathbf{SI}^{31}$. So the probability that SO_j^{30} can be deduced is $(87.5\%)^4 \approx 58.62\%$, i.e., only about half of the SO_j^{30} can be deduced.

Problem 2: Even if RK_{64}^{31} is fully determined, only 48-bit of RK_{64}^{30} can be deduced from RK_{64}^{31}, while the other 16 bits remain unknown. This is due to the design of key scheduling for PRESENT. In Fig.7, the known and unknown key bits are colored green and red, respectively. Note that each SO_j^{30} is exactly associated with 3 known key-bits and 1 unknown key-bit. Hence, that bit in the output nibble connected to the unknown-key bit cannot be directly used for analysis.

Recall that the judgment on i relies on whether the impossible value appears in SO_j^{30} or not. Such impossible value may have two candidates due to the unknown key-bit. However, the appearance of this impossible value can be confirmed if both of the candidates appear. If the impossible value appears, the guess on i must be wrong.

The misjudgment could still happen when the guessed i is incorrect and not all of the two candidates of impossible values appear. When the guessed i is wrong, the calculated SO_j^{30} will be wrong, and it will basically follow uniform distribution. Due to Problem 1, about $0.58N_c$ nibbles can be collected for SO_j^{30}. The misjudgment rate P_{mj4}, i.e., the probability that at least one of candidates is missed for all SO_j^{30} $(1 \leq j \leq 16)$, is:

$$P_{mj4} = \left(2 \times \left(\frac{15}{16}\right)^{0.58N_c} - \left(\frac{14}{16}\right)^{0.58N_c}\right)^{16} \tag{11}$$

Recall Sec.6.4.1, at least 44 ciphertexts are required to recover $\mathbf{c}^{min}$. With such number of ciphertexts, the misjudgment rate P_{mj4} is lower than 3.76×10^{-7}, which means our method can extract the value of i very accurately.

Thus, the last round key RK_{64}^{31} can be recovered with a similar process in Sec.5.5, where 44 ciphertexts will be collected at first.

6.4.3 Attack on the penultimate round

The attack on the penultimate round is quite similar to that on the last round. However, there are two major differences. One is that only 58.62% nibbles are remained as discussed. The other is that f, i and 48 bits of RK_{64}^{30} are already recovered in advance when attacking the last round.

Eq.(10) is used again to estimate $\mathbf{c}^{min}$ for the penultimate round. Note that each c_j^{min} has only two candidates now. The reason is that $S[i]$ and 3 bits of the key nibble are known, and c_j^{min} equals the XOR between $S[i]$ and the j^{th} nibble of RK_{64}^{30}. 10000 simulations show that 101 ciphertexts are required to recover $\mathbf{c}^{min}$ on average, which is very close to the 98 required ciphertexts to recover $\mathbf{c}^{min}$ for the last round. The reason is

that each c_j^{min} has only two candidates in the penultimate round, although only about half of the ciphertexts can be used.

7 Experimental Results

In this section, we experimentally verify our proposed method and understand how the address of software program (especially the SBox tables) corresponds to the physical location of the microcontroller with carefully designed scans. We investigate: (1) Mapping the logical address of data to the physical location in SRAM and finding out the exact region of both AES and PRESENT SBox. (2) Identifying effective injections for AES and PRESENT. (3) Finding distribution of the sensitive region and understanding the characteristics and mechanisms of faults.

To investigate previous attacks with more insights, full access is considered, allowing that the data in the entire SRAM can be directly exported before and after the fault injection. For simplification, both AES and PRESENT are implemented in the same microcontroller. An automated scan with certain granularity of steps in x-y direction is conducted.

7.1 Automated Scan

A software program is written to control the laser spot scanning automatically. The scan strategy is set as shown in Fig.8. At a very beginning, a fixed and visible point needs to be decided as the reference point to make sure the experiment is accurate and reproducible. The point is chosen at the lower right corner of the chip, and the motorized stage is programmed to automatically move towards the reference point first, and then prepare the automated scan.

For each scan, three parameters need to be defined: (1) the region to be scanned, (2) the distance of steps d_x and d_y, in x and y direction respectively. (3) the time interval t between two consecutive laser injections in one scan. The detailed path for the scan is shown in the enlarged part in Fig.8. During the scan, the predefined information, such as ciphertexts and SRAM contents, will be collected from the μC and saved with its corresponding coordinate after each laser injection, then the μC is reset to prepare for the next injection.

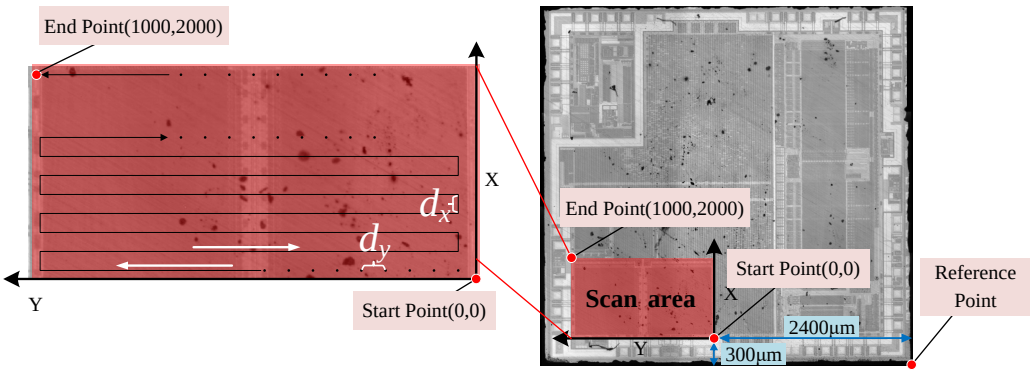


Figure 8: The laser scan trajectory to cover full area of the SRAM. The lower right corner of the chip is chosen as the reference point. A total area of $1000\mu m \times 2000\mu m$ will be scanned to cover the SRAM.

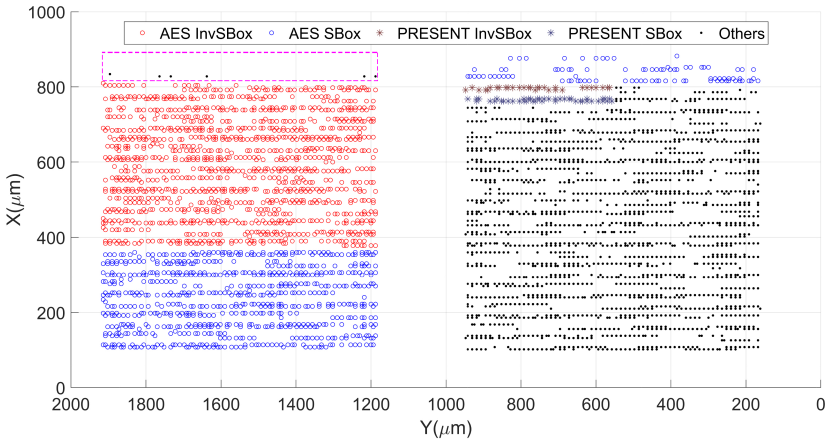


Figure 9: Scanning results of SRAM.

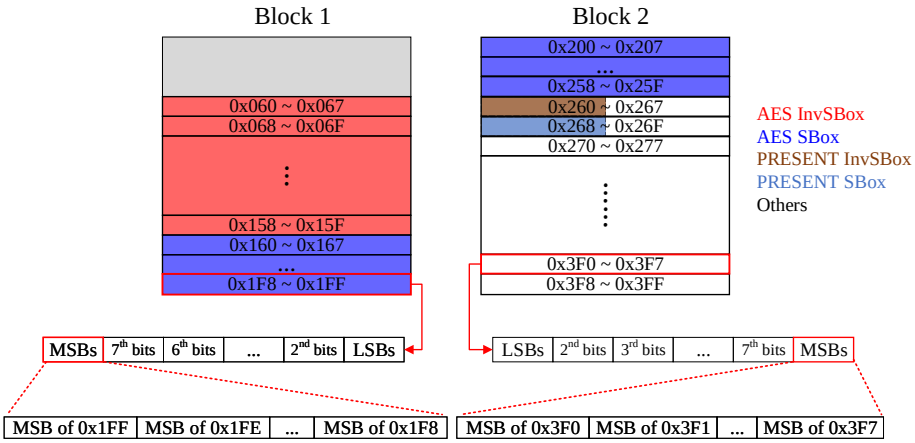


Figure 10: Address mapping of SRAM. Addresses of SBoxes and InvSBoxes are colored. One row of each SRAM block is zoomed to detail bit arrangement.

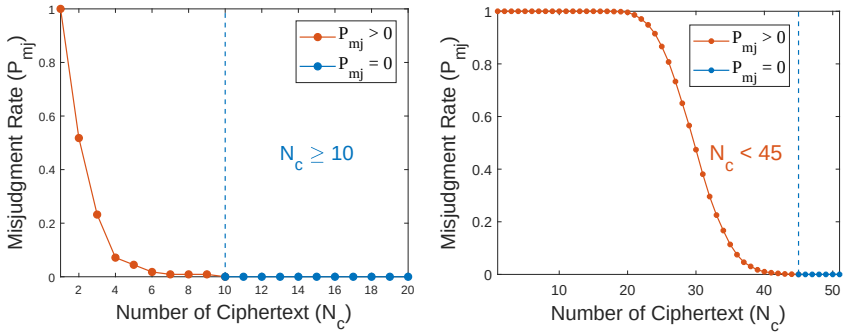
7.2 Coarse Scan for Mapping Address

The automated scan can help mapping the logical address of data to the physical location in SRAM as well as finding the exact region of the AES SBox.

For each laser pulse during the scan, the SRAM data will be dumped from μC and compared with the original SRAM data which is read before the first laser pulse. From the change of SRAM data and the coordinate of the corresponding laser pulse, the relationship between logical address and physical location of SRAM data can be established. With the map between logical address and physical location, the exact region of the SBox can be found since the logical address field of SBox can be easily analyzed from the readback SRAM data.

As mentioned before, three parameters must be determined: the region of the scan, the time interval t, and the size of step d. The region of the scan is set to cover the entire SRAM. Since the computer reads all of the SRAM data from μC after each laser pulse, the time interval t is set to 250ms to ensure that the communication can be finished. The size of step d is set to $6\mu m$ to balance the precision granularity and the total time duration. Thus a total of $\lceil \frac{2000}{6} \rceil \times \lceil \frac{1000}{6} \rceil = 55778$ laser pulses are performed.

A total of 4044 bit flips are found during our scan and they are shown in Fig.9. The SRAM consists of two square areas, denoted as Block 1 and 2 respectively, which



(a) AES **(b)** PRESENT

Figure 11: The misjudgment rate P_{mj} v.s. the number of ciphertexts. No misjudgment will be made with 10 and 45 ciphertexts for AES and PRESENT, respectively.

are consistent with the layout shown in Fig.1. The actual size of each square is about $800\mu m \times 800\mu m$ and each bit occupies an area of about $12.5\mu m \times 12.5\mu m$. From the readback data, it is learned that SBox and InvSBox of AES occupy the address field 0x160∼0x25F and 0x060∼0x15F, respectively. Meanwhile, the PRESENT SBox and InvSBox occupy 0x160∼0x25F and 0x060∼0x15F, respectively. Note that the PRESENT elements have only 4 bits and each of them is stored in the lower 4 bits of a byte.

The mapping between logical address and physical location is detailed in Fig.10. Block 1 and 2 hold the address range of 0x060∼0x1FF and 0x200∼0x3FF, respectively. Each row in the block contains 8 bytes, but it is interesting that these 8 bytes are arranged by bits rather than bytes. For example, eight MSB bits from different bytes are grouped into one row, instead of putting eight bits (from MSB to LSB) together.

The region marked with a pink box in Fig.9 is quite different, which is marked as grey in Fig.10. This area should contain 96 bytes of data according to its size. However, it can not be readback after the injection, and few faults are with the address range of 0x050∼0x05F. According to the datasheet of ATmega163L [Mic03], the data memory address is organized as follows: 0x000∼0x01F are for register files, 0x020∼0x05F for IO registers, and 0x060∼0x45F for 1KB SRAM. Based on the injection results, the marked area should be associated with the address range 0x000∼0x05F. [2]

7.3 Verification of Methods for Identifying Effective Injections

In Sec.4.2 and 6.3, two different methods are proposed to identify effective injection, i.e., to test whether the injection corrupted the SBox or not. Both of the theoretical misjudgment rates are calculated and proved to be negligible with small number of ciphertexts. In this part, these two methods will be verified with physical experiments.

According to the analysis in Sec.4.2 and 6.3, after each injection, 20 and 50 ciphertexts will be collected for AES and PRESENT respectively for our methods to check the effectiveness of the injection. In addition, the SRAM will be dumped after each injection to provide the ground truth by directly comparing the SBox before and after the injection.

Again, a scan is carried out for the verification and three parameters need to be determined. The region of scan is set to cover the entire $2000 \times 1000\mu m$ SRAM to make sure both SBox and non-SBox area are covered. The time interval t is set to 2500ms to ensure that the collection on the ciphertexts and SRAM can be finished. The step size d_x and d_y are both set to $20\mu m$ to make sure that faults can be injected into the SBoxes.

[2] It is not clear where those addresses 0x400∼0x45F are physically located and why they are missing. However, this is not important as we only inject faults to SBoxes whose address never reaches 0x3FF.

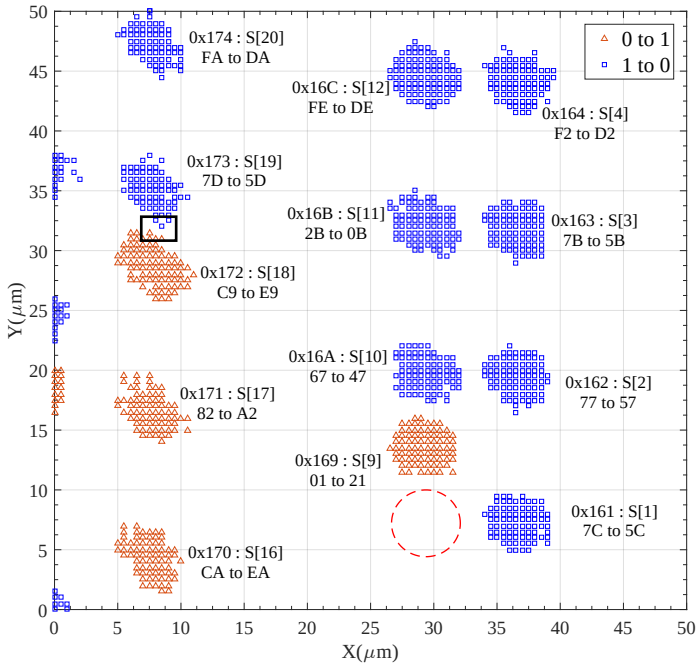


Figure 12: Distribution of the sensitive region in $50\mu m \times 50\mu m$ area.

Therefore, a total of $(\frac{2000}{20}+1)\times(\frac{1000}{20}+1)=5151$ laser pulses are performed, taking about four hours.

After the scan, the ground truth shows that a total of 112 and 4 faults were injected into the SBox of AES and PRESENT, respectively. Meanwhile, our identification methods also find the same amount of faults at the identical location in SRAM, which confirms that our methods work quite well. Then, how our identification methods correspond to the number of required ciphertexts is investigated. The results are shown in Fig.11. The orange and blue curves show the numbers of ciphertexts that with and without misjudgment, respectively. It can be found that no misjudgment will be made by our methods with more than 11 ciphertexts for AES and 49 ciphertexts for PRESENT.

7.4 Refined Scan for Finding Sensitive Region

Each bit occupies an area of about $12.5\mu m \times 12.5\mu m$. Since the facility can be as precise as 0.1 μm per step, it is quite interesting to know which specific part of that area is sensitive to bit flips. In order to find the distribution of sensitive regions, a refined scan is carried out additionally.

A random area of $50\mu m \times 50\mu m$ is selected for the refined scan, which should contain about 4×4 SRAM cells. It is large enough for us to find out the distribution of sensitive regions. $dx = dy$ are set to be $0.5\mu m$, which is precise enough for a $12.5\mu m \times 12.5\mu m$ cell. t is still set to be 250ms, which is the same as in the coarse scan.

The scan result is shown in Fig.12. This region contains 17 SRAM cells, each of which is the 6^{th} bit of some SBox bytes. Those $1 \to 0$ and $0 \to 1$ bit flips are represented by triangles and squares, respectively. These areas actually reflect the location of the NMOS or PMOS transistors in the SRAM structure. According to the direct observation in Fig.12, the size of a sensitive region can be measured by about $5\mu m \times 5\mu m$.

In Fig.12, the sensitive regions on the right need further investigation. 7 of 8 sensitive regions of $S[1] \sim S[4]$ and $S[10] \sim S[12]$ are well organized (the pairwise distance is about $5\mu m$ vertically), while that of $S[9]$ is very close to $S[10]$. The distance between them is

only about $1\mu m$ vertically. This is because the sensitive regions for $1 \rightarrow 0$ and $0 \rightarrow 1$ flips are different, which are determined by the structural characteristics of the SRAM cell. The current sensitive region for $S[9]$ is a $0 \rightarrow 1$ flip, while the rest seven regions are $1 \rightarrow 0$ flips. If the 6^{th} bit of $S[9]$ is 1, its sensitive region should be in the red circle.

In addition, due to the adjacent sensitive regions shown in Fig.12, it is possible to flip two bits simultaneously with only one laser pulse. One of the possible region to position the laser beam is shown as the black box in Fig.12, which will cause both $S[18]$ and $S[19]$ to change at the same time.

7.5 PFA with Double Faults

It is observed from our practice that $S[18]$ is changed from 0x$C9$ to 0x$E9$ and $S[19]$ is from 0x7D to 0x5D. A simple analysis is carried out to deal with the double fault. Now c_j^{min} has two values since two of the SBox elements are corrupted, and they can be denoted as c_{j0}^{min} and c_{j1}^{min}. Both of them can be recovered with MLE.

To get the statistic result of N_c, 10000 simulations are conducted for recovering all 16 bytes of c_{j0}^{min} and c_{j1}^{min}. 5000 ciphertexts are collected for each simulation. 2146 ciphertexts are required on average. In addition, as there are two impossible values, each key byte will have two candidates. An exhaustive search is required to test which one is correct.

8 Conclusion and Discussion

In this paper, we put the so-called persistent fault attack into practice with a physical experiment under traditional fault injection scenarios. An AES-Sbox implementation on ATmega163L is targeted and successfully attacked. First, the fault is injected into SBox stored in SRAM of the microcontroller with random laser pulses. Then, a persistent fault analysis is conducted with the single-byte fault injected by laser pulses, showing that PFA still works while the fault value and location are unknown. The statistic result shows that 1641 ciphertexts are enough for recovering the master key of AES, which is 28% lower than 2273 ciphertexts in the previous works. Moreover, the proposed attack is extended to the PRESENT cipher, which has a different structure that is difficult to attack. Finally, the entire SRAM is fully investigated with revisits in order to help us to understand the reasons and mechanisms that are hidden behind.

Several types of countermeasures can be used to defend against the proposed attack. One possible direction of countermeasure design is at the circuit layer, where a few ring oscillators can be deployed on the top of sensitive regions. Those ROs can send out the alarm signal once the laser injection is detected. The other possible countermeasure design is relatively simple. The device can store one pair of plaintext P and ciphertext C which are encrypted with the secret key. During the encryption, the device can perform an online health check by comparing the runtime ciphertext with the precomputed C, which might be difficult for PFA to bypass.

Acknowledgments

This work was supported in part by Open Fund of State Key Laboratory of Cryptology (Grant No. MMKFKT201805), by Alibaba-Zhejiang University Joint Institute of Frontier Technologies, by Zhejiang Key R&D Plan (2019C03133), by Major Scientific Research Project of Zhejiang Lab (2018FD0ZX01), by Young Elite Scientists Sponsorship Program by CAST (17-JCJQ-QT-045), by National Natural Science Foundation of China (61772236, 61802180), by Natural Science Foundation of Jiangsu Province (BK20180421), by National Cryptography Development Fund (MMJJ20180105), by Fundamental Research Funds for

the Central Universities (NE2018106), by Leading Innovative and Entrepreneur Team Introduction Program of Zhejiang, and by Research Institute of Cyberspace Governance in Zhejiang University. The corresponding author is Zhe Liu (Email: zhe.liu@nuaa.edu.cn).

References

[BKL+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. *Lecture Notes in Computer Science*, 4727:450–466, 2007.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. *Lncs*, 1294:513–525, 1997.

[BSH75] D. Binder, E. C. Smith, and A. B. Holman. Satellite anomalies from galactic cosmic rays. *Nuclear Science IEEE Transactions on*, 22(6):2675–2680, 1975.

[CLMFT14] Franck Courbon, Philippe Loubet-Moundi, Jacques JA Fournier, and Assia Tria. Adjusting laser injections for fully controlled faults. In *International workshop on constructive side-channel analysis and secure design*, pages 229–242. Springer, 2014.

[DDL97] Boneh Dan, Richard A. Demillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *International Conference on Theory and Application of Cryptographic Techniques*, pages 37–51, 1997.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked aes with fault countermeasures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 315–342. Springer, 2018.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 547–572, 2018.

[DLV03] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on aes. In *International Conference on Applied Cryptography and Network Security*, pages 293–306. Springer, 2003.

[FJLT13] Thomas Fuhr, Eliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on aes with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 108–118. IEEE, 2013.

[Hab65] 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.

[KQ08] Chong Hee Kim and Jean-Jacques Quisquater. New differential fault analysis on aes key schedule: Two faults are enough. In *International Conference on Smart Card Research and Advanced Applications*, pages 48–60. Springer, 2008.

[MBD+19] Alexandre Menu, Shivam Bhasin, Jean-Max Dutertre, Jean-Baptiste Rigaud, and Jean-Luc Danger. Precise spatio-temporal electromagnetic fault injections on data transfers. In *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 1–8. IEEE, 2019.

[Mic03] Microchip. ATmega163(L). https://www.riscure.com/security-tools/inspector-fi/, 2003.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against spn structures, with application to the aes and khazad. In *International workshop on cryptographic hardware and embedded systems*, pages 77–88. Springer, 2003.

[PZRB19] Jingyu Pan, Fan Zhang, Kui Ren, and Shivam Bhasin. One fault is all it needs: Breaking higher-order masking with persistent fault analysis. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2019.

[SA02] 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.

[TMA11] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. In *IFIP international workshop on information security theory and practices*, pages 224–233. Springer, 2011.

[WM62] J. T. Wallmark and S. M. Marcus. Minimum size and maximum packing density of nonredundant semiconductor devices. *Proceedings of the Ire*, 50(3):286–298, 1962.

[WW10] Gaoli Wang and Shaohui Wang. Differential fault analysis on present key schedule. In *2010 International Conference on Computational Intelligence and Security*, pages 362–366. IEEE, 2010.

[ZLZ+18] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. Persistent fault analysis on block ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 150–172, 2018.

A Proof for Eq.(5)

$$\begin{aligned} & Pr(C_j|c_j^{min}=\theta \wedge c_j^{max}=\theta \oplus f) \\ =&\frac{N_c!}{\prod_{v=0}^{255}(n_j^v!)} \cdot \delta_{n_j^\theta} \cdot (\frac{2}{256})^{n_j^{\theta\oplus f}} \cdot (\frac{1}{256})^{N-n_j^\theta-n_j^{\theta\oplus f}} \\ =&Const_0 \cdot \delta_{n_j^\theta} \cdot 2^{n_j^{\theta\oplus f}} \end{aligned} \tag{12}$$

As the probability mass function of ciphertext byte is specified in Eq.(4), the probability of the event that N_c ciphertexts exactly follow the distribution C_j, i.e., $Pr(C_j|c_j^{min}=l \wedge c_j^{max}=l\oplus f)$ follows multinomial distribution:

$$\begin{aligned} & Pr(C_j|c_j^{min}=\theta \wedge c_j^{max}=\theta \oplus f) \\ =&\frac{N_c!}{\prod_{v=0}^{255}(n_j^v!)} \cdot \left(\frac{1}{256}\right)^{N-n_j^\theta-n_j^{\theta\oplus f}} \cdot 0^{n_j^\theta} \cdot \left(\frac{2}{256}\right)^{n_j^{\theta\oplus f}} \\ =&\frac{N_c!}{256^N \cdot \prod_{v=0}^{255}(n_j^v!)} \cdot 0^{n_j^\theta} \cdot 2^{n_j^{\theta\oplus f}} \\ =&Const_0 \cdot \delta_{n_j^\theta} \cdot 2^{n_j^{\theta\oplus f}}, \end{aligned} \tag{13}$$

where $Const_0 = \frac{N_c!}{256^N \cdot \prod_{v=0}^{255}(n_j^v!)}$, and $0^{n_j^\theta}$ is actually equivalent $\delta_{n_j^\theta}$.

B Proof for Eq.(7)

$$\hat{f} = \arg\max_\theta Pr(C_0C_1...C_{15}|f=\theta) = \arg\max_\theta \prod_{j=0}^{15}\left(\sum_{l=0}^{255}\delta_{n_j^l} \cdot 2^{n_j^{l\oplus\theta}}\right) \tag{14}$$

Since each byte of the ciphertext is independent of the others, and $f = S[i] \oplus S'[i]$, we have:

$$Pr(C_0C_1...C_{15}|f=\theta) = \prod_{j=0}^{15} Pr(C_j|S[i]\oplus S'[i]=\theta) \tag{15}$$

The probability of the distribution of j^{th} byte of ciphertext can be calculated by formula of total probability:

$$\begin{aligned} & Pr(C_j|S[i]\oplus S'[i]=\theta) \\ =&\sum_{l=0}^{255} Pr(c_j^{min}=l|S[i]\oplus S'[i]=\theta) \cdot Pr(C_j|c_j^{min}=l \wedge S[i]\oplus S'[i]=\theta) \\ =&\sum_{l=0}^{255} \frac{1}{256} \cdot Pr(C_j|c_j^{min}=l \wedge c_j^{max}=l\oplus\theta) \end{aligned} \tag{16}$$

Recalling Eq.(13) we have

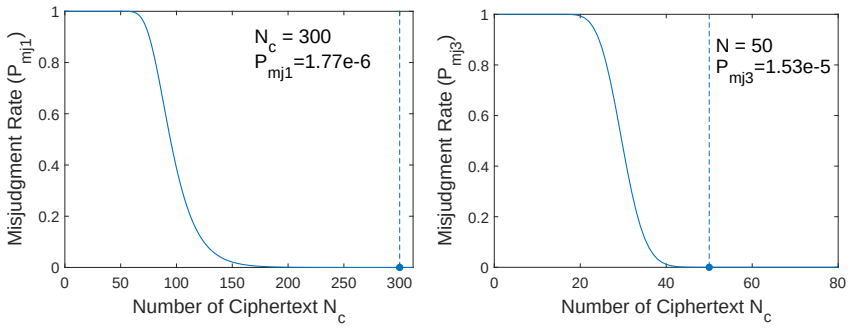
$$Pr(C_j|c_j^{min} = l \wedge c_j^{max} = l \oplus \theta) = Const_0 \cdot \delta_{n_j^l} \cdot 2^{n_j^{l \oplus \theta}} \tag{17}$$

With Eq.(14)∼(17), the estimate of fault value $\hat{f}$ is

$$\begin{aligned}
\hat{f} &= \arg\max_{\theta} Pr(C_0 C_1 ... C_{15} | f = \theta) \\
&= \arg\max_{\theta} \prod_{j=0}^{15} (\sum_{l=0}^{255} Const_0 \cdot \delta_{n_j^l} \cdot 2^{n_j^{l \oplus \theta}}) \\
&= \arg\max_{\theta} Const_0^{16} \prod_{j=0}^{15} (\sum_{l=0}^{255} \cdot \delta_{n_j^l} \cdot 2^{n_j^{l \oplus \theta}}) \\
&= \arg\max_{\theta} \prod_{j=0}^{15} (\sum_{l=0}^{255} \delta_{n_j^l} \cdot 2^{n_j^{l \oplus \theta}}), \theta = 1, 2, ..., 255
\end{aligned} \tag{18}$$

where $Const_0^{16}$ is ignored since it is a positive constant and will not affect the result.

C Figure for Misjudgment Rates.



(a) P_{mj1} for AES **(b)** P_{mj3} for PRESENT

Figure 13: The relation between the misjudgment rates and N_c. Both of them will be negligible with proper number of ciphertexts.