IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–24. DOI:XXXXXXXX

Practical Multiple Persistent Faults Analysis

Hadi Soleimany[1], Nasour Bagheri[2,3], Hosein Hadipour[4], Prasanna Ravi[5], Shivam Bhasin[5] and Sara Mansouri[1]

[1] Cyber Research Center, Shahid Beheshti University, Tehran, Iran
h_soleimany@sbu.ac.ir,sar.mansouri@gmail.com
[2] Shahid Rajaee Teacher Training University, Tehran, Iran
[3] School of Computer Science (SCS), Institute for Research in Fundamental Sciences (IPM),Tehran, Iran
nbagheri@sru.ac.ir
[4] Graz University of Technology, Graz, Austria
hossein.hadipour@iaik.tugraz.at
[5] Temasek Laboratories, NTU, Singapore
prasanna.ravi@ntu.edu.sg,sbhasin@ntu.edu.sg

Abstract.
We focus on the multiple persistent faults analysis in this paper to fill existing gaps in its application in a variety of scenarios. Our major contributions are twofold. First, we propose a novel technique to apply persistent fault in the multiple persistent faults setting that decreases the number of survived keys and the required data. We demonstrate that by utilizing 1509 and 1448 ciphertexts, the number of survived keys after performing persistent fault analysis on AES in the presence of eight and sixteen faults can be reduced to only 2^9 candidates, whereas the best known attacks need 2008 and 1643 ciphertexts, respectively, with a time complexity of 2^{50}. Second, we develop generalized frameworks for retrieving the key in the ciphertext-only model. Our methods for both performing persistent fault attacks and key-recovery processes are highly flexible and provide a general trade-off between the number of required ciphertexts and the time complexity. To break AES with 16 persistent faults in the Sbox, our experiments show that the number of required ciphertexts can be decreased to 477 while the attack is still practical with respect to the time complexity. To confirm the accuracy of our methods, we performed several simulations as well as experimental validations on the ARM Cortex-M4 microcontroller with electromagnetic fault injection on AES and LED, which are two well-known block ciphers to validate the types of faults and the distribution of the number of faults in practice.

Keywords: Fault Attack · Persistent Fault Analysis · Multiple Faults · AES

1 Introduction

Fault attacks are a class of physical attacks that consists of two phases; (1) fault injection and (2) fault analysis. In the first phase, the adversary tries to disturb the operation of the target device by using the available tools for injecting the desired fault. Fault can be induced by some common methods, such as clock glitches, voltage starving, voltage spikes, electromagnetic pulses, laser pulses, light pulses, hardware Trojans, and so on. In the second phase, the adversary analyzes the response of the target device to the fault with the aim of retrieving some sensitive information like the secret key.

Boneh et al. were the first to introduce fault attacks with their application on RSA [BDL97]. Shortly after this seminal work, Biham and Shamir proposed the Differential Fault Analysis (DFA) on block cipher DES [BS97]. DFA is the most common fault analysis technique that has been successfully applied to various block ciphers. Besides, novel techniques have been proposed in follow-up works such as Fault Sensitivity Analysis (FSA) [LSG+10], Differential Fault Intensity Analysis (DFIA) [GYTS14], Safe-Error Analysis (SEA) [YJ00], Ineffective Fault Analysis (IFA) [Cla07], Statistical Fault Analysis (SFA) [FJLT13], and so on. Dobraunig et al. exploited Statistical Fault Analysis in an ineffective fault setting to introduce Statistical Ineffective Fault Analysis (SIFA) [DEK+18].

The generated faults can be generally classified into three categories based on the fault duration. Most proposed fault attacks rely on transient faults which have a temporary effect on the system. The permanent fault is another type of fault that has irreversible effects during the lifetime. The third type of fault is the persistent fault that drops between transient and permanent faults. The faulty value persists from one encryption to another one, while it disappears when the target device is reset.

The notion of persistent fault was first coined by Schmidt et al. who presented an attack on AES by erasing the non-volatile memory with ultraviolet light [SHP09]. This notion was later expounded by Zhang et al. in an extended way to a more dedicated framework [ZLZ+18]. Persistent fault attack or analysis (PFA) has several advantages over the previous fault attacks. The main advantageous features are as follows: 1) The attack only requires a set of ciphertexts, which is a less restrictive model than known- or chosen-plaintext models. 2) The fault injection does not require precise timing or location based on the synchronization of the encryption process. 3) PFA takes advantage of the inherent characteristic to bypass some redundancy-based countermeasures.

1.1 Related Works

The original PFA proposed in [ZLZ+18] has a number of limitations. The major limitation of original attacks concerns the assumption that the fault location and the value of the fault are known to the attacker. In a follow-up work at CHES 2020 [ZZJ+20], it is shown that the assumption of having exact knowledge of the fault value can be relaxed. Besides, this work introduced a new framework based on utilizing maximum likelihood estimation (MLE) that led to a 28% reduction in the required ciphertexts compared to the original attack [ZLZ+18]. However, this framework is only applicable if a single fault occurs, and it cannot be extended to the multiple-faults setting unless the attacker knows the exact location and value of the fault again. This issue was addressed in [ESP20]. Engels et al. presented a new attack called Statistical Persistent Fault Analysis (SPFA), by making use of the statistical fault analysis (SFA) [FJLT13] in the persistent fault settings. While SPFA makes it possible to perform persistent fault analysis in the multiple-faults setting without requiring the location and value of fault, the average residual key entropy is still high. For instance, the correct key of AES can only be found through an exhaustive search with a complexity of 2^{50}. Hence, the overall complexity of SPFA is dominated by the process of finding the correct key among possible candidates after performing the attack.

Aside from the the aforementioned works on circumventing PFA constraints, several publications focused on other aspects, mainly demonstrating the application of the original attack to various ciphers and implementations. Pan et al. indicated that PFA can break any higher-order masking countermeasures with only one persistent fault injection in [PZRB19]. Caforio and Banik presented studies on the application of PFA for reverse-engineering purposes in the chosen-key model [CB19]. Gruber et al. applied the PFA to authenticated encryption schemes such as Deoxys-II, OCB, and COLM [GPT19]. In [CGR20], the authors meliorated the PFA by using several steps such as estimation theory, rank, and key combination algorithms. Very recently, Xu et al. proposed Enhanced PFA (EPFA) [XZY+21] which is more a key recovery approach in the single fault model. In their work, they employ the leaked information in other rounds also to reduce the number of required ciphertexts. However, an extension of this method to the multiple faults setting is challenging, as we will discuss later. Besides, the improvement in this method comes with the cost of a less feasible assumption in which the attacker knows the location of the faults.

1.2 Our Contributions

Persistent fault attacks that are published so far can be classified in different ways: 1) Whether the value of fault is known or unknown. 2) Whether the attack can be efficiently extended in the multiple-faults setting. 3) Whether the average residual key entropy (and consequently the overall complexity) is low or high. 4) Whether finding the correct key among the remaining candidates (after performing the PFA) is considered in the ciphertext-only scenario or the known-plaintext scenario.

By considering the proposed attacks based on the aforementioned classifications, a number of important problems have remained open that can be identified as follows:

1. While multiple faults is a realistic scenario, there is no framework for applying PFA in a multiple-faults setting that is as efficient as a single fault setting and does not require knowledge of the value of the fault. It is important to note that multiple persistent faults may occur during practical fault injection, especially as the technology node shrinks faster than the fault injection capability.
2. Because the PFA cannot uniquely retrieve the secret key, especially when there is a limited amount of data available, it is critical to have an efficient method for obtaining the correct key in the ciphertext-only scenario rather than utilizing a pair of plaintext-ciphertext in the known-plaintext scenario. To the best of our knowledge, there is no efficient and generalized framework applicable to all SPN ciphers to uniquely retrieve the correct key from a given set of key candidates in the ciphertext-only scenario under the assumption of multiple persistent faults occurring, with the exception of [ESP20], which requires a quite high runtime.

Our contribution is twofold. In the first part of our twofold contribution, we propose a novel technique that enables us to significantly reduce the number of remaining key candidates up to 2^n for an $n \times n$ Sbox in the cipher. The main features of the proposed methods are as follows: a) It does not rely on knowledge about the value or the location of the fault. b) It can be effectively extended to the cases where only an extremely limited number of ciphertexts is available, e.g. less than 500 ciphertexts, while keeping the number of remaining candidates within a very reasonable bound. The main results of this part are compared with noted previous works in Table 1. In contrast to previous works, our methods are parametric. Hence, Table 1 includes only the selected results of this paper. For more instances, we refer to Section 3.

In the second part, we introduce two generalized and efficient methods for determining the correct key among a set of key candidates in the ciphertext-only scenario. These methods are described in Section 4 and target ciphers which use large Sboxes and lightweight ciphers with small Sboxes. Our experimental results demonstrate that the key-recovery processes can retrieve the correct key with the success probability one in a very short time.

Besides, we performed practical electromagnetic fault injection experiments which are reported in Section 5. The experiments were performed on the 32-bit ARM Cortex-M4 microcontroller running both `AES` and `LED`. The experiments validate that faulting multiple Sbox elements is more likely than faulting single elements. Moreover, the key recovery is validated on the resultant ciphertext.

The proposed method is validated on block ciphers with different block sizes and in different scenarios based on the number of faults. In particular, we tested on `AES`-128 and `LED`-64 to represent the Sbox of 8×8 and 4×4 respectively. The source code for our simulations in pure `Python3` language is publicly available at address below: https://github.com/hadipourh/faultyaes.

Table 1: Summary of multiple persistent fault analysis on AES which are derived from the theory, †: It is assumed the faulty values are known, the upper bound for the number of ciphertexts is estimated 2000 in [ZLZ+18].

Reference	Ciphertexts	Remaining Key Candidates	Number of Faults
[ZLZ+18]†	2000	2^{16}	2
[ESP20]	7775	2^{50}	2
Section 3.4	1552	2^{23}	2
[ZLZ+18]†	2000	2^{50}	8
[ESP20]	2008	2^{50}	8
Section 3.4	1509	2^{8}	8
Section 3.5	671	$2^{14.56}$	8
[ZLZ+18]†	2000	2^{64}	16
[ESP20]	1643	2^{50}	16
Section 3.4	1448	2^{8}	16
Section 3.5	477	$2^{24.52}$	16

2 Preliminaries

2.1 Persistent Fault Analysis

Following the introduced PFA model by Zhang et al. [ZLZ+18], in this paper, we consider an R-round word-oriented Substitution-Permutation Network (SPN) cipher $E_K(P)$ which accepts two inputs: b-bit plaintext block P and k-bit key K. We denote the ciphertext by C. The encryption of the cipher can be described as an iteration of a function called round $F_{sk_r}(X)$, where sk_r denotes the r-th round key for $1 \leq r \leq R$. The b-bit state X consists of L words of the same size $n = b/L$ denoted by $X[i]$ where $0 \leq i \leq L-1$.

Let $x_r[j]$ and $y_r[j]$ denote the j^{th} word input and output of the substitution layer in the r-th round, respectively. Hence, the j^{th} word of the ciphertext can be calculated as $C[j] = y[j] \oplus sk_R[j]$, where $sk_R[j]$ is the j^{th} word of the last round key. Let us assume that the injected fault alters the correct value v to the faulty value $= v^* = v \oplus \Delta$ where S^* denotes the faulty look-up table of Sbox. Hence, the value of v no longer appears in $y_R[j]$, while the value of v^* is expected to appear twice as usual in $y[j]$. As a result, the probability distribution of $y_R[j]$ and consequently the probability distribution of $C[j]$ is not uniform. Hence, the adversary collects N ciphertexts and calculates the statistic distribution of each byte of ciphertext by counting the appearance of each of the possible values in $C[j]$ for all $0 \leq j \leq L-1$. If a large enough number of ciphertexts N is available, the adversary is able to find the minimal and maximal number of counts uniquely. We denote the value that is not observed in $C[j]$ at all by $C_{min}[j]$ and denote the value that appears more than other values by $C_{max}[j]$. Finding $C_{min}[j]$ and $C_{max}[j]$ can be used to derive $sk_R[j]$ in a variety of ways. If v is known and N is large enough, $sk_R[j]$ can be retrieved directly from $C_{min}[j]$ as $sk_R[j] = C_{min}[j] \oplus v$. Similarly, given v^*, the adversary can also retrieve $sk_R[j]$ directly from $C_{max}[j]$ as $sk_R[j] = C_{max}[j] \oplus v^*$.

In the aforementioned methods, the adversary needs to know the exact position or the value of the fault. Besides, a large number of faulty ciphertexts is required to find $C_{min}[j]$ and $C_{max}[j]$ uniquely. In response to these challenges, Zhang et al. adopted the main principles of the original PFA attack but made use of Maximum Likelihood Estimation to estimate $C_{min}[j]$ without knowing the fault position or the value of the fault [ZZJ+20].

2.2 Limitations of PFA in Multiple-Faults Setting

An extension of the proposed technique in [ZZJ+20] to the case with multiple faults is possible, but this comes with the cost of assuming the adversary knows the value of the fault. Recently, Engels et al. [ESP20] proposed Statistical Persistent Fault Analysis (SPFA) that is applicable to a block cipher in the multiple-faults setting while the attacker does not need to know the value of the fault. The proposed method can be applied to any number of faults, but as we discuss later, its runtime is quite high.

The insurmountable bottleneck of existing PFA techniques in the multiple-faults scenario is the high entropy of residual keys, where an exhaustive search with almost infeasible time complexity is required to be performed to find the correct key. The key candidates cannot be reduced even by increasing the number of faulty ciphertexts. After performing the PFA based on the proposed techniques in [ZLZ+18, ZZJ+20], the number of key candidates cannot become less than λ^L where λ is the number of faults. For instance, independent of the available ciphertexts, at least 2^{48} and 2^{64} candidates will remain after applying in PFA on AES for the case of multiple faults $\lambda = 8$ and $\lambda = 16$, respectively. Likewise, SPFA suffers from a similar limitation, since the correct key of AES can be retrieved with the time complexity of 2^{50} [ESP20] which is a considerable runtime.

2.3 Target Ciphers

Our methods are applicable to SPN block ciphers. However, we apply the proposed methods to two well-known ciphers, i.e., `AES` [DR99] and `LED` [GPPR11], to demonstrate the flexibility of our methods.

3 Our Framework for PFA with Multiple Faults

This section starts with a short introduction to the fault model which is considered in this paper. Then we give an overview of a specialized technique that makes performing PFA feasible in the multiple faults setting. Subsequently, we introduce our method for different scenarios based on the number of available ciphertexts.

3.1 Fault Model and Notation

The conventional PFA technique on table-based implementations of SPN ciphers and associated notation presented in Section 2.1 can be generalized to multiple faults. Suppose the correct values $\mathcal{V} = \{v_0, \cdots, v_{\lambda-1}\}$ are altered to the faulty values $\mathcal{V}^* = \{v_0^*, \cdots, v_{\lambda-1}^*\}$, respectively such that $\mathcal{V} \cap \mathcal{V}^* = \varnothing$.[1] Similar to the PFA with a single fault, we can take a similar discussion to conclude exactly λ values will never be observed in each word of the ciphertexts $C[j]$ where $0 \le j \le L-1$. To detect these impossible values, the adversary counts the appearance of each of the possible values in $C[j]$ for all $0 \le j \le L-1$, given N faulty ciphertexts. Assume that the adversary observes λ'_j minimum values for each word of the ciphertext $C[j]$. We denote the observable minimum values of the byte $C[j]$ by $C_{min_i}[j]$ where $0 \le i \le \lambda'_j$. In additionally, $\mathcal{D}_j = \{C_{min_i}[j] | 0 \le i \le \lambda'_j - 1\}$ defines the set of minimum values in the byte $C[j]$. Obviously, $sk_R[j] \oplus v$ equals to one of minimum values of the byte $C[j]$. If $\mathcal{V}$ is known and N is large enough, this relation can be used to retrieve $sk_R[j]$. There exist at least λ impossible values due to the existence of λ persistent faults. If N is insufficiently large, the number of observable minimum values λ'_j can be larger than λ. In the next part, we discuss the relation between the number of available faulty ciphertext and the number of minimum values.

3.2 The Effect of Available Data

The exact value of $\lambda'_j - 1$ depends on the number of available faulty ciphertexts N. If N is large enough, it is expected for each word of the ciphertext $C[j]$, exactly $\lambda'_j = \lambda$ minimum values will be observed. If a limited number of ciphertexts is available, each set $\mathcal{D}_j$ for $0 \le j \le L-1$ is expected to have more than λ elements. The estimation of λ'_j can be

[1] If v_i is altered to v_i^* such that $v_i^* \in \mathcal{V}$, this fault is an ineffective fault in PFA framework.

seen as an instance of the coupon collector's problem, a well-known problem in probability theory. In what follows, we aim to find a closed formula for the answer to this question: How many ciphertexts (N) are required to expect $m' = 2^n - \lambda'$ values to be observed given $m = 2^n - \lambda$ possible values for $C[j]$?

Let N denotes the number of ciphertexts needed to observe m' values, and t_i denotes the time required to observe the i^{th} value after $(i-1)^{th}$ value has been observed. Then the relation $N = t_1 + ... + t_{m'}$ holds. The expectation of t_i equals to $\frac{1}{p_i} = \frac{m}{m-i+1}$ because the probability of observing the i-th new value is $p_i = \frac{m-(i-1)}{m} = \frac{m-i+1}{m}$. Due to the linearity of expectations, the expectation of N can be computed as shown in Equation 1.

$$\begin{aligned} E(N) &= \mathrm{E}(t_1 + t_2 + \cdots + t_{m'}) = \mathrm{E}(t_1) + \mathrm{E}(t_2) + \cdots + \mathrm{E}(t_m) = \frac{1}{p_1} + \frac{1}{p_2} + \cdots + \frac{1}{p_{m'}} \\ &= \frac{m}{m} + \frac{m}{m-1} + \cdots + \frac{m}{m-m'+1} = m \cdot (\mathrm{H}_m - \mathrm{H}_{m-m'}), \end{aligned} \tag{1}$$

where $\mathrm{H_n}$ is the n-th harmonic number. Based on Equation 1, we can calculate the number of observable minimum values λ' given the number of ciphertexts N and the number of faults λ.

On the other hand, the number of observable minimum values λ' can be estimated by an exponential function given in Lemma 1.

Lemma 1. *Given the number of ciphertexts N and the number of faults λ, the number of observable minimum values λ' can be estimated by the exponential function $\lambda' = f(N, \lambda) = ae^{-bN} + c$ where $a = 2^n - \lambda$, $b = \frac{1}{2^n-\lambda}$ and $c = \lambda$.*

Proof. The harmonic number H_m can be approximated as $\ln(m) + \gamma + \frac{1}{2m}$ where γ is the Euler–Mascheroni constant. Hence, from Equation 1, we have:

$$\begin{aligned} &\frac{N}{m} = H(m) - H(m') = \ln m + \gamma + \frac{1}{2m} - \ln(m - m') - \gamma - \frac{1}{2(m-m')} \\ &\Rightarrow \ln(2^n - \lambda) - \ln(\lambda' - \lambda) = \frac{N}{2^n - \lambda} - \frac{1}{2(2^n-\lambda)} + \frac{1}{2(\lambda'-\lambda)} \simeq \frac{N}{2^n-\lambda} \\ &\ln\left(\frac{2^n-\lambda}{\lambda'-\lambda}\right) \simeq \frac{N}{2^n-\lambda} \Rightarrow \frac{2^n-\lambda}{\lambda'-\lambda} \simeq e^{\frac{N}{2^n-\lambda}} \Rightarrow \lambda' \simeq (2^n-\lambda)e^{-(\frac{1}{2^n-\lambda})\cdot N} + \lambda \end{aligned}$$

□

In the next parts, we demonstrate how the PFA can be performed efficiently in different scenarios to retrieve the secret key in a multiple-faults setting.

3.3 Core Idea

If N is large enough, for each word of the ciphertext $C[j]$ it is expected to observe exactly λ minimum values, which means $\lambda' = \lambda$ holds. Hence, each value of $C_{min_i}[j]$ is equal to the exclusive-or difference between one of the corrupted values $v \in \mathcal{V}$ and $sk_R[j]$. Consequently, the set of minimum values of the byte $C[j]$ is equal to

$$\mathcal{D}_j = \{v_i \oplus sk_R[j] | 0 \le i \le \lambda - 1\} = \mathcal{V} \oplus sk_R[j] \tag{2}$$

We let δ_j denotes the exclusive-or difference between $sk_R[0]$ and $sk_R[j]$. The crucial observation that can be deduced from Equation (2) is that there exist a relation between each set $\mathcal{D}_j$ and the set $\mathcal{D}_0$ as expressed in Equation (3).

$$\mathcal{D}_j = \mathcal{V} \oplus sk_R[j] = (\mathcal{V} \oplus sk_R[0]) \oplus (sk_R[0] \oplus sk_R[j]) = \mathcal{D}_0 \oplus \delta_j \tag{3}$$

where $\delta_j = sk_R[j] \oplus sk_R[0]$. In other words, given any $C_{min_i}[j]$ there exists i' such that the relation $C_{min_i}[j] \oplus C_{min_{i'}}[0] = \delta_j$ holds. We exemplify this important observation by considering a case with three persistent faults ($\lambda = 3$). Given enough ciphertexts, exactly two minimum values can be observed for each byte $C[j]$ where $0 \leq j \leq L-1$. Considering Equation (2) and without loss of generality, we assume $\mathcal{D}_0 = \{C_{min_0}[0] = v_0 \oplus sk_R[0], C_{min_1}[0] = v_1 \oplus sk_R[0], C_{min_2}[0] = v_2 \oplus sk_R[0]\}$, and $\mathcal{D}_1 = \{C_{min_0}[1] = v_0 \oplus sk_R[1], C_{min_1}[1] = v_1 \oplus sk_R[1], C_{min_2}[1] = v_2 \oplus sk_R[1]\}$. It is easy to observe that the following equations hold:

$$\delta_1 = C_{min_0}[1] \oplus C_{min_0}[0] = C_{min_1}[1] \oplus C_{min_1}[0] = C_{min_2}[1] \oplus C_{min_2}[0]$$

where $\delta_1 = sk_R[0] \oplus sk_R[1]$.

Equation (3) implies that a careful comparison between the set of minimum values of the byte $C[0]$ and the set of minimum values of the byte $C[j]$ might determine the value of δ_j. One should note that any information about the value of δ_j where $0 \leq j \leq L-1$ can directly lead to a decrease in the number of candidates for the last round key. In the case that the adversary could retrieve the values δ_j for all $0 \leq j \leq L-1$, it is sufficient for him/her to guess only the value of $sk_R[0]$, since other words of the last round key can be determined simply as $sk_R[j] = sk_R[0] \oplus \delta_j$.

The method described here may bear some resemblance to collision attacks that rely on internal collisions detected via side-channel leakage [SWP03, SLFP04]. Collisions in side-channel attacks imply that associated intermediate values are equivalent, whereas we create a set of relationships between the intermediate values using the ciphertext's impossible values. Additionally, collision attacks are relevant because they rely on certain key-dependent intermediate values, despite the fact that they are highly dependent on the cipher's structure and key schedule, whereas our approach makes no use of the cipher's internal specifications (like linear layer).

In the next parts, we will precisely describe how Equation (3) can be exploited to retrieve the values δ_j in different scenarios. In this work, we discern between two scenarios based on the number of available ciphertexts.

3.4 Multiple Persistent Faults with a Large Number of Ciphertexts

Assuming that a sufficient number of ciphertexts are available, each set $\mathcal{D}_j$ for $0 \leq j \leq L-1$ has exactly λ elements. This scenario corresponds to the model assumed in Section 3.3. Equation (3) implies that there exist exactly one value ℓ where $0 \leq \ell \leq \lambda - 1$ such that the relation $C_{min_0}[0] \oplus C_{min_\ell}[j] = \delta_j$ holds. We use this fact to retrieve the value of δ_j as illustrated in Algorithm 1. More precisely, we compute the value $\alpha_\ell = C_{min_0}[0] \oplus C_{min_\ell}[j]$ for all $0 \leq \ell \leq \lambda - 1$, and verify whether the relation $\mathcal{D}_j = \mathcal{D}_0 \oplus \alpha_\ell$ holds or not. If this relation satisfies for α_ℓ, then we conclude that $\delta_j = \alpha_\ell$ holds.

We refer to the below table as an example of performing Algorithm 1 for a byte-oriented block cipher ($n = 8$). The first and second rows represent the elements of $\mathcal{D}_0$ and $\mathcal{D}_1$, respectively.

$\mathcal{D}_0$	0x49	0x52	0x74	0x8C	0x94	0xA5	0xD2	0xE5
$\mathcal{D}_1$	0x53	0x6E	0x75	0x82	0xAB	0xB3	0xC2	0xF5

We can easily verify if $\alpha_1 = C_{min_0}[0] \oplus C_{min_0}[1] = 49 \oplus 53 = 1a$ is a candidate for the value of δ_1 by checking whether $\mathcal{D}_1 = \mathcal{D}_0 \oplus \alpha_1$ holds or not. Since $C_{min_1}[0] \oplus \alpha_1 = 52 \oplus 1a = 48 \notin \mathcal{D}_1$, we conclude $\delta_1 \neq \alpha_1$. Then we continue the process for the value $\alpha_2 = C_{min_0}[0] \oplus C_{min_1}[1] = 49 \oplus 6E = 27$. It is observed that $C_{min_\ell}[0] \oplus \alpha_2 \in \mathcal{D}_1$ for all $0 \leq \ell \leq \lambda - 1$. In other words, for any $C_{min_\ell}[0]$ there always exists an element in $\mathcal{D}_1$ such that the difference between the elements is equal to 27:

$$27 = 49 \oplus 6E = 52 \oplus 75 = 74 \oplus 53 = 8C \oplus AB = 94 \oplus B3 = A5 \oplus 82 = D2 \oplus F5 = E5 \oplus C2$$

Algorithm 1 Finding $\delta_j = sk_R[0] \oplus sk_R[j]$

```
Require: $\mathcal{D}_0 = \{C_{min_0}[0], \ldots, C_{min_{\lambda-1}}[0]\}$ and $\mathcal{D}_j = \{C_{min_0}[j], \ldots, C_{min_{\lambda-1}}[j]\}$
Ensure: $\Delta_j$ = Candidates for $\delta_j$
1:  $\Delta_j \leftarrow \emptyset$
2:  for $\ell = 0$ to $\lambda - 1$ do
3:      $\alpha_\ell = C_{min_0}[0] \oplus C_{min_\ell}[j]$
4:      cnt = 1
5:      $\mathcal{D} \leftarrow \mathcal{D}_j \setminus \{C_{min_l}[j]\}$
6:      for $i = 1$ to $\lambda - 1$ do
7:          $E \leftarrow C_{min_i}[0] \oplus \alpha_\ell$
8:          if $E \in \mathcal{D}$ then
9:              cnt ← cnt + 1
10:             $\mathcal{D} \leftarrow \mathcal{D} \setminus \{E\}$
11:     if cnt = $\lambda$ then
12:         $\Delta_j \leftarrow \Delta_j \cup \{\alpha_\ell\}$
13: return $\Delta_j$
```

It is easy to check that the relation $\mathcal{D}_1 = \mathcal{D}_0 \oplus \alpha_\ell$ holds only for $\ell = 1$. Hence, we can deduce that $\delta_1 = \alpha_1 = 27$.

3.4.1 Complexity Analysis

We assumed that the number of minimal values observed for each of the bytes $C[j]$ equals λ. In other words, we assumed $\lambda' = \lambda$. The expected value of N can be computed using the formula given by Section 3.2. By replacing $m = m' = 2^n - \lambda$ in Section 3.2, the expected value of N can be calculated as given in Equation (4).

$$\mathrm{E}(N) = \frac{m}{m} + \frac{m}{m-1} + \cdots + \frac{m}{1} = m \cdot \left(\frac{1}{m} + \frac{1}{m-1} + \cdots + \frac{1}{1}\right) = m \cdot \mathrm{H}_m \,. \tag{4}$$

We observe in the case of two faults (i.e. $\lambda = 2$), Algorithm 1 returns two candidates for each value of δ_j. This observation comes from the fact that if the relations $C_{min_0}[0] \oplus C_{min_0}[j] = C_{min_1}[0] \oplus C_{min_1}[j]$ holds, then the relation $C_{min_0}[0] \oplus C_{min_1}[j] = C_{min_1}[0] \oplus C_{min_0}[j]$ holds as well. In this case, it is not possible to determine δ_j uniquely. The attacker needs to guess the value of $sk_R[0]$ and calculate $sk_R[j] = sk_R[0] \oplus \delta_j$. Since there are two candidates for each value of δ_j, the number of candidates for the last round key equals to

$$\tau = 2^n \times 2^{L-1} = 2^{n+L-1} \quad \text{if} \quad \lambda = 2 \tag{5}$$

In case the number of faults is larger than two (i.e. $\lambda \geq 3$), the condition in line 11 of Algorithm 1 does not necessarily hold if $\alpha_\ell \neq \delta_j$. The probability that a random n-bit value α_ℓ satisfies the condition in line 11 of Algorithm 1 can be estimated by

$$\frac{\lambda-1}{2^n} \times \frac{\lambda-2}{2^n} \times \cdots \times \frac{1}{2^n} = \frac{(\lambda-1)!}{2^{n(\lambda-1)}} \tag{6}$$

This probability decreases with the number of faults. Hence, the success probability of our method always increases with the number of faults. We remind that the value of δ_j can be retrieved by Algorithm 1 by considering all α_ℓ where $0 \leq \ell \leq \lambda - 1$. Hence, the expected number of values for δ_j (i.e., $|\Delta_j|$) can be estimated by Equation (7) where $1 \leq j \leq L - 1$.

$$1 + (\lambda - 1) \times \frac{(\lambda-1)!}{2^{n(\lambda-1)}} \tag{7}$$

It is worth noting that for typical values of λ, and n the terms $1 + \frac{(\lambda-1)(\lambda-1)!}{2^{n\cdot(\lambda-1)}}$ is almost equal to 1. The adversary needs to only guess the n-bit $sk_R[0]$ as the remaining part of

sk_R, i.e., $sk_R[j]$ for $1 \leq j \leq \lambda - 1$ can be uniquely determined from $sk_R[j] = sk_R[0] \oplus \delta_j$. To sum up, the number of remaining candidates for the last round key equals to

$$\tau = 2^n \times [1 + (\lambda - 1) \times \frac{(\lambda - 1)!}{2^{n(\lambda-1)}}]^{L-1} \quad \text{if} \quad \lambda \geq 3 \tag{8}$$

Table 2 includes the number of required ciphertexts derived from Equation (4), and the number of remained key candidates derived from Equation (5) in case of $\lambda = 2$ and Equation (8) in case of $\lambda \geq 3$ for an SPN cipher, some typical values of λ and λ', when $n = 8$ and $L = 16$. It can be observed that in all cases the number of key candidates is significantly less than the number of possible candidates for the last subkey, i.e., $2^{n \cdot L}$. Besides, the number of required ciphertexts decreases slightly as the number of faults increases.

Table 2: Expected remained key candidates and required ciphertexts for an AES-like cipher.

λ	2	4	8	16
Key Candidates (τ)	2^{23}	2^{8}	2^{8}	2^{8}
Required Ciphertexts	$2^{10.6}$	$2^{10.58}$	$2^{10.56}$	$2^{10.50}$

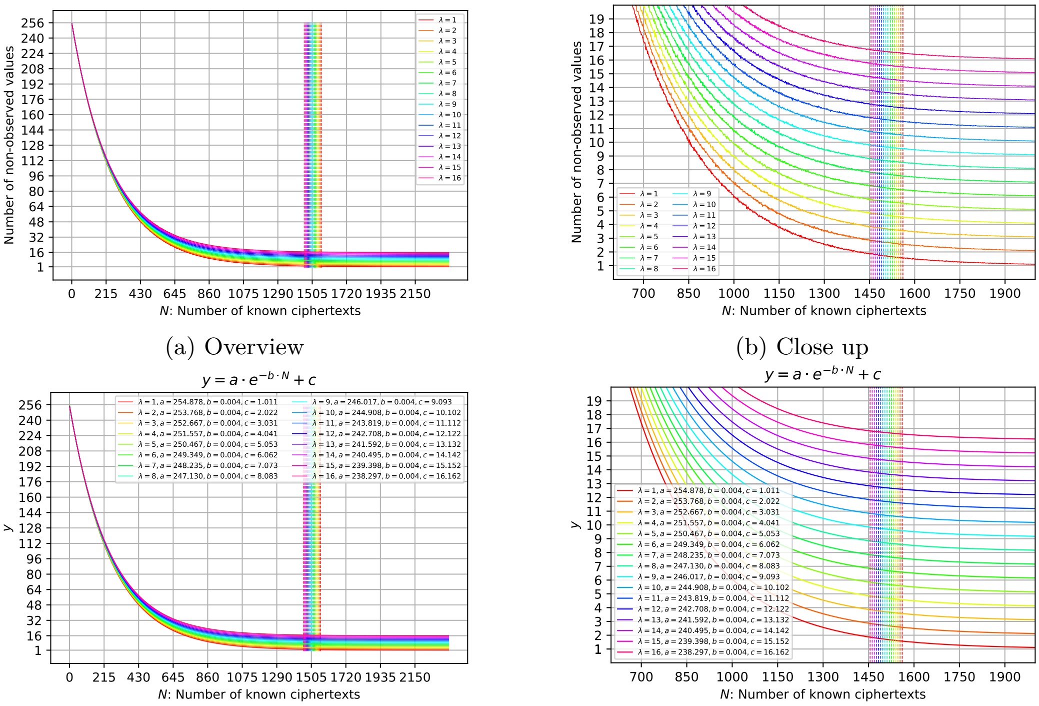
3.4.2 Simulation Results

We evaluated the accuracy of Equation (4) by selecting AES-128 as the target cipher. Assuming that λ faults are applied, we altered λ values in the AES Sbox at random such that there is no overlap between original and faulty values. Similar to the previous works, persistent faults were applied after deriving the sub-keys in our simulations. After encrypting N random plaintexts, we counted the number of distinct observed values at each output byte of ciphertext. We repeated this experiment for 100 random secret keys and counted the average number of observed values for an arbitrary byte of ciphertext. Figure 1(a) and Figure 1(b) illustrate the output of our simulations for the different numbers of faults $1 \leq \lambda \leq 16$. It can be seen that the number of non-observed values exponentially converges to the number of faults as the number of ciphertexts grows, such that for more than $m \cdot \mathrm{H}_m$ available ciphertexts, the number of non-observed values is almost equal to the number of faults. Moreover, the derived data from our simulation appropriately fits the exponential curve $a \cdot e^{-b \cdot N} + c$ such that a, b, and c are very close to the theoretical values derived from Lemma 1, as you can see in Figure 1(c) and Figure 1(d) which confirms the high accuracy of Lemma 1 in practice.

Considering AES-128 as the target cipher, we also implemented Algorithm 1 and experimentally evaluated the average number of candidates for δ_j returned by this algorithm where $1 \leq j \leq 15$. To do so, for the given number of faults and number of available ciphertexts, i.e., λ and N respectively, we applied λ random faults to generate N (faulty) ciphertexts as before. Next, feeding Algorithm 1 with the non-observed values in our experiment, we generate some candidates for δ_j where $1 \leq j \leq 15$. We repeated this experiment for 100 random secret keys to compute the average number of candidates returned by Algorithm 1 for an arbitrary output byte. When the number of ciphertexts was larger than $(m \cdot \mathrm{H}_m)$ where $m = 2^8 - \lambda$, for each δ_j, where $1 \leq j \leq 15$, Algorithm 1 returned only one candidate and two candidates for the case $\lambda \geq 3$ and $\lambda = 2$, respectively. These results confirm the estimation given in Equation (7) and the accuracy of Algorithm 1.

3.5 Multiple Persistent Faults with a Limited Number of Ciphertexts

If a limited number of ciphertexts is available, each set $\mathcal{D}_j$ for $0 \leq j \leq L - 1$ is expected to have more than λ elements. If we denote the size of the set $\mathcal{D}_j$ by λ'_j, then $\lambda'_j \geq \lambda$. In this case, the deterministic relation described in Equation (3) does not hold anymore. This



(a) Overview

(b) Close up

(c) Overview

(d) Close up

Figure 1: Average number of non-observed values at each output byte of ciphertext, given N ciphertexts in presence of λ persistent faults, where $1 \leq \lambda \leq 16$, and the target cipher is AES-128. Moreover, vertical lines specify the required number of ciphertexts according to Equation (4) for each λ.

comes from the fact that due to the limited number of available ciphertexts, $\mathcal{D}_j$ includes $\lambda'_j - \lambda$ elements that are not impossible. We discern two challenges in applying our initial method (described in Section 3.4). First, it is unclear whether the value $C_{min_0}[0]$ is an impossible value or not. Second, the relation $C_{min_i}[0] \oplus \delta_j \in \mathcal{D}_j$ does not always satisfy for all values of i. In this part, we take a sophisticated technique to circumvent these challenges in the case $\lambda'_j \geq \lambda$. The correct value of δ_j can be determined by utilizing the process illustrated in Algorithm 2 which is similar to a specialized technique proposed in Section 3.4 with two main changes. First, we utilize a trial and error approach to find the first impossible value $C_{min_k}[0] \in \mathcal{D}_0$ where k is an unknown value and $0 \leq k \leq \lambda'_0 - \lambda$. We make use of $C_{min_k}[0]$ as a base to determine the value of δ_j. Second, we extend the preliminary technique to the probabilistic setting. Instead of checking the relation $\mathcal{D}_j = \mathcal{D}_0 \oplus \alpha_\ell$, we look for the value of $\alpha_\ell = C_{min_k}[0] \oplus C_{min_\ell}[j]$ where the desired relation $C_{min_i}[0] \oplus \alpha_\ell$ holds at least λ times.

3.5.1 Complexity Analysis

Let us assume that $C_{min_k}[0]$ is an impossible value. We can take a similar discussion to conclude that there exists an unknown $0 \leq \ell \leq \lambda'_j - 1$ such that δ_j equals to α_ℓ where α_ℓ is defined as $\alpha_\ell = C_{min_k}[0] \oplus C_{min_\ell}[j]$. In the case that $\lambda = 2$, we expect the relation $C_{min_i}[0] \oplus \alpha_\ell \in \mathcal{D}_j$ to hold at least λ times for $0 \leq i \leq \lambda'_j - 1$ if $\alpha_\ell = \delta_j$ or $\alpha_\ell = \delta_j \oplus v_0 \oplus v_1$. In the case that $\lambda \geq 3$, we expect the relation $C_{min_i}[0] \oplus \alpha_\ell \in \mathcal{D}_j$ at least λ times holds for $1 \leq i \leq \lambda'_j$ if $\alpha_\ell = \delta_j$. For other values of α_ℓ, we expect the relation $C_{min_k}[0] \oplus \alpha_\ell \in \mathcal{D}_j$

Algorithm 2 Finding $\delta_j = sk_R[0] \oplus sk_R[j]$

Require: $\mathcal{D}_0 = \{C_{min_0}[0], \ldots, C_{min_{\lambda'_0 - 1}}[0]\}$ and $\mathcal{D}_j = \{C_{min_0}[j], \ldots, C_{min_{\lambda'_j - 1}}[j]\}$

Ensure: Δ_j = Candidates for δ_j

```
 1: $\Delta_j \leftarrow \emptyset$
 2: for $k = 0$ to $\lambda'_0 - \lambda$ do
 3:     $\mathcal{D}_0 \leftarrow \mathcal{D}_0 \setminus \{C_{min_k}[0]\}$
 4:     for $\ell = 0$ to $\lambda'_j - 1$ do
 5:         $\alpha_\ell = C_{min_k}[0] \oplus C_{min_\ell}[j]$
 6:         $\texttt{cnt} = 1$
 7:         $\mathcal{D} \leftarrow \mathcal{D}_j \setminus \{C_{min_\ell}[j]\}$
 8:         for all $d \in \mathcal{D}_0$ do
 9:             $E \leftarrow d \oplus \alpha_\ell$
10:             if $E \in \mathcal{D}_j$ then
11:                 $\texttt{cnt} \leftarrow \texttt{cnt} + 1$
12:                 $\mathcal{D} \leftarrow \mathcal{D} \setminus \{E\}$
13:         if $\texttt{cnt} \geq \lambda$ then
14:             $\Delta_j \leftarrow \Delta \cup \{\alpha_\ell\}$
15: return $\Delta_j$
```

holds on average x times with the probability given in Equation (9).

$$\begin{aligned}\Pr[\texttt{cnt} = x|\alpha_\ell \neq \delta_j] &= (1 - \frac{\lambda'_j}{2^n})^{\lambda'_0 - x} \times \frac{\lambda'_j - 1}{2^n} \times \cdots \times \frac{\lambda'_j - x + 1}{2^n} \times \binom{\lambda'_j - 1}{x - 1} \\ &= (1 - \frac{\lambda'_j}{2^n})^{\lambda'_0 - x} \times (x-1)! \times \binom{\lambda'_j - 1}{x - 1}^2 \times 2^{-n\cdot(x-1)}\end{aligned} \tag{9}$$

Hence the probability that the relation $C_{min_i}[0] \oplus \alpha_\ell \in \mathcal{D}_j$ holds more than λ times can be computed as Equation (10) where $\Pr[\texttt{cnt} = x|\alpha_\ell \neq \delta_j]$ can be calculated based on Equation (9).

$$\mathcal{P} = \sum_{x=\lambda}^{\lambda'_j} \Pr[\texttt{cnt} = x|\alpha_\ell \neq \delta_j] = \sum_{x=\lambda}^{\lambda'_j} (1 - \frac{\lambda'_j}{2^n})^{\lambda'_0 - x}(x-1)!\binom{\lambda'_j - 1}{x - 1}^2 2^{-n\cdot(x-1)} \tag{10}$$

In the case $C_{min_k}[0]$ is not an impossible value, the condition of line 13, i.e. $\texttt{cnt} \geq \lambda$, holds with probability $\mathcal{P}$ for the calculated α_ℓ where $0 \leq \ell \leq \lambda'_j$. In the worst case, the first impossible value in the set $\mathcal{D}_0$ is $C_{min_{\lambda'_j - \lambda}}$. In the case $C_{min_k}[0]$ is an impossible value, the condition of line 13 holds with probability $\mathcal{P}$ for $\alpha_\ell \neq \delta_j$ (i.e. $(\lambda'_j - 2)$ times and $(\lambda'_j - 1)$ times for $\lambda = 2$ and $\lambda \geq 3$, respectively). To sum up, the upper bound of the expected number of values for δ_j can be estimated by Equation (11) where $\mathcal{P}$ is given in Equation (10).

$$\begin{cases} 2 + [(\lambda'_j - 2) + (\lambda'_j - \lambda)\lambda'_j] \times \mathcal{P}, & \text{if } \lambda = 2 \\ 1 + [(\lambda'_j - 1) + (\lambda'_j - \lambda)\lambda'_j] \times \mathcal{P}, & \text{if } \lambda \geq 3 \end{cases} \tag{11}$$

Equation (10) provides an estimate of the number of candidates for δ_j under the worst-case scenario of no repetition. As a result, the number of candidates may be reduced in practice, particularly when $\mathcal{P}$ is large.

The adversary needs to guess the n-bit value of $sk_R[0]$. Then, he/she finds other bytes of the last round key based on the relation $sk_R[j] = sk_R[0] \oplus \delta_j$. As a consequence, the number of remaining candidates for the last round key equals to

$$\tau = \begin{cases} 2^n \times \lceil [2 + [(\lambda'_j - 2) + (\lambda'_j - \lambda)\lambda'_j] \times \mathcal{P}]^{L-1} \rceil, & \text{if } \lambda = 2 \\ 2^n \times \lceil [1 + [(\lambda'_j - 1) + (\lambda'_j - \lambda)\lambda'_j] \times \mathcal{P}]^{L-1} \rceil, & \text{if } \lambda \geq 3 \end{cases} \tag{12}$$

Table 3 includes the number of required ciphertexts derived from Section 3.2, and the number of remained key candidates derived from Equation (12) for an SPN cipher, and some typical values of λ and λ', when $n = 8$ and $L = 16$ (like AES). It also shows that $\lambda' = 2 \cdot \lambda$ is cost-efficient.

Table 3: Expected number of remained key candidates and required ciphertexts in an SPN cipher for some typical values of λ and λ', when $n = 8$ and $L = 16$.

λ	2			4			8			16	
λ'	4	6	8	8	12	16	16	24	32	32	48
$\mathcal{P}$	0.0343	0.0917	0.170	3×10^{-6}	0.002	0.040	10^{-6}	0.001	0.028	10^{-8}	0.001
Required Ciphertexts	$2^{10.19}$	$2^{10.00}$	$2^{9.86}$	$2^{9.98}$	$2^{9.74}$	$2^{9.56}$	$2^{9.70}$	$2^{9.39}$	$2^{9.16}$	$2^{9.32}$	$2^{8.90}$
Key Candidates (Theory)	$2^{26.43}$	$2^{40.87}$	$2^{60.25}$	$2^{8.33}$	$2^{20.23}$	$2^{56.12}$	$2^{8.01}$	$2^{15.39}$	$2^{73.20}$	$2^{8.00}$	$2^{29.93}$
Key Candidates (Simulation)	$2^{25.07}$	$2^{34.54}$	$2^{50.07}$	$2^{8.71}$	$2^{16.11}$	$2^{40.67}$	$2^{8.00}$	$2^{12.78}$	$2^{55.62}$	$2^{8.00}$	$2^{29.94}$

3.5.2 Simulation Results

To experimentally validate the correctness as well as the quality of Algorithm 2, we carried out several random experiments to see how many candidates this algorithm returns on average when a limited number of ciphertexts are available. More precisely, choosing `AES`-128 as the target cipher as before, for a given number of faults and a number of non-observed values, namely λ and λ' respectively, we firstly apply λ random faults and generate N (faulty) ciphertexts such that $N = m \cdot (\mathrm{H}_m - \mathrm{H}_{m-m'})$ according to Section 3.2. Next, detecting the non-observed values at each output byte, i.e., $\mathcal{D}_j$, we feed Algorithm 2 by $(\mathcal{D}_0, \mathcal{D}_j)$ to derive some candidates for δ_j, where $1 \leq j \leq 15$, and $|\mathcal{D}_0| = |\mathcal{D}_j| = \lambda'$. Iterating this experiment for 100 randomly chosen secret keys we compute the average number of candidates returned by Algorithm 2. Assuming that $|\mathcal{D}_0| = |\mathcal{D}_j| = \lambda'$ holds for the input of Algorithm 2, we computed the average number of candidates returned by this algorithm for different values of λ and λ'. As shown in Table 3, Equation (11) and Equation (12) provide an accurate estimate of the output size of Algorithm 2 and, consequently, the number of key candidates when the probability $\mathcal{P}$ is small. However, when $\mathcal{P}$ is large (as indicated by the blue color in Table 3), simulation results reveal that the attack performs significantly better than Equation (11) predicts. This discrepancy arises because increasing $\mathcal{P}$ results in an increase in the number of repeats.

3.6 Side Information about Faults

In this part, we demonstrate that the proposed framework can also leak side information about the injected faults.

3.6.1 Determining the Number of Faults (λ)

The assumption of knowing the number of faults usually makes sense as it can be estimated via a profiling phase by the attacker. However, the profiling phase is not necessarily possible in all applications. An interesting property of the proposed methods described in Section 3.4 and Section 3.5 is that the number of faults can be determined by the adversary as a piece of side information.

If the attacker is able to query enough ciphertexts as it is assumed in Section 3.4, then he can gradually increase the number of ciphertexts and monitor the size of $\mathcal{D}_j$ for $0 \leq j \leq L-1$. The set of minimum values of the byte $C[j]$ (i.e. size of $\mathcal{D}_j$) decreases as the number of ciphertexts increases. After a while, the number of ciphertexts does not affect the size of $\mathcal{D}_j$. At this point, the adversary can determine the number of faults as $\lambda = min\{|\mathcal{D}_j|\}$ where $0 \leq j \leq L-1$.

If the attacker has access to a limited number of ciphertexts, then he/she needs to perform Algorithm 2. As it is mentioned in Section 3.5, the corresponding value of `cnt` in the case of $\alpha_\ell = \delta_j$ becomes equal or larger than λ. In other words, if we denote the maximum value of cnt during the execution of Algorithm 2 for retrieving the δ_j by $\texttt{cnt}_j^{max}$, then the relation $\lambda \leq \texttt{cnt}_j^{max}$ holds. Hence, a similar process described in Algorithm 2 can find not only δ_j based on the $\texttt{cnt}_j^{max}$ but it also determines an upper bound for the number of faults that means $\lambda \leq min\{\texttt{cnt}_j^{max}\}$.

During our experiments to verify the proposed method in Section 3.5, we noticed that appearing an α_ℓ such that $\texttt{cnt}(\alpha_\ell) > \lambda$ is almost impossible in Algorithm 2. In other words, most of the time we observed that $\texttt{cnt}(\alpha_\ell) \leq \lambda$ for each α_ℓ, which leads us to the correct value of λ.

3.6.2 Determining Values of Faults

In this part, as another interesting property of the proposed methods in Section 3.4 and Section 3.5, we demonstrate that some information about $\mathcal{V}$ and $\mathcal{V}^*$ can be obtained by the adversary. Let us assume that by performing Algorithm 1 or Algorithm 2, τ key candidates K^i are obtained where $1 \leq i \leq \tau$. Besides, we are given $\mathcal{D}_j$ for $0 \leq j \leq L-1$. We assume $\mathcal{D}_u$ has the minimum size among all $\mathcal{D}_j$ for $0 \leq j \leq L-1$. For each key candidate K^i, the corresponding $\mathcal{V}^i$ can be computed as $\mathcal{V}^i = K^i[u] \oplus \mathcal{D}_u$ for $1 \leq i \leq \tau$. In this way, the τ returned candidates for the secret key are converted to τ candidates tuples of $(K^i, \mathcal{V}^i)$ for $1 \leq i \leq \tau$.

On the other hand, the probability of $C[j]$ when $C[j] \in \mathcal{V}^* \oplus sk_R[j]$ is higher compared to other cases (since $\Pr(C[j]) \geq 2 \times 2^{-n} | C[j] \in \mathcal{V}^* \oplus sk_R[j]$). We make use of this non-uniform distribution to determine the $C_{max}[j]$ candidates, i.e. $\mathcal{D}_j^*$ for $0 \leq j \leq L-1$. It is clear $|\mathcal{D}_j^*| \leq \lambda$ and for given enough ciphertexts we know the exact value of λ. However, we overestimate each $\mathcal{D}_j^*$ to do not miss any member of $\mathcal{V}^* \oplus sk_R[j]$. Hence, we include λ'' values of $C_{max}[j]$ in $\mathcal{D}_j^*$, λ'' can be fine-tuned dependent on the number of available ciphertexts. Next, we can use Algorithm 3 to narrow the members and determine the correct set $\mathcal{D}_0^*$. Given the $(K^i, \mathcal{V}^i)$ candidates for $1 \leq i \leq \tau$ and $\mathcal{D}_0^*$, we define $\mathcal{V}^{*i} = K^i[0] \oplus \mathcal{D}_0^*$ for $1 \leq i \leq \tau$. In this way, τ candidates of $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$ are determined and could be used later in the key recovery process, i.e. Section 4.3.

Implementing Algorithm 3 and considering the AES-128 as the target, we also performed several random experiments to confirm the correctness as well as the quality of Algorithm 3. In every single experiment, after applying λ random faults we produce a sufficiently large number of random faulty ciphertexts to collect non-observed values at each output byte, i.e., $\mathcal{D}_j$ for $0 \leq j \leq 15$. By sufficiently large number, we mean larger than $m \cdot \mathrm{H}_m$, where $m = 2^n - \lambda$. Next feeding Algorithm 1 by the derived $(\mathcal{D}_0, \mathcal{D}_j)$ we find some candidates for each δ_j, where $1 \leq j \leq 15$. It should be recalled that Algorithm 1 returns a unique value on average when a sufficiently large number of ciphertexts are available. Lastly, we call Algorithm 3 to retrieve the $\mathcal{D}_0^*$. Throughout our experiments, capturing the $\texttt{cnt}[x]$ for all $0 \leq x \leq 255$, we observed that all members of correct $\mathcal{D}_0^*$ can be simply distinguished from the other values, since $\texttt{cnt}[x]$ for $x \in \mathcal{D}_0^*$ was always much higher in comparison to the $x \notin \mathcal{D}_0^*$ for correct $\mathcal{D}_0^*$ in all of our random experiments. The used filter in line 8 of Algorithm 3 is to consider the possible overlap in $\mathcal{V}^*$. Consequently, Algorithm 3 returns the correct $\mathcal{D}_0^*$ in practice, when a sufficiently large number of ciphertexts are available.

4 Key-recovery Process for Remaining Key Candidates

In this section, we propose generalized techniques for the key-recovery process in the ciphertext-only model over the remaining key candidates after the PFA attack.

Algorithm 3 Finding $\mathcal{D}_0^*$

```
Require: $\mathcal{D}_0^* = \{C_{max_0}[0], \ldots, C_{max_{\lambda_0''-1}}[0]\}$ and $\mathcal{D}_j^* = \{C_{max_0}[j], \ldots, C_{max_{\lambda_j''-1}}[j]\}$ and $\delta_j$ for $1 \leq j \leq L-1$
Ensure: $\mathcal{D}_0^*$
for $j = 1$ to $L-1$ do
    $\mathcal{D}_j^* \leftarrow \mathcal{D}_j^* \oplus \delta_j$
for $x = 0$ to $2^n - 1$ do
    $\texttt{cnt}[x] = 0$
for $j = 0$ to $L-1$ do
    for $x \in \mathcal{D}_j^*$ do
        $\texttt{cnt}[x] \leftarrow \texttt{cnt}[x] + 1$
$\mathcal{D} \leftarrow \{x \in \{0, \ldots, 2^n - 1\} \mid \texttt{cnt}[x] \geq \frac{L}{2}\}$
$\mathcal{D} \leftarrow [x_0, \ldots, x_{|\mathcal{D}|-1} \mid \texttt{cnt}[x_0] \geq \cdots \geq \texttt{cnt}[x_{|\mathcal{D}|-1}];\ x_i \in D]$
return at most $\lambda$ first elements of $\mathcal{D}$ as $\mathcal{D}_0^*$
```

4.1 Attack Scenarios

As we demonstrated in Section 3.6.2, the attacker can obtain corresponding $\mathcal{V}$ for each possible key candidate as a piece of side information by performing Algorithm 1 or Algorithm 2. Because retrieving $\mathcal{V}^*$ is not always possible, we distinguish between two situations based on whether the attacker knows the exact set of $\mathcal{V}^*$ for a particular key candidate or not. Determining the set of $\mathcal{V}^*$ depends on available ciphertexts and the size of the utilized Sbox. If the adversary only has access to a small number of ciphertexts, he/she can only use the $\mathcal{V}$. Having a larger number of available ciphertexts, on the other hand, might indicate that the adversary is aware of both $\mathcal{V}$ and $\mathcal{V}^*$. In contrast to a cipher with a bigger Sbox, such as `AES`, a cipher with a smaller Sbox, such as `LED`, requires a fewer number of ciphertexts to produce $\mathcal{V}^*$. With few ciphertexts, it is difficult to identify the precise set of $\mathcal{V}^*$ and hence D_j^* for a cipher with a bigger Sbox.

In case that a limited number of ciphertexts is available to the adversary, he/she has access only to the $\mathcal{V}$. In contrast, having a larger number of available ciphertexts can be interpreted that both $\mathcal{V}$ and $\mathcal{V}^*$ are known to the adversary. On the other hand, a smaller number of ciphertexts is required to obtain $\mathcal{V}^*$ for a cipher with a smaller Sbox, e.g. `LED`, compared to a cipher with a larger Sbox, e.g. `AES`. In contrast, for a cipher with a larger Sbox it is hard to determine the exact set of $\mathcal{V}^*$ and therefore D_j^* with few ciphertexts.

Standard cipher designers often pick a larger Sbox but a smaller number of rounds. In contrast, lightweight block ciphers often have a tiny Sbox, but the round function is repeated across a larger number of rounds. Dependent on this distinction, we suggest two distinct techniques for the key-recovery process in Section 4.2 and Section 4.3, based on whether the adversary knows $\mathcal{V}^*$ or not. This technique gives the attacker the option of selecting one of the approaches dependent on the application. We should point out that in these approaches, the attacker has no knowledge of the location or values of the faults.

4.2 Key-recovery Attack Based on $\mathcal{V}$

Let us assume the adversary is given N faulty ciphertexts $C_1, \ldots, C_N$ that are produced using an identical faulty Sbox S^*. The input and the output of the Sbox layer of the r^{th} round are designated by x_r and y_r, each of them consists of L words of the same size $n = b/L$. Similar to previous research, we assume that the cipher's key schedule is invertible and that $|sk_R| = k = b$, i.e. given sk_R it is possible to determine the master key K uniquely. Furthermore, we assume that the round keys are precomputed and are unaffected by faults.

By performing PFA as described in Section 3, the adversary has τ candidates of $(K^i, \mathcal{V}^i)$ where $\mathcal{V}^i = \{v_0^i, \cdots, v_{\lambda-1}^i\}$. The correct key and corresponding $\mathcal{V}$ are unclear to the adversary. However, it is apparent that the correct values $\mathcal{V} = \{v_0, \cdots, v_{\lambda-1}\}$ were not produced in the output of Sboxes throughout the encryption process. If the candidate $(K^i, \mathcal{V}i)$ is a correct pair and the state is unaffected by faults, the elements of $\mathcal{V}i$ should not appear after the Sboxes during the decryption of ciphertexts under K^i. This distinguisher appears to be beneficial in locating the correct key among the key candidates. However, this fact cannot be used straightforwardly, since the process of decryption is challenging in the PFA model. Although the Sbox used in an SPN block cipher is always invertible, this is not true for $S^*(.)$. More precisely, if $\beta \notin \{\mathcal{V} \cup \mathcal{V}^*\}$ then we can uniquely determine $S^{*-1}(\beta)$; if $\beta \in \mathcal{V}^*$ then there will be more than one possible value for $S^{*-1}(\beta)$ and if $\beta \in \mathcal{V}$ then $S^{*-1}(\beta) = \perp$.

We make use of the aforementioned properties to propose a sophisticated key-recovery framework in the probabilistic setting. The process is described in Algorithm 4. For any candidate $(K^i, \mathcal{V}^i)$ we allocate a counter $\texttt{cnt}(K^i, \mathcal{V}^i)$ and set them zero. We decrypt each C_j under the key candidate K^i round by round. In the r^{th} round , for $1 \leq r \leq R-1$, we calculate y_r and increase the counter $\texttt{cnt}(K^i, \mathcal{V}^i)$ if the elements of $\mathcal{V}^i$ do not appear in y_r. More precisely, we increase the counter $\texttt{cnt}(K^i, \mathcal{V}^i)$ if $\{y_r[0], \cdots, y_r[L-1]\} \cap \mathcal{V}^i = \varnothing$. Finally, a pair $(K^i, \mathcal{V}^i)$ with the highest counter value is returned as our guess for the correct pair.

4.2.1 Attack Analysis

In what follows, we estimate the expectation of the counter $\texttt{cnt}(K^i, \mathcal{V}^i)$ for the wrong key and correct key to proving this can be used as a strong distinguisher for determining the correct key among the key candidates.

Given a wrong pair $(K^i, \mathcal{V}^i)$, we expect the calculated value y_r will be random for $1 \leq r \leq R-1$. Hence, the probability of achieving a y_r such that $\{y_r[0], \cdots, y_r[L-1]\} \cap \mathcal{V}^i = \varnothing$ is $(1 - \frac{\lambda}{2^n})^{L \cdot r'}$, where $r' = L - r$ (Figure 2a). As a consequence, the expected value of $\texttt{cnt}(K^i, \mathcal{V}^i)$ for a wrong pair $(K^i, \mathcal{V}^i)$ can be estimated as given in Equation (13).

$$\texttt{cnt}(K^i, \mathcal{V}^i) \approx N \cdot \sum_{r=1}^{R-1} p^r, \quad \text{where} \quad p = (1 - \frac{\lambda}{2^n})^L \tag{13}$$

Similarly, we can calculate the expected value of $\texttt{cnt}(K^i, \mathcal{V}^i)$ for the correct pair $(K^i, \mathcal{V}^i)$. Given the correct pair $(K^i, \mathcal{V}^i)$, $\texttt{cnt}(K^i, \mathcal{V}^i)$ can be increased under two circumstances as is depicted in Figure 2b. First, we consider a situation in which $\{y_h[0], \cdots, y_h[L-1]\}$ have not been alerted by the introduced faults for $R-1 \leq h \leq r-1$, i.e. $\{S(x_h[0]), \cdots, S(x_h[L-1])\} \cap \mathcal{V}^i \neq \varnothing$ for $R-1 \leq h \leq r-1$. In this case, $\texttt{cnt}(K^i, \mathcal{V}^i)$ is increased with probability $(1 - \frac{\lambda}{2^n})^{L \cdot r'}$ in the r^{th} round. Hence, the expected increase in $\texttt{cnt}(K^i, \mathcal{V}^i)$ under this circumstance (left branch in Figure 2b) can be estimated as given in Equation (14).

$$N \cdot \sum_{r=1}^{R-1} p^r \tag{14}$$

Second, we consider a situation in which $\{y_h[0], \cdots, y_h[L-1]\}$ have been altered by the introduced faults for $r-1 \leq h \leq R$ but $\{y_r[0], \cdots, y_r[L-1]\} \cap \mathcal{V}^i = \varnothing$. For instance, in case of y_{R-1}, if $\{S(x_R[0]), \cdots, S(x_R[L-1])\} \cap \mathcal{V} \neq \varnothing$ but $\{y_{R-1}[0], \cdots, y_{R-1}[L-1]\} \cap \mathcal{V} = \varnothing$ then $\texttt{cnt}(K^i, \mathcal{V}^i)$ is increased with probability $p_1 = (1-p) \cdot p$. As can be followed from Figure 2a, this observation can be extended to other rounds as well. In general, the counter $\texttt{cnt}(K^i, \mathcal{V}^i)$ is increased again with the probability of $p_{r'} = (p^{r'-1} \cdot (1-p) + p_{r'-1}) \cdot p = r' \cdot p^{r'} \cdot (1-p)$ over the r^{th} round, where $r' = L - r$. Consequently, the expected increase in $\texttt{cnt}(K^i, \mathcal{V}^i)$ under this circumstance (right branch in Figure 2b) can be estimated as given in Equation (15).

$$N \cdot \sum_{r=1}^{R-1} r \cdot p^r \cdot (1-p) \tag{15}$$

Algorithm 4 Key-recovery attack on multiple fault model, given candidates of $(K^i, \mathcal{V}^i)$

Require: $C_1, \ldots, C_N$ and τ candidates of $(K^i, \mathcal{V}^i)$
Ensure: Correct $(K^i, \mathcal{V}^i)$

```
1:  for $i \leftarrow 1$ to $\tau$ do
2:      $\texttt{cnt}(K^i, \mathcal{V}^i) = 0$
3:      for $l \leftarrow 1$ to $N$ do
4:          for $r \leftarrow R-1$ down to 1 do
5:              $y_r \leftarrow C_l^{-r,K^i}$
6:              for $j \leftarrow 0$ to $L-1$ do
7:                  if $y_r[j] \in \mathcal{V}^i$ then
8:                      Go to the line 3 (next $l$)
9:              $\texttt{cnt}(K^i, \mathcal{V}^i) \leftarrow \texttt{cnt}(K^i, \mathcal{V}^i) + 1$
10:     return Higher ranked $\{\texttt{cnt}(K^i, \mathcal{V}^i)\}$
```

By considering both aforementioned circumstances presented with corresponding expected values presented in Equation (14) and Equation (15), the expected value of $\texttt{cnt}(K^i, \mathcal{V}^i)$ for a correct pair $(K^i, \mathcal{V}^i)$ can be estimated as given in Equation (16).

$$\texttt{cnt}(K^i, \mathcal{V}^i) \approx N \cdot \sum_{r=1}^{R-1} p^r + N \cdot \sum_{r=1}^{R-1} r \cdot p^r \cdot (1-p) \tag{16}$$

It is worth noting in the above estimation while decrypting an affected y_r under a correct $(K^i, \mathcal{V}^i)$, we assumed the whole y_{r-1} will be random. However, depending on the diffusion layer of the cipher and the number of affected words of y_r, it is possible that only part of y_{r-1} behaves randomly and the rest pass the verification deterministically. Hence, we expect a higher actual value for $\texttt{cnt}(K^i, \mathcal{V}^i)$ of the correct pair which enhances the distinguishability of the correct key recovery from the wrong keys. This fact is confirmed by our simulations which demonstrates the key-recovery attack works slightly better compared to our formulations. For more details about the simulation results, we refer to the next part (Section 4.2.2).

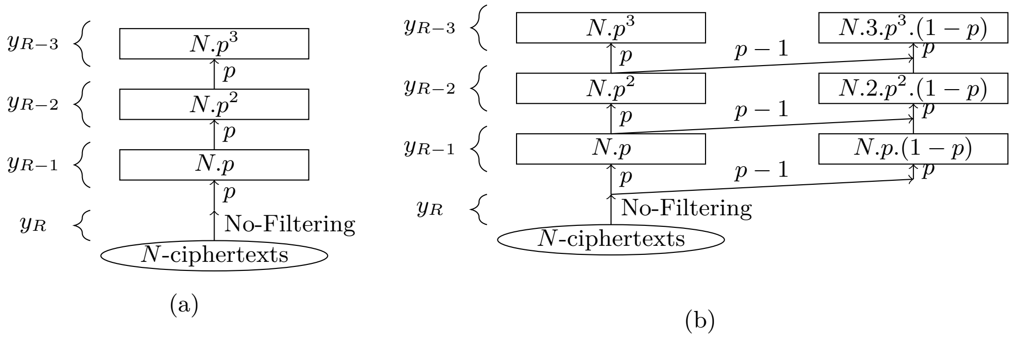


Figure 2: Expectation of added value to $\texttt{cnt}(K^i, \mathcal{V}^i)$ over the last four rounds; (a) wrong guess of $(K^i, \mathcal{V}^i)$; (b) correct guess of $(K^i, \mathcal{V}^i)$.

4.2.2 Simulation Results

To verify the described key-recovery attack, we consider AES-128. Table 4 represents numerical results for $1 \leq \lambda \leq 16$, where theoretical values are driven from the framework (Equation (16) and Equation (14)) and the experimental results are conducted based on taking the average over 100 random experiments. In each experiment, we chose a secret key at random, applied λ random faults, and generated N random faulty ciphertexts at first.

After that, we detect non-observed values at each output byte, i.e, $\mathcal{D}_j$ for $0 \le j \le 15$, and perform Algorithm 2 to derive the candidates for δ_j, where $1 \le j \le 15$. Then we obtained candidates for the last round key as well as its corresponding $\mathcal{V} = \mathcal{D}_0 \oplus sk_R[0]$. These steps provided the input of Algorithm 4, by calling which we could retrieve the correct key. We calculated the average of $\texttt{cnt}(K^i, V^i)$ for the correct as well as the wrong keys. We have not only observed that Algorithm 4 can uniquely retrieve the correct key, but the experimental results precisely match the theoretical expectations concerning the $\texttt{cnt}(K^i, V^i)$ for wrong keys (as it can be seen in Table 4). We also have observed that $\texttt{cnt}(K^i, V^i)$ for the correct key is even higher than the expected value according to Equation (16). There are some dependencies in the case of the correct key. As we considered the worst-case scenario for the correct key by disregarding any dependencies, the results are always better for the correct key in practice. As we discussed in Section 4.2.1, this observation adjusts with the fact that the calculated values of state y_{r-1} for the correct key are not completely random. The higher actual value of $\texttt{cnt}(K^i, \mathcal{V}^i)$ for the correct key is helpful for the key-recovery attack. Our results confirm the high accuracy of Algorithm 4 in retrieving the correct key uniquely in practice.

It is worth noting that, with a non-optimized implementation of Algorithm 4 in the `Python3` language running on a single core Intel Core i7-9750H at 2.60GHz, the correct key can be recovered in less than a minute when $\lambda > 2$. For $\lambda = 2$, if we utilize basic data-parallel programming, we can obtain the right key in a few hours on the same machine, albeit the number of key candidates may be more than in the previous cases. To accomplish this, we simply divide the set of key candidates into several equally subsets and run Algorithm 4 on each subset in parallel, ultimately picking the key with the highest counter as the correct key.

Table 4: Expected values of counters of a wrong candidate $(K^i, \mathcal{V}^i)$ versus the correct $(K^i, \mathcal{V}^i)$, i.e. $\texttt{cnt}_{\texttt{WK}}$ and $\texttt{cnt}_{\texttt{CK}}$ respectively.

		Theory		Experiment				Theory		Experiment	
N	λ	$\texttt{cnt}_{\texttt{WK}}$	$\texttt{cnt}_{\texttt{CK}}$	$\texttt{cnt}_{\texttt{WK}}$	$\texttt{cnt}_{\texttt{CK}}$	N	λ	$\texttt{cnt}_{\texttt{WK}}$	$\texttt{cnt}_{\texttt{CK}}$	$\texttt{cnt}_{\texttt{WK}}$	$\texttt{cnt}_{\texttt{CK}}$
1554	2	7865.94	11744.44	7866.06	12316.36	1490	11	1459.29	2906.68	1459.17	3798.01
1547	3	6088.03	10064.41	6089.08	10642.27	1483	12	1281.85	2557.55	1281.45	3419.05
1540	4	4817.60	8519.74	4816.42	9253.09	1476	13	1132.86	2262.55	1133.12	3113.09
1533	5	3889.99	7192.24	3888.73	8038.08	1469	14	1006.41	2011.19	1006.53	2843.33
1526	6	3197.91	6086.93	3197.89	6983.73	1462	15	898.08	1795.32	898.37	2604.68
1519	7	2670.56	5179.21	2671.10	6093.00	1455	16	804.50	1608.58	804.87	2389.48
1511	8	2259.12	4433.64	2258.90	5346.17	1343	32	179.79	359.58	179.84	785.64
1504	9	1934.63	3825.15	1934.87	4749.19	1231	48	46.07	92.13	46.12	282.93
1497	10	1673.20	3323.45	1673.31	4232.39	1121	64	11.35	22.70	11.36	91.59

4.3 Key-recovery Attack Based on $\mathcal{V}$ and $\mathcal{V}^*$

In this part, we demonstrate how the knowledge of $\mathcal{V}^*$ can be utilized to provide a more efficient key-recovery attack to retrieve the correct key uniquely given the candidates of $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$. Let us assume that the attacker can obtain a list of candidates for $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$ by performing Algorithm 2, as it is described in Section 3.6.2. Considering the permutation layer of an SPN cipher, the output of an Sbox in the $(r-1)^{th}$ round is affected by t Sbox(es) in the r^{th} round in the backward direction, where t depends on the structure of the target block cipher and the target word $y_r[j]$. We denote the corresponding t-value for the j^{th} word by t_j. For the block ciphers like `AES` and `LED` which utilize strong permutations $t_j = 4$ for all words $0 \le j \le 16$ while $1 \le t_j \le 3$ for other block ciphers like `CRAFT` [BLMR19] and `SKINNY` [BJK+16] depending on j. In what follows we assume that $y_{R-1}[j]$ depends on $\{y_R[j_1], y_R[j_2], \cdots, y_R[j_{t_j}]\}$.

If $\{y_R[j_1], y_R[j_2], \cdots, y_R[j_{t_j}]\} \cap \mathcal{V}^{*i} = \varnothing$, the probability that the word $y_{R-1}[j]$ would not be in $\mathcal{V}^i$ depends on whether the candidate $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$ is a correct or wrong candidate as it is given in Equation (17).

$$\Pr(y_{R-1}[j] \notin \mathcal{V}^i) = \begin{cases} 1 & \text{Correct candidate } (K^i, \mathcal{V}^i, \mathcal{V}^{*i}), \\ (1 - \frac{\lambda}{2^n}) & \text{Wrong candidate } (K^i, \mathcal{V}^i, \mathcal{V}^{*i}). \end{cases} \tag{17}$$

Equation (17) emphasizes that $y_{R-1}[j] \in \mathcal{V}^i$ is an impossible event for the correct candidate. In other words, if $y_{R-1}[j] \in \mathcal{V}^i$ happens during the decryption process over a candidate $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$, then it can be interpreted that the candidate is certainly wrong. This strong distinguisher can be utilized to filter the wrong candidates of $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$, as it is described in Algorithm 5. To determine that a candidate $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$ is a correct or wrong candidate we follow this procedure: For each available ciphertext C_l, we obtain the words of the corresponding y_{R-1} as much as possible. If there is a $y_{R-1}[j] \in \mathcal{V}^i$, then we remove $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$ as it is a wrong candidate.

4.3.1 Attack Analysis

Given a ciphertext C_l, we can determine $y_{R-1}[j]$ only if $\{y_R[j_1], y_R[j_2], \cdots, y_R[j_{t_j}]\} \cap \mathcal{V}^{*i} = \varnothing$ which occurs with the following probability:

$$\Pr(\{y_R[j_1], y_R[j_2], \cdots, y_R[j_{t_j}]\} \cap \mathcal{V}^{*i} = \varnothing) = (1 - \frac{2 \cdot \lambda}{2^n})^{t_j} \tag{18}$$

Hence, we expect to be able to determine $\Psi = \sum_{j=0}^{L-0} (1 - \frac{2 \cdot \lambda}{2^n})^{t_j}$ words of y_{R-1} for a ciphertext. The probability of passing the filter (i.e. $y_{R-1}[j] \in \mathcal{V}^i$ never happened) by a wrong candidate of $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$ is $(1 - \frac{\lambda}{2^n})^{\Psi}$. Given N ciphertexts $C_1, \ldots, C_N$, the probability that a wrong candidate can pass the filter in all N experiments is $(1 - \frac{\lambda}{2^n})^{N \cdot \Psi}$ while the correct candidate $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$ always passes the filer over all experiments. When $(1 - \frac{\lambda}{2^n})^{\Psi}$ is small enough, the correct key may be obtained uniquely.

Algorithm 5 Key-recovery attack on multiple fault model, given candidates of $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$

Require: $C_1, \ldots, C_N$ and τ candidates of $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$
Ensure: Correct $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$

```
1: for $i \leftarrow 1$ to $\tau$ do
2:     for $h \leftarrow 1$ to $N$ do
3:         Determine $y_R$
4:         for $j \leftarrow 0$ to $L-1$ do
5:             if $\{y_R[j_1], y_R[j_2], \cdots, y_R[j_{t_j}]\} \cap \mathcal{V}^{*i} = \varnothing$ then
6:                 Determine $y_{R-1}[j]$
7:                 if $y_{R-1}[j] \in \mathcal{V}^i$ then
8:                     Remove $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$ from candidates and go to line 1 (next $i$)
9: return remaining candidates for $(K^i, \mathcal{V}^i, \mathcal{V}^{*i})$.
```

4.3.2 Application on LED-64 and AES-128

LED-64 consists of eight steps, each of which has four rounds. Hence, it comprises a total of 32 rounds. As the primary source of diffusion, LED-64 employs an MDS function. As a result, each word of y_{31} is a function of four words of y_{32}. In other words, $t_j = 4$ holds for all words in LED-64, i.e. $0 \leq j \leq 15$. For example, the value of $y_{31}[0]$ is solely determined by the four words $\{y_{32}[0], y_{32}[4], y_{32}[8], y_{32}[12]\}$, if we consider the backward direction and it can be determined uniquely if $\{y_{32}[0], y_{32}[4], y_{32}[8], y_{32}[12]\} \cap \mathcal{V}^{*i} = \varnothing$. Similarly, each value of $y_{31}[5]$, $y_{31}[10]$ and $y_{31}[15]$ only depends on $\{y_{32}[0], y_{32}[4], y_{32}[8], y_{32}[11]\}$. Therefore, if we can determine $y_{31}[0]$ uniquely, we can also determine the values of these three words ($y_{31}[5]$, $y_{31}[10]$, and $y_{31}[15]$) as well. A similar argument can be made for the remaining word in y_{31}. Hence, given a ciphertext C_l we can determine four words of its corresponding y_{31} if the corresponding four words from y_{32} are not in $\mathcal{V}^{*i}$, which happens with probability $(1 - \frac{2 \cdot \lambda}{2^n})^4$. Given N ciphertexts, we expect to find $\Omega = N \cdot 4 \cdot 4 \cdot (1 - \frac{\lambda}{2^n})^4$ words of y_{31}, since each ciphertext contains four such sets. Consequently, the expected wrong keys that could pass the filtering will be $|WK| = \tau \cdot (1 - \frac{\lambda}{2^4})^{\Omega}$. Table 5 represents the numerical results for $2 \leq \lambda \leq 7$. Following the provided experimental results in Section 5, $\lambda \leq 6$ are more probable for 4-bit block ciphers, e.g. `LED`, and the proposed attack filters the wrong keys for these values of λ perfectly.

The same argument can be used for `AES`, and to determine 4 words of y_9 we need 4 corresponding words from y_{10}. Table 5 represents the numerical results for $2 \leq \lambda \leq 32$ faults on `AES`. We also implemented Algorithm 5 to simulate the key recovery attack on `AES`-128. We observed that if a sufficiently large number of ciphertexts are provided, it always delivers the correct key uniquely.

Table 5: Expected values of survived wrong keys on different values of λ for `LED`-64 and `AES`-128, where $\Pr(WK) = (1 - \frac{\lambda}{2^8})^{\Omega}$.

LED-64											
τ	N	λ	Ω	$\Pr(WK)$	$\|WK\|$	τ	N	λ	Ω	$\Pr(WK)$	$\|WK\|$
2^{19}	2^8	2	1296	6.95724E-76	3.6476E-70	2^5	$2^{7.91}$	3	587.2017	1.11719E-53	3.57501E-52
2^5	$2^{7.91}$	4	240.5178	8.91265E-31	2.85205E-29	2^5	$2^{7.91}$	5	76.10134	4.1327E-13	1.32246E-11
2^5	$2^{7.91}$	6	15.03236	0.000854268	0.027336575	2^5	$2^{7.91}$	7	0.93	0.582417518	18.63736058
AES-128											
τ	N	λ	Ω	$\Pr(WK)$	$\|WK\|$	τ	N	λ	Ω	$\Pr(WK)$	$\|WK\|$
2^{23}	2^8	2	3845.938	7.93925E-14	1.33199E-06	2^8	2^8	3	3725.29	8.48368E-20	2.17182E-17
2^8	2^8	4	3607.504	2.12202E-25	5.43237E-23	2^8	2^8	5	3492.533	1.20812E-30	3.09279E-28
2^8	2^8	6	3380.332	1.52308E-35	3.8991E-33	2^8	2^8	7	3270.857	4.13866E-40	1.0595E-37
2^8	2^8	8	3164.063	2.36051E-44	6.04291E-42	2^8	2^8	9	3059.905	2.75336E-48	7.0486E-46
2^8	2^8	10	2958.34	6.40242E-52	1.63902E-49	2^8	2^8	11	2859.325	2.89452E-55	7.40996E-53
2^8	2^8	12	2762.816	2.48253E-58	6.35527E-56	2^8	2^8	13	2668.772	3.94315E-61	1.00945E-58
2^8	2^8	14	2577.149	1.13286E-63	2.90012E-61	2^8	2^8	15	2487.905	5.75244E-66	1.47262E-63
2^8	2^8	16	2401	5.04702E-68	1.29204E-65	2^8	2^8	32	1296	6.95724E-76	1.78105E-73

5 Experimental Results

In this section, we report results from our practical fault injection experiments performed on `AES` and `LED` block-ciphers. The main aim of our experiments is to identify the types of persistent faults on the Sbox that are achievable.

5.1 Target Platform

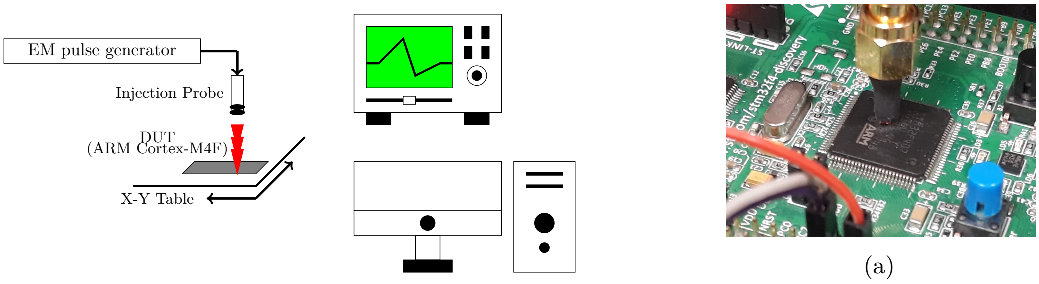
Our DUT is the STM32F407VG microcontroller based on the 32-bit ARM Cortex-M4 processor housed on the STM32F4DISCOVERY evaluation board. The core and peripherals of the DUT are clocked at the maximum possible clock frequency of 168 MHz. We compiled our implementations using the `arm-none-eabi-gcc` compiler with the highest compiler optimization level-`O3`. We utilize the ST-LINK/v2.1 add-on board for UART communication with our DUT and OpenOCD framework for flash configuration and on-chip hardware debugging with the aid of the GNU debugger for ARM (`arm-none-eabi-gdb`). For AES-128, we used a simple round based implementation written in the `C` language with a focus on Sbox transfer from flash to RAM on boot-up[2]. For LED, the experiments were done on a publicly available implementation[3].

[2] Adapted from https://github.com/kokke/tiny-AES-c/blob/master/aes.c

[3] https://github.com/vedadux/configurable-LED

5.2 Experimental Setup for Fault Injection

We utilize Electromagnetic Fault Injection (EMFI) to inject persistent faults into our DUT. Our choice of EMFI is motivated by several reasons. Firstly, the fault injection can be done in a completely non-invasive manner. Moreover, it can be used to inject faults from the front-side of the chip, and thus requires very minimal and in some cases, no preparation of the DUT for fault injection. Our EMFI setup consists of a pulse generator that can generate high voltage pulses (up to 200V) with very low rise times (<4ns). A controller software on the laptop synchronizes the operation of the EM pulse generator and DUT through serial communication. The pulse generator is directly triggered by an external trigger signal from the DUT, which synchronizes the voltage pulse with the DUT's operation. The EM pulse injector is a customized hand-made EM probe designed as a simple loop antenna. The location of the EM pulse injector on the chip is controlled by an XYZ motorized table. Our setup also contains an additional relay switch to perform an automated power-on reset of the device, used during validation of the persistent faults. Refer to Figure 3 for the EM probe used in our experiments.



(a)

Figure 3: Electromagnetic Fault Injection on the STM32F407VG microcontroller based on 32-bit ARM Cortex-M4 core (a) Experimental Setup, (b) and its placement over the DUT

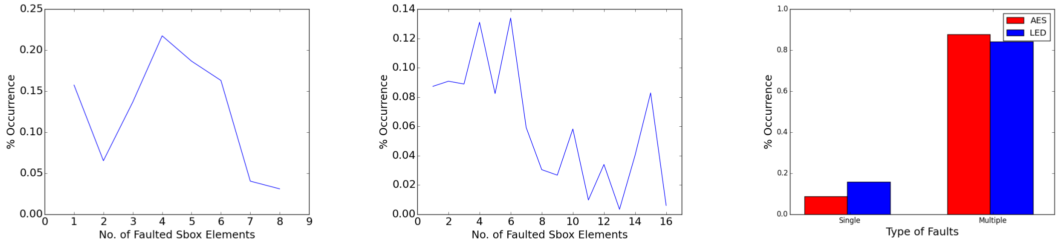
5.3 Persistent Faults on Sbox through EMFI

We consider the scenario of cryptographic software running on such embedded microcontrollers wherein the Sbox is typically present as part of the code stored in flash memory. Upon boot up (or an encryption call), the Sbox is retrieved from flash and is stored in a designated location in the main memory (RAM). Subsequently, the encryption procedure utilizes the Sbox stored in RAM to compute the ciphertext. This approach is desirable to decrease Sbox access times, especially in devices such as constrained microcontrollers with no cache memory. Thus, if an attacker is able to fault the movement of Sbox from flash to RAM, then it leads to a persistent fault in the Sbox. A similar fault model has been reported by Menu et al. [MBD^{+}19]. In our experiments, we consider two types of Sboxes: (1) 4-bit Sbox (LED-64) and (2) 8-bit Sbox (AES-128). The Sbox values are loaded from the flash memory into the registers, in an iterative manner, using the 32-bit LDR.W load instruction and subsequently, they are stored from the registers into the RAM using the 32-bit STR.W store instruction. For our practical experiments, we fix the width of the pulse to 7 ns (nanoseconds) and the voltage to about 190 v, as we are able to observe reliable faults with these fault injection parameters. We then perform a thorough fault injection campaign over the entire area and a full sweep of the injection delay to identify the different types of faults that can be observed on the Sbox values.

5.4 Fault Injection Results

While our aim is to study the different types of faults achievable on the Sbox values, our main focus is on the number of Sbox values that can be corrupted and not the value that the faulted Sbox is corrupted to, as it is not relevant for our analysis. In the case of the 4-bit Sbox of LED-64, the attacker is much more likely to fault 3-5 elements in the Sbox, compared to faulting a single entry (Figure 4(a)). In the case of the 8-bit Sbox of AES-128, the chance to fault 4 or 6 elements of the Sbox is the highest (Figure 4(b)). Refer to Figure 4(c) for the bar plots demonstrating the frequency of faults affecting single and multiple Sbox elements. We found that in total, 18,865 faults successfully targeted the AES S-box, out of which over 87% were affecting multiple elements, and single element faults were less than 10%. A few faults affect more than half of the elements of the Sbox, which violates our fault model and are considered out of scope (see Section 3.1). We observed about 3% and 0.1% of faults for AES and LED Sbox of such types. While single faults are achievable with high repeatability, it requires detailed profiling of the chip surface to identify exact coordinates and EM pulse parameters. An attacker who does not spend much effort in the profiling phase is more likely to get multiple faults. We also verified the proposed key-recovery attacks by incorporating the faulty Sboxes with 2, 4, 6 injected faults driven from experiments. In all cases, both Algorithm 4 and Algorithm 5 returned the correct key efficiently. We also made two interesting observations: 1) $\mathcal{V}^* < \mathcal{V}$, which shows a bias in the injected fault and 2) for the driven Sbox with 8 faults, we observed an ineffective fault value. Hence, the exact λ was 7 not 8. Interestingly, Algorithm 3 returned the correct λ which was 7. The details of Sboxes and key-recoveries are available under the following address: https://github.com/hadipourh/faultyaes

Further, the presented attacks can be extended to other implementation choices. A common performance-oriented implementation choice for AES is the use of T-tables instead of Sbox. T-tables merge SubBytes, ShiftRows and MixColumns into 4 8 × 32 t-tables. As the last round of AES does not execute MixColumns, often Sbox is used in the last round, thus keeping our analysis technique unchanged. Even if a modified t-table is used for the last round, the proposed analysis can be trivially adapted, as already shown in [ZLZ+18]. Similarly, masked implementation based on look-up tables is also vulnerable, as previously demonstrated in [PZRB19].



(a) LED-64 (b) AES-128 (c) Single vs Multiple

Figure 4: Distribution of the no. of faulted entries of the Sbox for (a) LED-64, (b) AES-128 and (c) Single vs Multiple Faults

6 Conclusion

While the feasibility of persistent fault analysis with a single-fault injected is demonstrated in the literature, there are some challenges in extending the known techniques to the multiple faults setting. In this paper, we provided new insight into PFA by proposing novel methods for extending its application to the multiple faults setting that can be performed in practice. We provided parametric frameworks that can be easily adjusted to different scenarios. This paper can be considered as a significant step in performing PFA under multiple faults in more realistic scenarios.

Acknowledgments

The authors wish to thank Florian Mendel and the anonymous reviewers for their detailed comments and helpful suggestions.

References

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[BJK$^+$16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[BLMR19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Trans. Symmetric Cryptol.*, 2019(1):5–45, 2019.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[CB19] Andrea Caforio and Subhadeep Banik. A study of persistent fault analysis. In Shivam Bhasin, Avi Mendelson, and Mridul Nandi, editors, *Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings*, volume 11947 of *Lecture Notes in Computer Science*, pages 13–33. Springer, 2019.

[CGR20] Sébastien Carré, Sylvain Guilley, and Olivier Rioul. Persistent fault analysis with few encryptions. In Guido Marco Bertoni and Francesco Regazzoni, editors, *Constructive Side-Channel Analysis and Secure Design - 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1-3, 2020, Revised Selected Papers*, volume 12244 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2020.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.

[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.

[DR99] Joan Daemen and Vincent Rijmen. The rijndael block cipher: Aes proposal. In *First candidate conference (AeS1)*, pages 343–348, 1999.

[ESP20] Susanne Engels, Falk Schellenberg, and Christof Paar. SPFA: SFA on multiple persistent faults. In *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*, pages 49–56. IEEE, 2020.

[FJLT13] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 108–118. IEEE Computer Society, 2013.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The led block cipher. In *International workshop on cryptographic hardware and embedded systems*, pages 326–341. Springer, 2011.

[GPT19] Michael Gruber, Matthias Probst, and Michael Tempelmeier. Persistent fault analysis of ocb, DEOXYS and COLM. In *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2019, Atlanta, GA, USA, August 24, 2019*, pages 17–24. IEEE, 2019.

[GYTS14] Nahid Farhady Ghalaty, Bilgiday Yuce, Mostafa Taha, and Patrick Schaumont. Differential fault intensity analysis. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 49–58. IEEE, 2014.

[LSG$^+$10] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault sensitivity analysis. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2010.

[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.

[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 Jürgen Teich and Franco Fummi, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, pages 1–6. IEEE, 2019.

[SHP09] Jörn-Marc Schmidt, Michael Hutter, and Thomas Plos. Optical fault attacks on AES: A threat in violet. In Luca Breveglieri, Israel Koren, David Naccache, Elisabeth Oswald, and Jean-Pierre Seifert, editors, *Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009*, pages 13–22. IEEE Computer Society, 2009.

[SLFP04] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A collision-attack on AES: combining side channel- and differential-attack. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 163–175. Springer, 2004.

[SWP03] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A new class of collision attacks and its application to DES. In Thomas Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2003.

[XZY+21] Guorui Xu, Fan Zhang, Bolin Yang, Xinjie Zhao, Wei He, and Kui Ren. Pushing the limit of PFA: enhanced persistent fault analysis on block ciphers. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 40(6):1102–1116, 2021.

[YJ00] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on computers*, 49(9):967–970, 2000.

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

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