Linked Fault Analysis

Ali Asghar Beigizad, Hadi Soleimany, Sara Zarei and Hamed Ramzanipour

***Abstract*— Numerous fault models with distinct characteristics and effects have been developed. The costs, repeatability, and practicability of these models should be assessed. Moreover, there must be effective ways to use the injected fault to retrieve the secret key, particularly if the implementation includes any countermeasures. In this paper, we introduce a new fault analysis called "linked fault analysis" (LFA), a more powerful technique than other well-known fault attacks against implementations of symmetric primitives, especially in software implementations. For known fault analysis, the basis for the fault model is either the bias over the faulty value or the relationship between the correct value and the faulty one. In the LFA, however, a single fault involves two intermediate values. The faulty target variable, u', is linked to a second variable, v, such that a particular relation holds: $\mathsf{u}' = \mathsf{l}(\mathsf{v})$. LFA lets the attacker perform fault attacks without the input control, using far fewer data than previously introduced fault attacks in the same class. We show the utilization of LFA in the presence or absence of typical redundancy-based countermeasures by introducing "Linked Differential Fault Analysis' ' (LDFA) and "Linked Ineffective Fault Analysis' ' (LIFA). We also demonstrate that, under specific circumstances, LFA is still effective even when masking protections are in place. We have performed our attacks against the public implementation of AES and PRESENT in ATMEGA328p to show the feasibility of LFA in the real world. The practical results and simulations validate our theoretical models as well.**

***Index Terms*— Fault Analysis, Linked Fault**

I. Introduction

Fault attacks are a type of physical attack in which the attacker purposefully induces a fault in the target device and monitors the target's reaction to that fault to learn some information on the secret key. Based on how long they last, induced faults can be grouped into three categories. The majority of the suggested fault attacks are based on transient faults, which affect the system only temporarily. "Permanent faults" are the second category. They have a long-lasting negative impact on the target device. Persistent faults are the third form of fault. They persist but can be removed by resetting the target device. In this paper, we will concentrate on transient faults.

Research on transient fault attacks against symmetric primitives has largely focused on how well they can convert an intermediate value into an faulty one. The attacker injects a fault into value u to transform it into the faulty value u'. These two values are typically expected to have some extractable correlations. One can mention fault models like stuck-at, bit-flips, random-AND, biased, random fault, etc., where the correlation is distinct in each case. In this work, we take a previously unseen perspective and develop a novel fault-based attack called "linked fault analysis" (LFA) based on an out-of-sight type of relation. While in traditional fault attacks, the fault model is defined based on the relation between the correct and the faulty values, in the LFA model, the fault involves more than one intermediate value, the target variable u, and a second variable v. The relationship brought on by the fault effect is the transformation of the u value to the faulty value u' according to the v value. We entitle this event as a linked change and utilize it in our new analysis.

A.A. Beigizad, H. Soleimany, and S. Zarei are with the Cyber Research Center, Shahid Beheshti University, Tehran, Iran, e-mail: beigizad@yahoo.com, h_soleimany@sbu.ac.ir, and sarazareei.94@gmail.com

H. Ramzanipour is with the Electrical Engineering Dept. of Shahid Rajaee Teacher Training University (SRTTU), Tehran, Iran, e-mail: h.ramzanipour@sru.ac.ir

A. Overview of Prior Transient Fault Attacks

Due to numerous factors playing a role in the settings of fault attacks, and also different demands and assumptions, it is challenging to declare a comprehensive classification for fault attacks. In what follows, we classify well-known existing fault attacks based on two critical benchmarks. Along with the classification, we will take a quick look on three additional most notable characteristics of attacks that fall under each group. It would help to pinpoint the precise location of LFA and its properties.

Control vs. no-control over the inputs. From this point of view, attacks can be classified based on whether or not the attacker can repeatedly encrypt a fixed plaintext. Sometimes this fixed input is presumed to be chosen by the attacker, and sometimes this is not a requirement. The underlying presumption in each situation is that the attacker has access to the input. A second class of attacks, on the other hand, does not necessitate input control on the part of the attacker.

Bypassing vs. losing to redundancy-based countermeasures. In the presence of typical redundancy-based countermeasures (infection-based [1]–[3] or detection-based [4]), the adversary is unable to detect defective ciphertexts. Hence, it would not be possible for him to execute attacks that rely on the faulty ciphertext. However, some faults do not alter any intermediate value and are therefore invisible to embedded countermeasure systems. This phenomenon is known as an "ineffective fault." There are attacks that are respectively able to circumvent redundancy-based countermeasures because they exclusively employ ineffective faults.

Table I represents the four subcategories created by the two abovementioned criteria, along with a list of well-

TABLE I: Classifications of prior transient fault attacks on block ciphers.

Group Tag	Attack Name	Input's Control	Redundancy-based Countermeasures Bypassing	Is Applied to Masked Implementations	Sensitivity to Noise: Missed Fault - Shuffling	Accurate Equipment
(G1)	DFA	Need	✗	✗	Low-Medium [†]	✗
	IDFA			✗	Low-Medium [†]	✗
	DFIA			✗	Low-Medium [†]	✗
	CFA			✗	Low-Medium [†]	✗
(G2)	FTA	Need	✓	✓	Unexplored [¶]	✓
	FSA			✗	Unexplored [¶]	✗
	SEFA			✓	Low-High	✗
(G3)	SFA	Do Not Need	✗	✗	High-High	✗
	LDFA[*]			✗	Low-Medium [†]	✗
(G4)	SEA	Do Not Need	✓	✗	Unexplored [¶]	✗
	IFA			✗	Low-Medium [†]	✓
	SIFA			✓	High-High	✗
	LIFA[**]			✓	Low-Medium [†]	✗

[*],[**] Linked-DFA and Linked-IFA, proposed in sections III-B, and III-C of this research.
[†] Due to using deterministic relations.
[¶] To the best of our knowledge.

known attacks falling within each group. The first group (G1) contains Differential Fault Analysis (DFA) [5], one of the most prominent fault attacks, able to defeat many implementations requiring few computations. However, this power comes at the cost of the adversary's access to both correct and faulty ciphertexts corresponding to a given plaintext. DFA has been widely applied to numerous unprotected implementations of block ciphers [6]–[9], authenticated encryption schemes [10], and even stream ciphers [11]. Extensive research has been conducted on DFA, and multiple generalized and automated frameworks have been developed to identify vulnerabilities against this attack methodology [12]–[14]. The Impossible Differential Fault Attack (IDFA) is a differential-based fault attack based on an impossible characteristic [15]. Algebraic Fault Attack (AFA) is similar to DFA, since it requires both faulty and correct outputs [16]–[18]. However, AFA constructs algebraic equations and focuses on the solution of a system of equations instead of using differential characteristics. Differential Fault Intensity Analysis (DFIA) [19] is another method in which the key-recovery procedure relies on the repetition of encryption for a fixed plaintext and the knowledge of the value of faulty ciphertexts. Collision Fault Analysis (CFA) [20] is another fault attack belonging to the first group. It applies when an attacker encrypts two related inputs, injects a fault into one of the computations, and then exploits a collision that may occur (usually in the outputs). Redundancy-based countermeasures can prevent attacks of group (G1), since the attacker needs to control the input.

The second group (G2) contains attacks that require input control but can bypass redundancy-based countermeasures. Fault Sensitivity Analysis (FSA) [21] needs to encrypt the same plaintext during the profiling phase to determine the critical fault injection intensity for various plaintexts. The adversary should be able to determine whether the injected fault was effective. Therefore, FSA does not necessarily require faulty ciphertexts. On the other hand, Fault Template Attack (FTA) [22] is another technique that can circumvent redundancy-base countermeasures since it does not rely on faulty ciphertexts. However, an attacker should be able to encrypt a fixed (but unknown) plaintext several times. Therefore, both FSA and FTA can circumvent redundancy-based countermeasures, but in practice, the attacker must control the input to repeat the encryption process of the same input. The last member of this group is the newly introduced Statistical Effective Fault Analysis (SEFA) [23]. A dual for the most famous SIFA attack.

The last two groups ((G3) and (G4)) comprise attacks not requiring input control. Statistical Fault Analysis (SFA) [24] eliminates the requirement of repeated encryptions of a particular plaintext by means of employing statistics. However, it demands faulty ciphertexts and therefore can be thwarted by redundancy-based countermeasures. The fourth group (G4) includes attacks that can get over redundancy-based countermeasures while not requiring input control. These attacks exclusively target cases in which the fault does not affect the computation. Safe Error Attack (SEA) [25] is typically relevant to public-key cryptographic systems. Ineffective Fault Analysis (IFA) [26] and its statistical counterpart (SIFA) [27] are included in this group as well. IFA typically uses models such as stuck-at faults, which are difficult to achieve in practice and require the use of costly tools such as laser [28], [29].

As mentioned at the beginning of this section, we have also presented three other crucial fault attack characteristics in Table I: applicability when masking is present, noise sensitivity, and accuracy required for fault injection (the last three columns). Although it is not straightforward enough to measure or figure out these characteristics accurately in all listed attacks, we provided them with our best try just to give a helpful overlook. Applicability in the presence of masking is supplied in accordance with the published practical results in the community. It is worth noting given that masking is a crucial protection strategy. The perfect fault injection, on the other hand, is a common assumption in the literature, however it is rarely practical in real-world settings. So, we have set aside the column *Sensitivity to Noise* to look at this property under the many presented attacks. Unwanted events occur as a result of either missed faults or the noise created by hiding tactics like shuffling. Naturally, as noise levels rise, the data complexity of fault attacks rises as well, although noise's effects on attacks can vary. Given that missed and ineffective faults cannot be distinguished, even strong attacks like SIFA are unavoidably significantly influenced by high rated missed faults. Shuffling is a well-known effective countermeasure against side-channel and fault assaults. Therefore, shuffling also can increase data complexity. It has less impact on fault attacks that exploit a relation with a high likelihood between faulty and correct values compared to statistical attacks.

The third mentioned property is the accuracy needed in the equipment and setup of fault injection. In some attacks, the attacker should have control over the intensity, position, and duration of the fault injection, however, in others, just the distribution disturbance is important (the precise impact of the fault is presumed to be unknown). Attacks, such as FTA or IFA, in which the attacker will inject stuck-at or bit-flip faults, are really more difficult to accomplish in practice. On the other side, the cost of the necessary equipment is closely tied to the accuracy required for fault injection; more accuracy will demand more expensive equipment.

B. Our Contributions

In certain applications, it is not possible to encrypt (or sign) the same message more than once. A prominent example is in protocols, in which the message is padded with a random value. Consequently, fault attacks that require input control (whether they require chosen plaintexts or repeated computation with an unknown input) are inapplicable in such applications. This covers all of the attacks indicated in Table I's first two groups ((G1) and (G2)). Consider, as another example, the devices that use block ciphers in operation modes such as Cipher Feedback (CFB), Output Feedback (OFB), and Counter Mode (CTR). Even though it is sometimes assumed that the underlying block cipher is utilized in ECB mode, alternative modes like CTR, OFB, CBC, and CFB may also be applied in real-world applications, where the input is fundamentally not repeated. The modes construct the following block by encrypting successive values which depend on a value "counter" or "initial value (IV)" that will not repeat over an extended period. Attacks that require input control are challenging to carry out in these applications too because it is not always possible for an attacker to reset the counter [30].

On the other hand, techniques that do not necessitate input control usually rely on specific statistical properties (groups (G3), and (G4) in Table I). As a result, the key-recovery technique usually needs more samples than attacks with input control in order to statistically distinguish between the correct key and incorrect ones. On the other side of the spectrum is DFA, which requires input control but consumes a lot less data than SFA. In certain real-world settings, both small amounts of data and control-free scenarios might be crucial; however, none of them are free in the DFA, SFA, and their other family members. In this research, we propose a new fault technique that, compared to previously presented fault attacks in groups (G3) and (G4), greatly reduces the amount of data required to perform fault attacks without the need for input control. In our attack model, we assume that an intermediate value will become faulty and linked to another intermediate value due to the instruction skipping, and we are particularly focusing on software-based implementations on microcontrollers. Our approach can work both in the presence or absence of countermeasures.

1) In the Absence of redundancy-based Countermeasures: For DFA and CFA to work, there must exist associations between two intermediate values, one from the correct computation and the other from the faulty one. Despite the limited data available, this attribute enables the attacker to utilize associations and carry out the attack. We employ a similar strategy to that of DFA and CFA in our first suggested method, with a simple but tactical difference. Our approach is based on correlations between the intermediate values of only faulty computations. In other words, we employ an intrinsic attribute of the faulty system and do not need access to the correct computation. Consequently, our proposed method can be viewed within the framework carried out with a relatively limited number of inputs based on a deterministic relation (similar to DFA and CFA), and at the same time, it avoids the need for repeated encryption (similar to SFA). We choose the term Linked-DFA, or abbreviately LDFA for this method.

2) In the Presence of redundancy-based Countermeasures: Similar to other attacks that do not require input control and are applicable in the face of countermeasures (such as SIFA), the key to encountering embedded countermeasures is using ineffective ciphertexts. These ciphertexts can circumvent redundancy-based defenses. We demonstrate that our method applies to ineffective ciphertexts too, and it also gives the attacker the ability to find missed faults under certain conditions. Missed faults occur when the fault is injected, but the desired change is not achieved. These are important phenomena in real-world applications since they can influence the performance of a powerful attack like SIFA (more explanations are given in section III-E.1). We also demonstrate that LFA might be applicable in the presence of first-order masking and detection-based countermeasures. We choose the term Linked-IFA, or abbreviately LIFA for our second proposed method.

II. Preliminaries

A. Notations

While most of the fault attacks are not specific to any particular cipher, in the interest of clarity, we will describe the attacks with an illustration using an R-round word-oriented Substitution-Permutation Network (SPN) cipher. The majority of block ciphers, such as AES, have an SPN structure. Let us assume a standard SPN design $E_K(P)$ which accepts a b-bit plaintext P and a k-bit key to produce the ciphertext C. The process of encrypting the plaintext using the key can be viewed as an iteration of an invertible function that is referred to as round function $F_{sk_r}()$, where sk_r stands for the r-th round key for $1 \leq r \leq R$. As implied by the name of the structure, each round of the cipher employs a substitution $\mathcal{NL}$ (non-linear layer) and a permutation $\mathcal{L}$ (invertible linear layer), followed by the addition of a round key. Typically, the substitution layer is constructed by combining some relatively basic nonlinear bijective functions known as Sboxes. We denote the Sbox of the cipher by S. The b-bit state is formed of L words of the same length ($m = b/L$). Let $x_r[i]$ and $y_r[i]$ represent, respectively, the $i-$th word input into the substitution layer in the $r-$th round and the $i-$th word output from it where $0 \leq j \leq L-1$ and $1 \leq r \leq R$.

B. Target Ciphers

In our experiments, we concentrate on AES [31] and PRESENT [32], even though our methods apply to any SPN block cipher. AES is the most frequently utilized cipher in real-world applications. PRESENT is an ISO/IEC standard lightweight block cipher. In what follows, we briefly describe our target ciphers, and refer to the original proposals for more information [31], [32].

AES is a 128-bit SPN-structured block cipher with a key size of 128, 192, or 256 bits. It operates on an array of eight bytes ($m = 8$). In our experiments, we considered AES with 128-bit keys, denoted by AES-128, which contains $R = 10$ rounds, each of which involves four transformations: SubByte

(SB), ShiftRows (SR), MixColumns (MC), and AddRoundKey (AK). Exceptionally, the last round lacks MixColumns.

PRESENT is a 64-bit block cipher and supports a key length of 80 or 128 bits. Each of PRESENT's 31 rounds contains the application of a 4-bit Sbox to 16 nibbles, a simple bit-wise permutation, and a key addition.

C. DFA

The most widely used technique for fault analysis is differential fault analysis, which was first introduced in [5]. The assumption is that the attacker can inject a difference at a certain intermediate value in a particular round. Typically, the fault is injected into one of the state's words preceding the nonlinear layer. In the basic key-recovery technique, the attacker guesses the subkey bits involved in the last rounds of the cipher and partially decrypts both faulty and correct ciphertexts to compute the desired intermediate value. The intended difference is expected to be seen more frequently for the correct key than for the incorrect ones.

D. SFA, SIFA, and SEFA

The Statistical Fault Attack (SFA) employs a biased distribution across an intermediate value. The attacker decrypts faulty ciphertexts with the key candidates over the final round(s) of the cipher and computes a statistical scoring function for each key candidate to assess how closely the computed distribution resembles the predicted distribution with the correct key. SIFA, enabled by SFA and Ineffective Fault Attack (IFA) methods, utilizes the biased distribution of the intermediate value over ineffective faults and bypasses redundancy-based countermeasures since it only requires ineffective outputs. The younger twin of SIFA is SEFA, which employs the same statistical strategies as SIFA but focuses on making use of effective faults.

Different statistical tests can be applied given N samples. The attacker uses statistical test SEI to rank the key candidates, if he does not know the details of the faulty distribution.

$$\mathrm{SEI}(k) = \sum_{x\in\mathcal{X}} (\hat{p}_k(x) - \theta(x))^2 \,. \tag{1}$$

where θ denotes a uniform distribution and $\hat{p}$ is the probability distribution in the intermediate value for the key candidate. The number of required ciphertexts in SEI test to obtain the correct key can be estimated based on Equation (2) [27].

$$N_{\mathrm{SEI}} \approx \frac{\beta \cdot \Phi_{0,1}^{-1}(\alpha)}{C(p,\theta)} \tag{2}$$

where p is the true statistical distribution of intermediate value, and $C(p,\theta)$ denotes the capacity and is given in Equation (3) in case that p is close to θ.

$$C(p,\theta) = \sum_{x\in\mathcal{X}} \frac{(p(x) - \theta(x))^2}{\theta(x)} \,. \tag{3}$$

The probability of an ineffective and effective event is referred to as the ineffectivity rate Π_{i} and effectivity rate Π_{e}, respectively. Hence, the total number of required ciphertexts for SIFA and SEFA is $N = \frac{N_{\mathrm{SEI}}}{\Pi_{\mathsf{i}}}$ and $N = \frac{N_{\mathrm{SEI}}}{\Pi_{\mathsf{e}}}$, respectively.

SFA and SIFA were initially applied to the 8th and 9th rounds of AES, respectively. Because the SEI between a uniform distribution and the probability distribution $\hat{p}$ of the target intermediate value is the same for all key candidates, the original description of SFA and SIFA did not apply to the last round of AES. [33] has shown that statistical fault attacks can be deployed effectively on the last round Sbox of AES by taking into account the distribution of Hamming weight of the targeted variable. Section V-D will compare LFA to earlier fault attacks. We will evaluate the statistical fault attacks presented in [33] to have a fair comparison because we applied LFA on the last round of AES.

E. Limitations of Instruction Skip Attacks in Block Ciphers

A very efficient method of altering processed values is to skip the execution of one or more multiprocessor instructions. A variety of laser [4], [34], or electromagnetic pulses [35]–[37], clock [38]–[42], and power glitches [43] have been suggested for skipping single or multiple instructions on microcontrollers. Most instruction-skipping fault attacks on symmetric-key primitives require input control and cannot naturally get beyond typical redundancy-based countermeasures.

One possible attack is bypassing the final round's key addition [38], [44]. The last round subkey can be obtained by computing the exclusive-or of the faulty ciphertext with the correct ciphertext. Skipping addition is a well-known attack technique that has been employed against stream ciphers [45]. Pessl and Prokop recently demonstrated at CHES 2021 that addition skip can be used to attack lattice-based KEMs [46]. Despite the public key schemes [43], [47], this approach is not typically relevant to symmetric primitives in the presence of typical redundancy-based countermeasures. Given that symmetric primitives are not based on algebraic structures, skipping an operation typically results in an active fault that can be identified by redundancy.

In another method, skipping the conditional branching following the redundant computation [40], would detour the equality check. Albeit, such faults that target the equality check in detection-based countermeasures or reduce the number of iterations in a loop are apparent targets and the community is already aware of the necessity to protect these obvious targets with additional protections.

III. Linked Fault Analysis (LFA)

A. Fault Model

As stated before, we target software-based implementations on microcontrollers, and our analysis is based on instruction skipping. In our attack model, we have three assumptions:

1) The attacker can control the intensity and duration of the fault. Given that faults are physical disturbances, the more intensely they are introduced, the more severely the device being tested is impressed, resulting in the occurrence of one or more faults (depending on the intensity). Therefore, there is always a minimum

intensity threshold. On the other hand, if the intensity is increased significantly from a maximum value, the device's response might stop changing considerably or it might change in an unfavorable way for that attack model. Thus, the ideal interval should be discovered through trial and error. The duration (frequency) parameter is almost the same. The attacker should identify the optimal frequency range for the system. Fortunately, in our case (model), this interval isn't too wide to require a lot of experimentation. The vicinity is known (for instance, the Sbox of the last round). Later, in section V-D, where we report our experimental results, we will see that other attacks like SIFA and SFA also require this trial and error to determine their appropriate interval.

2) Similar to the majority of proposed fault attacks, the attacker is aware of the timing (fault location). Although later in Section III-D, we show that the assumption of precise knowledge about the timing can be relaxed under some circumstances.
3) The target is a word-oriented block cipher, and its nonlinear layer is implemented using one or more pre-computed look-up tables. We assume two variables u and v are processed sequentially to load the output of such look-up tables.

We use instruction skipping to cause a fault on an intermediate value u so that the faulted value u' is linked to another intermediate value v, without the attacker knowing what v, u, and u' are. Under certain conditions that v and u are processed sequentially, the value of faulty u (i.e., u') becomes equivalent to the v value (i.e., the created link is equality[1]). The created links across intermediate values may result in linked words in the ciphertexts, which an attacker can exploit to obtain information about the secret key. In the rest of this section, we describe the mechanism of injecting the fault using cheap instruments, well-timed power spikes, or clock glitches.

Each Sbox pre-computed table is called several times throughout round function execution to process all input values of the nonlinear part of the algorithm. The invocations result in loading the corresponding Sbox outputs to the RAM or flash memory. The attacker targets two of these load instructions with instruction skipping. Either skipping the instruction that selects the address of the data or the instruction that settles the values of these data into RAM or flash memory. Finding the exact location of the second load instruction after hitting the first one can be achieved by trial and error, and the attacker's knowledge about the implementation's timing.

Even though causing the instruction skipping fault and achieving our desired effect by it was a roughly challengeless task in our experimental setup, it may be difficult in some complex microcontrollers, and we do not assert that it is always achievable for all cases. Maybe more sophisticated methods can help or not [48]. Moreover, the underlying idea of our model, which is to link two variables, may also be satisfied via any other method instead of instruction skipping. We have chosen the instruction skip since it is more well-known, and we make no claim (rejection) about other methods that would (would not) exist for fulfilling this necessity. Anyway, we leave the investigation of these possibilities to future work. In this work, we simply state that our fault model and the general strategy of our proposed attack would be effective if the instruction skip and the described effect are achievable.

And last but not least, firstly, we assume that the instruction skip occurs perfectly. We will discuss the effects of missed or unwanted faults later in the last subsection.

[1]We do not assert that the equality relation is the only useful or conceivable link that can be made. We observed equality and found our model on its basis. However, there may be other relations useful for extending the model. We leave the investigation for future work.

B. Linked Differential Fault Attack (LDFA)

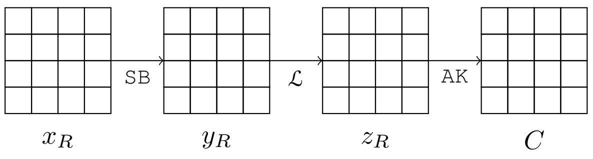
In the absence of redundancy-based countermeasures, attacks like DFA and CFA are applicable. A common property of DFA and CFA is utilizing the existence of two intermediate values that are either identical or related in the correct and faulty computations. This property permits the attacker to utilize deterministic relations and perform the attack with an extremely small amount of data. In our first proposed method, we apply a similar idea, our method is based on relations between the intermediate values of the faulty computation itself. Hence, it does not require input control. For the sake of clarity, we will first describe our attack against the last round of an arbitrary SPN cipher, and then present a general method.

1) Application to the Last Round of an Arbitrary SPN: First, we recall a well-known technique for analyzing the final round of an arbitrary SPN block cipher. Since both the key addition layer and the permutation layer $\mathcal{L}$ are linear, these operations in the final round can be interchanged by altering the subkey from sk_R to $esk_R = \mathcal{L}^{-1}(sk_R)$, which is called "*equivalent subkey*".

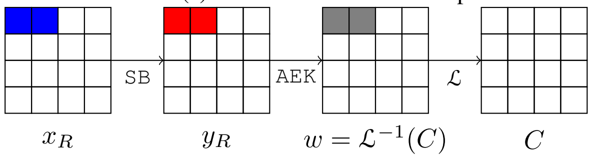
$$\mathcal{L}(y_R) \oplus sk_R = \mathcal{L}(y_R) \oplus \mathcal{L}(\mathcal{L}^{-1}(sk_R)) = \mathcal{L}(y_R \oplus \mathcal{L}^{-1}(sk_R)) = \mathcal{L}(y_R \oplus esk_R) \tag{4}$$

Based on Equation (4), an equivalent representation of the final round in SPN can be found as illustrated in Figure 1. For a simpler analysis, the adversary can eliminate the permutation layer $\mathcal{L}$ in the final round by obtaining the value of $w_R = \mathcal{L}^{-1}(C)$, which can be easily derived from the ciphertext C. If the attacker retrieves a certain m-bit word from the equivalent subkey esk_R, he can get m-bit information about the subkey sk_R, since the relation between sk_R and esk_R is bijective (i.e., $esk_R = \mathcal{L}^{-1}(sk_R)$). To illustrate that LFA applies to any word-oriented SPN block cipher, we will now examine the equivalent representation of the last round in an SPN cipher, which is illustrated in Figure 1b.

Assume that the word $x_R[j]$ becomes equal to $x_R[i]$ (demonstrated by blue) or $y_R[j]$ becomes equal to $S(x_R[i])$ (demonstrated by red) when an instruction skip occurs during the execution of Sboxes. The corresponding words in the state w_R, i.e. $w_R[i]$ and $w_R[j]$ are not necessarily equal after adding the equivalent subkey sk_R, but they can be used to retrieve information about the equivalent subkey (these words are illustrated in grey in Figure 1b).



(a) Last round in an SPN cipher



(b) Equivalent representation of last round in SPN (AEK stands for addition of equivalent subkey)

Fig. 1: Different representation of the last round in an arbitrary SPN block cipher

As shown in Equation (5), it is simple to verify that the words $w'_R[i]$ and $w'_R[j]$ that can be computed from the faulty ciphertext ($w' = \mathcal{L}^{-1}(C')$) are linked based on the value $\Delta esk_R[i,j] \triangleq esk_R[i] \oplus esk_R[j]$.

$$\begin{aligned} w'_R[i] \oplus w'_R[j] = (y'_R[i] \oplus esk_R[i]) \oplus (y'_R[j] \oplus esk_R[j]) \\ = \Delta esk_R[i,j] \end{aligned} \quad (5)$$

According to Equation (5), the relationship between the linked bytes in $w = \mathcal{L}^{-1}(C')$ is deterministic and depends on the value of the equivalent subkey.

$$\Pr(w'_R[i] \oplus w'_R[j] = \alpha) = \begin{cases} 1 & \alpha = \Delta esk_R[i,j], \\ 0 & \alpha \neq \Delta esk_R[i,j]. \end{cases} \quad (6)$$

As shown in Algorithm 1, Equation (6) shows how an attacker can get one word of information about the last equivalent subkey from just one faulty ciphertext without notable cost.

Algorithm 1 LDFA on the last round of an arbitrary SPN cipher

```
Require: One Faulty ciphertext $C'_1$ caused by a linked fault
    in $y_R[i]$ and $y_R[j]$        ▷ In case of LIFA, ineffective
    ciphertext is needed.
Ensure: Correct value of $\Delta esk_R[i,j]$
 1: $\Delta esk_R[i,j] \leftarrow \mathcal{L}^{-1}(C'_1)[i] \oplus \mathcal{L}^{-1}(C'_1)[j]$
 2: return $\Delta esk_R[i,j]$
```

Full key recovery: It should be noted that this attack is repeatable for different words. Assume, without loss of generality, that the words in the Sbox layer are executed from $x[0]$ to $x[L-1]$ in incremental order. In this example, a linked fault can occur between $x[i]$ and $x[i+1]$ for any value of i between 0 and $L-2$. So totally $(L-1)$ different linked faults can be induced. Since each linked fault can expose m-bits of information about the equivalent subkey esk_R, the adversary can get $(m \times (L-1))$-bit of information about the equivalent subkey. More precisely, a linked fault between $x[i]$ and $x[i+1]$ causes the value of $\Delta esk[i, i+1]$ to leak as shown in Algorithm 1 where $0 \leq i \leq L-2$. So the attacker can obtain the values $\Delta esk[i, i+1]$ for all $0 \leq i \leq L-2$ by repeating the attack $(L-1)$ times. Hence, if the attacker guesses the value of $esk_R[0]$, he can obtain $esk_R[j]$ for $1 \leq j \leq L$ as shown in Equation (7).

$$esk[j] = esk[0] \bigoplus_{\ell=0}^{j-1} \Delta esk_R[\ell, \ell+1] \quad (7)$$

Therefore, the number of candidates for the last equivalent key esk_R significantly decreases from $2^{m \times L}$ to 2^m by applying $(L-1)$ distinct linked faults. This is a generic and practical attack applicable to any SPN cipher with a time complexity of $(L-1)^2$ simple XOR operations and requiring only $(L-1)$ faulty ciphertexts from distinct faults.

2) General Framework for Middle Rounds: Assume that u and v, two $m-$bit intermediate values in the r-th round, are linked after fault injection. In other words, u' and v satisfy the relation $u' = l(v)$, for example, they are equal. The intermediate values u and v, as well as some subkey bits over the last round(s), influence a portion of the ciphertext. In other words, the intermediate values u' and v can be determined by guessing the value of these subkey bits. These bits are referred to as the target key bits. There are $\tau = 2^\kappa$ candidates for a κ-bit target key. Given N faulty ciphertexts $C'_1, C'_2, ..., C'_N$, we can partially decrypt the ciphertexts for a guessed key and check whether the relation $u' = l(v)$ holds. This technique is illustrated in Algorithm 2. The relation $u' = l(v)$ holds for the correct key in all cases, but with a probability of $2^{-m \cdot N}$ for the wrong key.

Equation (8) can be used to figure out how many faulty ciphertexts are needed to eliminate all the wrong candidates for the target subkey bits.

$$\tau \cdot 2^{-m \cdot N} < 1 \Rightarrow 2^{\kappa - m \cdot N} \leq 1 \Rightarrow \kappa - m \cdot N \leq 0 \Rightarrow N \geq \frac{\kappa}{m} \quad (8)$$

Regarding Equation (8), one should note that the actual probability of holding the characteristic for a wrong key may differ from $2^{-m \cdot N}$. Peeling the last rounds for a guessed key and checking a characteristic for the intermediate value(s) is a key-recovery method that dates back to the 1990s, before fault attacks were introduced. As with all previous approaches, due to the impossibility of approximating the exact probability of a wrong key, we adhere to the well-known *"hypothesis of wrong-key randomization"* [49], which is widely embraced by cryptographers and states that the key-dependent bias is significantly greater for the correct key than for a wrong key. This hypothesis comes from the fact that decrypting a ciphertext with a wrong key during the last rounds can be thought of as encrypting it again with a random key over more rounds. Therefore, when estimating the probability of a wrong key, we implicitly use an approximation that may not be accurate in reality. But, like other studies, the attack will work in real life with the same amount of data complexity or a little bit more data, depending on the situation. As will be described in Section V, the results of our experiments in various scenarios indicate that our approximation is realistic and Equation (8) provides a reasonable estimation in practice.

Application to PRESENT The time complexity of performing Algorithm 2, depends on the number of

Algorithm 2 Key-recovery process in LDFA

```
Require: Faulty ciphertexts $C'_1, C'_2, \ldots, C'_N$, Key candidates
    for last round(s) $k_0, ..., k_{\tau-1}$.
Ensure: Correct Key
1: for $\ell = 0$ to $(\tau - 1)$ do
2:     for $h = 1$ to $N$ do
3:         $(u', v) \leftarrow E^{-1}_{k_\ell}(C'_h)$        ▷ Partial decryption
4:         if $u' = l(v)$ then
5:             $cnt[\ell] = cnt[\ell] + 1$
6: return $argmax_\ell(cnt[\ell])$
```

candidates for the involved subkey bits $\tau = 2^\kappa$. The number of target key bits (κ) is determined based on the permutation layer used in the SPN block cipher. Similar to other attacks, LFA can be applied to earlier rounds (before the final round) of block ciphers with weaker linear layers. This part examines PRESENT, which employs a bit permutation as the linear layer. Bit permutation has been used frequently in lightweight block ciphers such as PRESENT because it is an effective and nearly cost-free method. Let us denote the i-th bit of the state x by $x\{i\}$ where the leftmost one is 0. Besides, we denote the concatenation of $\{i, i+1, ..., j\}$-th bits of x by $x\{i-j\}$ where $i < j$. In this part, we consider two consecutive nibbles in the 30th round of PRESENT that are linked following the injection of a fault. More precisely, we consider a fault in which the two nibbles $y_{30}\{0-3\}$, and $y_{30}\{4-7\}$ after the substitution layer in the 30th round become equal. Figure 2 depicts the propagation of the linked fault over the last two rounds of PRESENT, with the active bits shown in red. As seen in Figure 2, four Sboxes become active in the last round; hence, Algorithm 2 can be executed by guessing 24 bits of the subkeys. More precisely, 16 bits from the last round key $sk_{31}\{0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60\}$ and 8 bits from the 30th round $sk_{30}\{0, 1, 16, 17, 32, 33, 48, 49\}$ are involved. Since the number of involved subkey bits is $\kappa = 24$, the time complexity of performing Algorithm 2 is $\tau = 2^{24}$, which is feasible. The required number of faulty ciphertexts is around $N = \frac{24}{4} = 6$, which is determined based on Equation (8). Since the key length of the PRESENT master key is 80 bits, 56 bits are remaining to be determined. The remaining bits are similarly retrievable by injecting linked faults on other nibbles during the 30th round of PRESENT. If each repetition of Algorithm 2 leaks 24 bits of key information, this method must be repeated a maximum of four times to retrieve the entire key. So the total data complexity is at most $4 \cdot 6 = 24$ faulty ciphertexts.

C. Linked Ineffective Fault Attack (LIFA)

Some attacks concentrate on ineffective faults. As they let the attacker circumvent several fault countermeasures, such as detection-based and infection-based countermeasures. Examples are IFA, SIFA, etc.

LFA, unlike IFA, does not require complex equipment, and unlike SIFA, does not rely on a static that cannot be distinguished by a small number of ciphertexts. In addition, as we will show in Section III-E, LIFA is substantially less susceptible to undesired faults than SIFA. Therefore, linked fault analysis appears to be a suitable technique when a redundancy-based countermeasure is utilized in the target device. The following is a description of how LFA can be easily adapted to be performed over ineffective ciphertexts.

We begin by describing a linked fault in the final round of an arbitrary SPN block cipher that causes the word $x'_R[j]$ to equal to $x_R[i]$, or the word $y'_R[j]$ to equal to $S(x_R[i])$. The faulty ciphertext C' is unavailable to the attacker when redundancy-based countermeasures are present. Nevertheless, if $y_R[i] = y_R[j]$, the fault does not affect the processed data. In other words, if the relation $y_R[i] = y_R[j]$ holds, the fault is ineffective, and the device returns the output which is a fault-free ciphertext. The probability of observing an ineffective ciphertext is 2^{-m}. This indicates that, on average, the attacker must repeat the process 2^m times for each fault. Then, the attacker can retrieve m-bit information about the equivalent subkey in a manner similar to the previous subsection by observing an ineffective event. More precisely, $w_R[i]$ and $w_R[j]$ have a similar relationship, and Algorithm 1 can be used similarly but over ineffective ciphertexts.

$$\Pr(w'_R[i] \oplus w'_R[j] = \alpha|\mathsf{i}) = \begin{cases} 1 & \alpha = \Delta esk_R[i, j], \\ 0 & \alpha \neq \Delta esk_R[i, j]. \end{cases} \quad (9)$$

Generally speaking, linked faults occur between two m-bit intermediate values u and v, where m is quite small (typically 4 or 8 in most ciphers). Adopting Algorithm 2 over ineffective events is therefore highly efficient, as one ineffective fault happens on average for every 2^m faulty computation. Since LFA requires a very small number, the total amount of data required for LIFA might be significantly less than for other ineffective fault variant attacks.

D. Unknown Fault's Location

In all proposed methods in Section III-B and Section III-C, we assumed that the attacker knows the order in which the Sbox calls the words. Similarly, in most fault attacks, it is implicitly assumed that the attacker is familiar with the implementation, as the fault must be injected at a precise time. In some applications, the attacker does not necessarily know the implementation's details, since they are not publicly available or due to hiding-based countermeasures (such as dummy operations or shuffling). This makes the execution of the fault attacks difficult. In this part, we show how the assumption of exact knowledge of the loading order of intermediate values can be relaxed on the condition of using a linked fault. To demonstrate the advantage of a key-dependent link in a faulty ciphertext, we will analyze the LFA on the last round on an arbitrary SPN cipher proposed in Section III-B.1.

$$\begin{aligned} \ell_1 \cdot \ell_2 \cdot \tau \cdot 2^{-m \cdot N} < 1 &\Rightarrow 2^{\log(\tau)+\log(\ell_1)+\log(\ell_2)-m \cdot N} \leq 1 \\ &\Rightarrow \log(\tau) + \log(\ell_1) + \log(\ell_2) - m \cdot N \leq 0 \\ &\Rightarrow N \geq \frac{\log(\tau) + \log(\ell_1) + \log(\ell_2)}{m} \end{aligned} \quad (10)$$

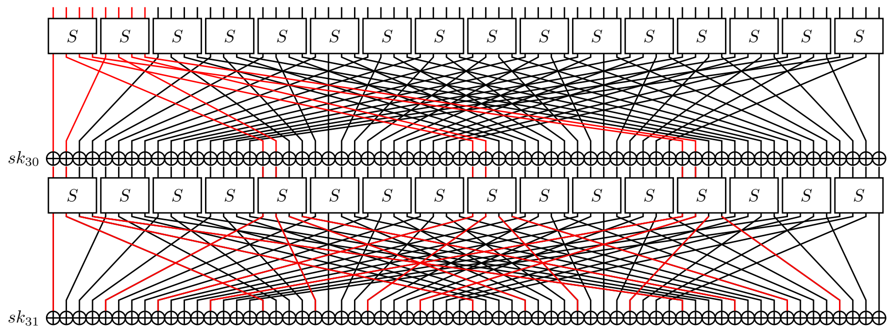


Fig. 2: Application of LDFA on the 30th round of PRESENT

The last round attack on an SPN cipher has respectively $\ell_1 = L$, $\ell_2 = L-1$, and $\tau = 2^m$ possibilities for i, j, and esk_R (here it refers to $\Delta esk_R[i,j]$), while m-bit filtering is provided. In the case of AES, given $N = 3$ faulty ciphertexts, the attacker can determine not only the value of $\Delta esk_{10}[i,j]$ but also the location of the fault and how it is linked to the other words. It should be noted that a similar technique may be applied to LIFA. However, rather than exploiting faulty ciphertext, the attacker applies Algorithm 3 to ineffective ciphertext.

Algorithm 3 Finding the location of fault

```
Require: Faulty ciphertexts $C'_1, C'_2, \ldots, C'_N$    ▷ In case of
    LIFA, only ineffective ciphertexts are considered
Ensure: Location of fault and its linked word $(i,j)$, and the
    correct value of $\Delta esk_R[i,j]$
1: for $h = 1$ to $N$ do
2:     for $i = 0$ to $L-1$ do
3:         for $j = i+1$ to $L-1$ do
4:             $\delta \leftarrow \mathcal{L}^{-1}(C'_h)[i] \oplus \mathcal{L}^{-1}(C'_h)[j]$
5:             $cnt_{i,j}[\delta] = cnt_{i,j}[\delta] + 1$
6: return $argmax_{i,j,\delta}(cnt_{i,j}[\delta], i, j)$
```

E. Missed and Unwanted Faults

The assumption of perfect fault injection is rarely practical in the real world. It is possible that the fault is not injected as desired due to several factors. In this part, we first look at how these occurrences affect the LFA and then show that LFA is less affected by undesirable events than other fault analyses.

1) Unwanted and Missed Faults: The instruction skip could occasionally happen at an undesirable time (location). Such events are referred to as "unwanted faults." They happen either as a result of incidental noise that naturally arises in real-world experiments or as a result of purposeful noise produced by hiding countermeasures like shuffling and dummy operations. These countermeasures are designed to make fault injection into a specific location more challenging by limiting the attacker's knowledge of the precise timing of instruction execution and data processing. Dummy operations cost significantly more and have a limited effect against attacks like SEFA. Shuffling, however, seems to be an efficient, low-overhead defense against the vast majority of fault attacks.

Other undesirable situations are missed faults. As introduced in section I-B.2, there is missed fault where no instruction skip happens after the fault injection. Missed faults are inevitable even with a reliable and perfect setup, and since it is difficult to identify them from ineffective faults, they have a significant impact on fault attacks that use ineffective events. This feature has encouraged the development of novel attacks, such as [23], [46], which employ correct ciphertexts that correlate to effective faults. To acquire non-faulty effective ciphertexts, an attacker must, however, perform the encryption twice: once to compute the faulty encryption to see if it is effective, and again to obtain the corresponding correct ciphertext.

2) LFA in the Presence of Undesired Cases: Most fault attacks are impacted by missed and unwanted faults, since they are typically indistinguishable from desired faults, especially when the attacker lacks input control to repeat the experiment with a fixed input. In this part, we demonstrate how LFA may manage undesirable situations without requiring input-controlled computation repetition. We solely discuss situations where extra countermeasures, such as shuffling, are not employed. Shuffling will be discussed later in Section IV-A. It indicates that the noise is the result of a missed fault or that the fault was introduced at the incorrect moment due to a weakness in the attack's setup. We denote the rate of missed faults and unwanted (and unintentional) faults by Π_m and Π_u, respectively. When N experiments are performed, there are approximately $N\times(1-(\Pi_m+\Pi_u))$ cases in which the relation $u' = l(v)$ always holds. Because no fault occurs or occurs in an incorrect location in the remaining $N \times (\Pi_m + \Pi_u)$ cases, the relation $u' = l(v)$ holds with the probability of 2^{-m} on average. In contrast to the situation discussed in Section III-B and Section III-C, the relation $u' = l(v)$ is not deterministic. However, as shown in Equation (11), $u' = l(v)$ satisfies with different probabilities for the correct key and a wrong key.

$$\Pr(u' = l(v)) = \begin{cases} (1-(\Pi_m+\Pi_u)) + 2^{-m}\cdot(\Pi_m+\Pi_u) & \text{Correct key} \\ 2^{-m} & \text{Wrong key.} \end{cases} \tag{11}$$

Hence, the attacker can apply Algorithm 2 similarly, but it requires more data.

IV. Countermeasures and Further Discussions

In this section, we investigate the efficiency of LFA against known countermeasures.

A. Shuffling

The performance of the majority of fault attacks is severely impacted by shuffling. Shuffling randomizes the order of operations in the executions of the Sboxes. As a result, injecting a fault at a certain time does not necessarily affect a fixed intermediate value.

To generate a linked fault like $u' = l(v)$, the values u and v must be processed sequentially. However, this is not always the case when shuffling is present. Let us assume that the linked fault is introduced during a certain processing time interval for two variables. In the presence of shuffling, there are $L \cdot (L-1)$ distinct ways of processing two words in this period of time. After injecting a fault, if shuffling is employed, there are two possible outcomes. The intended fault occurs when the intermediate values u and v (or v and u) are processed. The likelihood of such an occurrence is $\frac{2}{L(L-1)}$. The intermediate values processed during fault injection deviate from what the attacker desires with a probability of $1 - \frac{2}{L(L-1)}$. The relation $u' = l(v)$ may be randomly satisfied with a probability of 2^{-m} in these cases. As a result, as illustrated in Equation (12), $u' = l(v)$ satisfies with different probabilities for the correct key and a wrong key.

$$\Pr(u' = l(v)) = \begin{cases} \frac{2}{L(L-1)} + 2^{-m} \cdot (1 - \frac{2}{L(L-1)}) & \text{Correct key} \\ 2^{-m} & \text{Wrong key.} \end{cases} \quad (12)$$

Equation (12) indicates that the attacker can use Algorithm 2 to retrieve the key, but he needs more data. Similarly, if one portion of the faulty ciphertext C' (or $\mathcal{L}^{-1}(C')$) is linked to another via a key-dependent relationship, Algorithm 1 can be used. The amount of data N required to extract the correct key is determined by the number of wrong keys and the probability $\Pr(u' = l(v))$ for the correct and wrong keys. Our experimental results which will be presented in Section V demonstrate that LDFA and LIFA can be applied to AES by around 350 and 89,600 faulty computations, respectively. Note that in the presence of shuffling, an attacker can utilize the same collection of faulty ciphertexts to recover $\Delta_{i,j}$ for different values of i and j. It is proved in [33] that the required data for performing statistical fault attacks such as SIFA grows quadratically in the number of operations that are shuffled. In case of AES or PRESENT, the number of required data increases from N to $N \times 2^{16}$. It demonstrates that shuffling is far less effective against LFA than statistical fault analysis techniques such as SFA, SIFA, and SEFA.

B. Repetition of Instructions

The primary premise of the instruction-level countermeasures is that idempotent instructions, such as the move instruction, can be repeated several times without changing the output [50]. Inserting multiple faults is a way to undermine such an implementation. It has also been shown that their security can be violated even by adding a single clock glitch using a low-cost tool [42]. Such security issues with the protection provided by numerous copies of the instructions mean that these countermeasures should be used with caution. Despite these problems, it is clear that instruction-level countermeasures, like repeating idempotent instructions, can make it much harder to use LFA, especially when used with other countermeasures like shuffling.

C. Masking

Masking is the most widely used protection against side-channel attacks. Even though it is not intended to prohibit fault attacks, it can make it more challenging to use the majority of fault attacks in practice. In this part, we show that LIFA might work even when first-order masking and detection-based countermeasures are in place.

1) Precomputed Masked Sbox: A Sbox can be easily masked in software [51] by generating and storing masked Sbox tables $S_m(x \oplus m) = S(x) \oplus m'$, where m and m' are the input and output masks, respectively. The generation of such tables should be performed for all mask values. Therefore, as the number of masks increases, the time required to generate these tables, and the amount of memory required to store the tables increase as well. LIFA can be applied without additional difficulty if $S_{m'}(X[i]) \oplus m$ equals $S_m(X[j]) \oplus m$, as explained in Section III-C when using the first-order masking implementation of AES provided in [51].

Algorithm 4 SIFA-protected masked implementation of $\chi3$

```
Require: $(a_0, a_1, b_0, b_1, c_0, c_1)$
Ensure: $(r_0, r_1, s_0, s_1, t_0, t_1)$
1:  $T_0 \leftarrow b_0c_1$;   $T_2 \leftarrow a_1b_1$
2:  $T_1 \leftarrow b_0c_0$;   $T_3 \leftarrow a_1b_0$
3:  $T_0 \leftarrow T_0 \oplus a_0$;   $T_2 \leftarrow T_2c_1$
4:  $r_0 \leftarrow T_0 \oplus T_1$;   $t_1 \leftarrow T_2T_3$
5:
6:  $T_0 \leftarrow c_0a_1$;   $T_2 \leftarrow b_1c_1$
7:  $T_1 \leftarrow c_0a_0$;   $T_3 \leftarrow b_1c_0$
8:  $T_0 \leftarrow T_0 \oplus b_0$;   $T_2 \leftarrow T_2a_1$
9:  $r_0 \leftarrow T_0 \oplus T_1$;   $t_1 \leftarrow T_2T_3$
10:
11: $T_0 \leftarrow a_0b_1$;   $T_2 \leftarrow c_1a_1$
12: $T_1 \leftarrow a_0b_0$;   $T_3 \leftarrow c_1a_0$
13: $T_0 \leftarrow T_0 \oplus c_0$;   $T_2 \leftarrow T_2b_1$
14: $r_0 \leftarrow T_0 \oplus T_1$;   $t_1 \leftarrow T_2T_3$
```

2) SIFA-Protected Sbox: SIFA has proven to be applicable even when masking is present. A SIFA-protected masking scheme that can be used in both software and hardware implementations was proposed by [52]. Algorithm 4 illustrates a SIFA-protected masked implementation of $\chi3$, which we consider here for simplicity.

In certain circumstances, a linked fault may result in a scenario where an ineffective event leaks knowledge about an intermediate value, which LIFA subsequently takes advantage of. In each step of Algorithm 4, the values of T_0 and T_1 always consist of both shares of a single value. Let us assume that a linked fault attack causes T_1 to become equal to T_0 in the first step. An ineffective fault only happens when $T_0 = T_1$ and, equivalently, $b_0c_1 = b_0c_0$. The injected linked fault is always ineffective if $b_0 = 0$. However, when $b_0 = 1$, the injected linked fault is only ineffective when $b_0 = b_1$ or equivalently $b = 0$. The same holds for T_2 and T_3 in each step. Consider a second, more straightforward example where the input values are changed so that a_0 and a_1 are loaded sequentially in the input of the Sbox. In this instance, LFA may lead a_0 and a_1 to become equal. Consequently, the fault is ineffective only if $a_0 = a_1$ or, equivalently, $a = 0$. Hence, the ineffective event may leak information about intermediate values.

This observation does not contradict the security claim made in [52], but it does provide strong evidence that even the security of SIFA-protected masking should be strengthened by instruction-level countermeasures in software implementations or that the order of instructions should be taken into consideration to thwart attacks like LIFA. On the other hand, we note that the SIFA-masked implementation does not consider Keccak in the MAC mode, so there is no key for recovery. We just aimed to demonstrate that the non-linear layer of this implementation is also vulnerable to the LIFA.

V. Experiments

A. Experimental Setup

We realized the linked fault using a typical setup. We set our target system's (μController's) clock to be fed by an external clock. Then we built the clock with a Field-Programmable Gate Array (FPGA). The first achievement is the ability to increase the μController's working frequency from its highest achievable value by tens of times, using the FPGA's internal Phase-Locked Loop (PLL). In an ATMEGA328p (an 8-bit AVR μController) case, we had an increase from 16 MHz to 160 MHz. The second achievement is the ability to induce the fault at the exact desirable point in time and algorithm calculations in every repetition of the fault inducement. The FPGA's high synchronicity with the μController allows us to have accurate control over the time and location of our fault. The mechanism includes an alerting signal from the desired point of the algorithm (for instance, the start of the next to last round of the AES). From the FPGA side, this signal triggers the commencement of the frequency perturbation. Two more subsidiary parameters aid in finely determining the exact point at which the frequency rise will result in a linked fault, as well as the duration of the increase. We refer to these two parameters as the fault's *start* and *offset* values, respectively.

The AES and PRESENT implementations are carried out through conventional C codes with standard publicly available libraries [2],[3]. Since our attack involves only a single instruction skip, the performance of the attack will not be significantly affected by the use of assembly-optimized code.

Based on the recommendation by the reviewers, we also performed statistical fault attacks in the same platform.

[2]https://github.com/suculent/thinx-aes-lib

[3]https://github.com/Pepton21/present-cipher/blob/master/PRESENT.c

B. Application of LDFA and LIFA

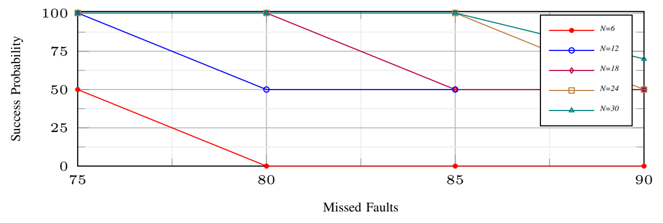
Firstly, we take the AES implementation and apply LFA to the last round of cipher. To convert $x_{10}[4]$ to $x_{10}[0]$, we applied a linked fault. Our primary results indicate that, out of 23862 tests, there were nearly 99.1% successfully linked faults. Besides, we observed ineffective, missed, and unwanted faults in 108, 42, and 54 of the experiments, respectively. Our trials' effective rate ($108/23862 = 2^{-7.79}$) is quite close to our prediction (2^{-8}). This result demonstrates that LIFA is also relevant if LDFA works. We repeated Algorithm 1 for one faulty ciphertext 20 times and were able to obtain the correct value of $\Delta_{0,4} = esk_{10}[0] \oplus esk_{10}[4]$ in all cases. This is not unexpected given that our fault configuration provides us with a high success rate and a minimal number of missing or undesirable faults. We applied similar linked faults in other locations and could obtain 15-byte information about esk_{10}. We finally used Equation (7) to reduce the number of candidates for esk_{10} from 2^{128} to 2^8.

We then repeated the procedure, this time for the PRESENT implementation. We targeted its last round to convert $x_{31}[0]$ to $x_{31}[1]$. In the experiment targeting the last round of the PRESENT, out of 34575 tests, there were nearly 93.3% successful linked faults. The number of ineffective, missed, and unwanted faults were 2164, 127, and 16, respectively. We repeated Algorithm 1 for one faulty ciphertext 20 times and could retrieve the correct value of $\Delta_{0,4} = sk_{31}[0] \oplus sk_{31}[4]$ in all cases. The remaining procedure closely resembles that described for AES. However, the results demonstrate that Algorithm 1 can be applied to various ciphers.

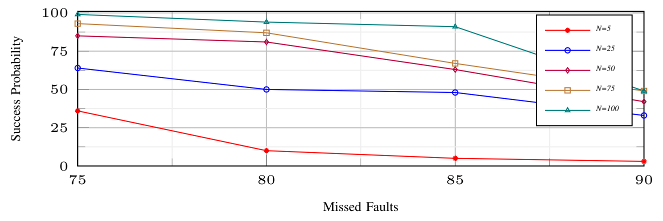
After that, we target the next to the last round of the PRESENT for the instance illustrated in Figure 2 (ref. to Section III-B.2). There were successfully linked faults in nearly 93.4% of 25454 tests. The number of ineffective, missed, and unwanted faults were 1552, 117, and 21, respectively. Equation (8) estimates that we need at most $N = 24/4 = 6$ faulty ciphertexts to perform Algorithm 2 successfully. However, our experiments demonstrate that even $N = 4$ faulty ciphertexts were enough in practice for retrieving the involved subkey bits presented in Figure 2. We repeated Algorithm 2, 20 times on the 30th round of PRESENT, and in all cases, we could retrieve the key uniquely by utilizing four faulty ciphertexts. Our experiments demonstrate that Equation (8) provides a reasonable estimation for the required data to perform Algorithm 2 successfully.

As stated in Section III-D, the attacker might not be aware of the precise location of the fault. Even though we were aware of this information, we assumed the attacker testing Algorithm 3 would not be able to access it. We executed Algorithm 3, 1,500 times and determined that the success probability of finding the fault location and the linked value for $N = 2$ and $N = 3$ is 65.84% and 99.84%, respectively.

Since very little noise is present in our setup, to investigate the effect of missed faults discussed in Section III-E, we followed this procedure: We generated a free-fault ciphertext with a probability of P_{missed}, and used one of the faulty ciphertexts obtained from the practical experiment with a probability of $(1 - P_{missed})$. Then, we repeated the attack with varying quantities of faulty ciphertexts to determine the probability of success for different missed fault rates. Figure 3 illustrates the outcomes for both AES and PRESENT which demonstrate that LFA is very effective in this scenario. For instance, by utilizing only 12 (respectively 100) faulty ciphertexts, LDFA can retrieve 8 (respectively 4) bits of information about the last round key equivalent of AES (respectively PRESENT) in the event of a 75% missed fault rate.



(a) AES-128



(b) PRESENT

Fig. 3: Success probability of retrieving a word of the last round key in the presence of missed faults, N denotes the number of available faulty ciphertexts.

C. LDFA and LIFA in the Presence of Countermeasure

We performed LDFA on AES implementation in the presence of shuffling. Shuffling is used with the nonlinear layer of the AES. We ran the tests 10 times and discovered that the correct key can be identified exclusively with around 350 faulty ciphertexts. Figure 4 reflects the results of this experiment. Accordingly, LIFA can retrieve the correct key by utilizing around $350 \times 256 \simeq 2^{16.45}$ data.

We then applied LFA on a publicly available byte-masked implementation[4] to check if it works with masked look-up tables in practice. We found that LDFA can easily be applied to this implementation. By introducing one linked fault, LDFA can get 8-bit information about the last round's key, just like in the typical scenario. We ran our experiments 1400 times and identified 7 ineffective and 8 missed faults. The fact that the ineffective rate is close to 2^{-8} proves that LIFA applies in this instance, as well.

Additionally, we applied LFA to the SIFA-protected, masked implementation of $\chi 5$, which is publicly accessible[5]. We tried to apply LFA to the given source, but it did not work because the variables we wanted to be linked are not processed in order (see Section IV-C.2). But to check if LFA could be applied to SIFA-protected masking implementations, we moved code line 351 to line 347. This modification does not alter the SIFA-masked implementation principle described in [52], but it makes it so that two shares of the single variable (like a) to be processed successively. Then we used LFA on this changed implementation and saw that the target values, which are called a_0 and a_1, were linked with a success rate of nearly 98.2% in 27432 experiments. This fault is only ineffective if $a_0 = a_1$ or, equivalently, $a = a_0 \oplus a_1 = 0$. Thus, LIFA can be employed for this kind of implementation too. This does not negate the security assertion made in [52], as stated in Section IV-C.2, but it provides compelling proof that even SIFA-protected masking must be used with caution due to LIFA's impact on instruction orders.

D. Comparison With Previous Attacks

We will compare LDFA and LIFA with previous works in this section. As noted in Section I-A, a fair comparison necessitates taking into account both the required data and the assumptions made about the attacks and their performance in various scenarios (such as undesired faults, the existence of various countermeasures, etc.) Therefore, we will compare LDFA (LIFA) to transient fault attacks, which cannot (can) bypass redundancy-based countermeasures.

1) Comparing LDFA with Previous Works: According to the published results, DFA, IDFA, DFIA, and CFA on AES require only a few faulty ciphertexts. Similarly, LDFA is applicable with a relatively minimal quantity of data; on AES. For example, it only requires 15 faulty ciphertexts for full key recovery, which is comparable to DFA attacks and substantially less than SFA. However, similar to SFA and in contrast to DFA, IDFA, DFIA, and CFA, LDFA does not require input control. This is a significant factor in various real-world applications, which are covered in detail in Section I-A.

The SFA can take advantage of any non-uniformly distributed fault. As explained in Section II-D, the required data for performing SFA is directly dependent on the bias that exists in the faulty word. The original paper that proposed SFA [24] does not offer experimental results, instead providing simulations based on theoretical models. According to the simulations in [24], SFA requires 80 faulty ciphertexts to obtain the AES key if the fault model is stuck at an unknown value, which is difficult to produce in practice. To provide a clearer instance, we attempted to implement SFA on our platform. In the last round of AES, we considered one

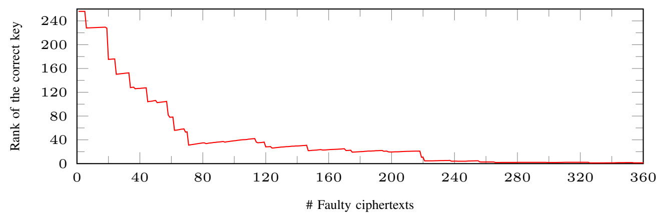
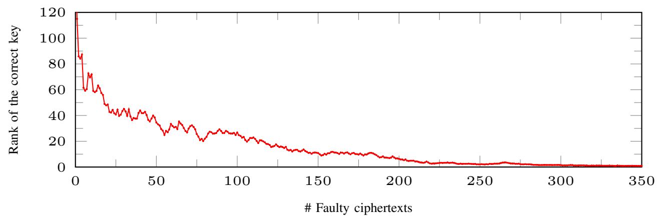


Fig. 4: The rank of the correct key for LDFA on last round of AES in the presence of shuffling

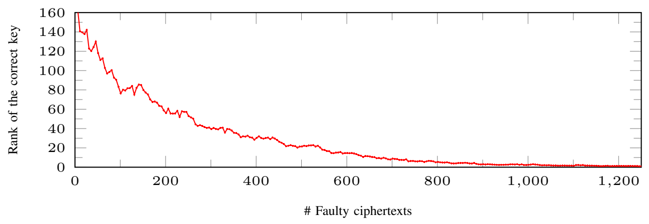
[4]https://github.com/Secure-Embedded-Systems/Masked-AES-Implementation/tree/master/Byte-Masked-AES

[5]https://github.com/sifa-aux/countermeasures/blob/master/keccakf200-avr8/main.c

byte and attempted to inject a biased fault on the target byte. As expected, the capacity of the biased fault varies greatly depending on the circumstances. To discover the most suitable situation to apply SFA, we employed the trial-and-error method. The *start* parameter was the beginning of the add round key at the end of the ninth round, the *offset* parameter was 1280 clocks, and the fault duration was 24 clocks. We were able to uniquely obtain the correct key by utilizing around 290 faulty ciphertexts. We repeated the key-recovery attack on ten separate sets of data and computed the average rank of the correct key based on the number of available faulty ciphertexts. Figure 5a depicts the results of our experiments.



(a) Without missed faults



(b) Missed faults with 80% rate

Fig. 5: The rank of the correct key on unprotected AES for applying SFA to the last round of AES

Our results, depicted in different figures demonstrate that LDFA is far less sensitive to undesired faults, whether they come from the noise associated with the attack's setup or from countermeasures such as shuffling. Even with a noisy setup and a missed fault rate of 75%, LDFA can obtain 8-bit key information from AES with only 12 faulty ciphertexts. Therefore, the total key space can be decreased from 2^{128} to 2^8. Figure 5b shows that with 80% missed fault rate, SFA key-recovery requires around 1275 ciphertexts. Compared to the LDFA's required data in the same scenario, SFA has a sharper increase. Finally, we applied SFA to an AES implementation that uses shuffling. Figure 6 presents the results of this test. As it demonstrates, key recovery requires more than 10000 ciphertexts which is notably more than what is required for LDFA when shuffling is present.

2) Comparing LIFA with Previous Works: The required data for SEFA and SIFA, like SFA, is based on the bias that exists on the faulty word over effective and ineffective events, respectively. Furthermore, the effective rate and ineffective rate influence the amount of total data required for SEFA and SIFA, respectively (ref. Section II-D). The original SIFA paper [27] offers a variety of experimental results demonstrating that data complexity varies greatly (from 1000 to 130,000 faulty computations) depending on the target device, implementation, setup noise, and so on. The same is true for SEFA [23].

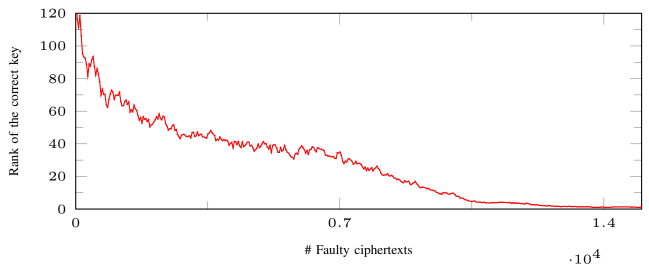
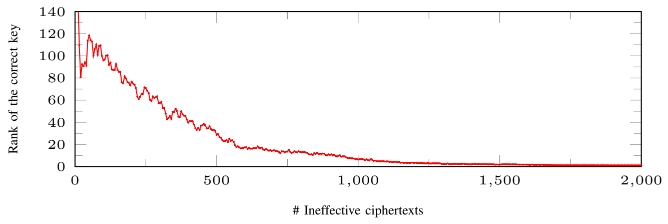
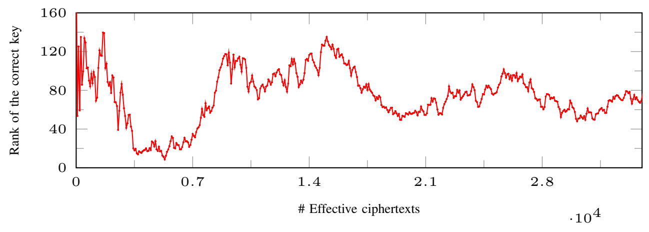


Fig. 6: The rank of the correct key on AES for applying SFA to the last round of AES in the presence of shuffling

Because of this, it is difficult to compare SIFA, SEFA, and LIFA fairly. However, we applied SIFA and SEFA to our platform to prepare a fair comparison. As previously explained, we target a Sbox in the last round of AES. We repeated the key-recovery attack on 10 different sets of data and computed the average rank of the correct key based on the number of accessible ineffective and effective ciphertexts for SIFA and SEFA. Figure 7a and Figure 7b depict the outcomes. SIFA requires 1900 ineffective ciphertexts, as shown in Figure 7a. Given that our experiment's ineffective rate was roughly 10%, the overall data complexity is 19,000 faulty computations, which is significantly more than LIFA. We remind that LIFA requires only one ineffective ciphertext (i.e., $1 \times 2^8 = 256$ faulty computations) as presented in Section V-B. Finally, even with tens of thousands of faulty computations, SEFA was unable to obtain the key. The reason could be that our setup has minimal noise. SEFA can outperform SIFA only in a noisy setup, as indicated in [23].



(a) Results for SIFA



(b) Results for SEFA

Fig. 7: The rank of the correct key on protected AES for applying SIFA & SEFA against to the last round of AES

As indicated in [23], [27], as noise grows, the data complexity of SIFA or SEFA increases considerably. This involves noise caused during the attack's setup as well as noise caused by the countermeasures. We simulated missed faults by adding non-faulty ciphertexts to the tests, as we did with LIFA, to assess the effect of noise and compare SIFA

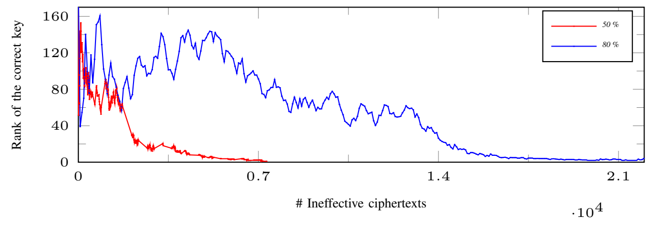


Fig. 8: The rank of the correct key on protected AES for applying SIFA against to last round of AES in the presence of missed faults

and LIFA in a noisy scenario. We repeated the key-recovery attack on ten separate sets of data and calculated the average rank of the correct key based on the number of available ineffective ciphertexts for SIFA in two scenarios: once with a 50% missed fault rate and once with an 80% missed fault rate. Figure 7b displays the results. As expected, the benefit of the LIFA over the SIFA grows as noise increases. SIFA requires between 7,000 and 17,000 ineffective ciphertexts (70,000 and 170,000 faulty computations, respectively) in the case of 50% and 80% missed faults. In case of 80% missed faults, LIFA necessitates only 18 ineffective ciphertexts ($18 \times 256 = 4608$ faulty computations) as it is shown in Figure 3a.

As stated in [27] and [23], shuffling has a significant impact on SIFA and SEFA. Even SEFA, which is not greatly impacted by missed faults, can be impacted by shuffling. This is because missed faults do not influence effective events, whereas shuffling has a direct effect. We could not retrieve the correct key by applying SIFA to the last round of AES, even by using 40,000 ineffective ciphertexts. As the ineffective rate in our experiment is 10%, the correct key cannot be found by SIFA even if the adversary performs 400,000 faulty computations in a case where the shuffling exits. As it is presented in Figure 4, LDFA can retrieve the correct key in the presence of shuffling by using 350 faulty ciphertexts. Hence, LIFA can requires $350 \times 2^8 = 89,600$ faulty computations to find the correct key in the presence of shuffling. And finally, the fact that even SIFA-protected masking can be broken by LIFA shows how powerful it could be.

3) Summary: To facilitate the comparison, we give the results of several attacks that do not require input control in different circumstances side by side in this section. Table II shows a summary of the practical outcomes for LDFA and SFA. As expected, LDFA has a significant advantage against SFA. Table III shows an overview of practical results of LIFA and SIFA. We would like to remind that the data necessary for SIFA can change substantially depending on the circumstances. As a result, we have taken into account two kinds of SIFA outcomes in this table. The first row of the table demonstrates the practical outcomes of SIFA on the platform where we executed the LIFA attack. The results of the SIFA simulation for a 4-bit random-AND fault, which produces a faulty distribution with a high bias are shown in the second row of Table III. This circumstance is challenging to achieve in practice. The publicly available codes from [23] [6] were used for these simulations. In the case of 4-bit random-AND, SIFA needs approximately 1,450 and 8,454 faulty computations for a missed fault rate of 50% and 80%, respectively. Table III demonstrates that when the setup is noisy or a common countermeasure, such as shuffling, is applied, the gap between LIFA and SIFA grows larger. Our experiments do not prove LIFA can always beat SIFA, but they show that in a noisy environment, LIFA outperforms SIFA with a high likelihood.

[6]https://github.com/Navidvafaei/SEFA

TABLE II: Comparison practical results of LDFA and SFA

	# Required faulty ciphertexts		
	Unprotected	**Missed Fault** (80%)	**With Shuffling**
SFA	290	1275	$13,000 \simeq 2^{13.6}$
LDFA	1	12	350

TABLE III: Comparison of LIFA and SIFA

	# Required Faulty Computations					
	Ineffective Rate	**With Detection-based Countermeasure**	**Missed Faults (50%)**	**Missed Faults (80%)**	**With Shuffling**	**Applicable to SIFA-protected Masking**
SIFA[†]	10%	$1900 \times 10 = 19,000$	$7000 \times 10 = 70,000$	$17,000 \times 10 = 170,000$	$> 40,000 \times 10 = 400,000$	No
SIFA[‡]	32%	232	$1,450$	$8,454$	$> 400,000$	No
LIFA	$2^{-7.79} \simeq 2^{-8}$	$1 \times 256 = 256$	$12 \times 256 = 3,072$	$18 \times 256 = 4,608$	$350 \times 256 = 89,600$	Maybe

[†] Practical results.
[‡] Simulation results with random-AND model.

Acknowledgements

The authors would like to thank anonymous reviewers for their thoughtful comments, and insightful suggestions which significantly improved this paper.

References

[1] B. Gierlichs, J. Schmidt, and M. Tunstall, “Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output,” in *LATINCRYPT*, ser. Lecture Notes in Computer Science, vol. 7533. Springer, 2012, pp. 305–321.
[2] H. Tupsamudre, S. Bisht, and D. Mukhopadhyay, “Destroying fault invariant with randomization - A countermeasure for AES against differential fault attacks,” in *CHES*, ser. Lecture Notes in Computer Science, vol. 8731. Springer, 2014, pp. 93–111.
[3] J. Feng, H. Chen, Y. Li, Z. Jiao, and W. Xi, “A framework for evaluation and analysis on infection countermeasures against fault attacks,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 391–406, 2020.
[4] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, “The sorcerer’s apprentice guide to fault attacks,” *Proc. IEEE*, vol. 94, no. 2, pp. 370–382, 2006.
[5] E. Biham and A. 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*, ser. Lecture Notes in Computer Science, vol. 1294. Springer, 1997, pp. 513–525.
[6] D. Saha, D. Mukhopadhyay, and D. R. Chowdhury, “A diagonal fault attack on the advanced encryption standard,” *IACR Cryptol. ePrint Arch.*, p. 581, 2009.
[7] S. Ali and D. Mukhopadhyay, “Differential fault analysis of twofish,” in *Inscrypt*, ser. Lecture Notes in Computer Science, vol. 7763. Springer, 2012, pp. 10–28.
[8] ——, “Improved differential fault analysis of CLEFIA,” in *FDTC*. IEEE Computer Society, 2013, pp. 60–70.

[9] S. Ali, D. Mukhopadhyay, and M. Tunstall, “Differential fault analysis of AES: towards reaching its limits,” *J. Cryptogr. Eng.*, vol. 3, no. 2, pp. 73–97, 2013.

[10] D. B. Roy, A. Chakraborti, D. Chang, S. V. D. Kumar, D. Mukhopadhyay, and M. Nandi, “Fault based almost universal forgeries on CLOC and SILC,” in *SPACE*, ser. Lecture Notes in Computer Science, vol. 10076. Springer, 2016, pp. 66–86.

[11] P. Dey, A. Chakraborty, A. Adhikari, and D. Mukhopadhyay, “Improved practical differential fault analysis of grain-128,” in *DATE*. ACM, 2015, pp. 459–464.

[12] S. Saha, D. Mukhopadhyay, and P. Dasgupta, “Expfault: An automated framework for exploitable fault characterization in block ciphers,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 2, pp. 242–276, 2018.

[13] J. Breier, X. Hou, and Y. Liu, “Fault attacks made easy: Differential fault analysis automation on assembly code,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 2, pp. 96–122, 2018.

[14] X. Hou, J. Breier, F. Zhang, and Y. Liu, “Fully automated differential fault analysis on software implementations of block ciphers,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2019, no. 3, pp. 1–29, 2019.

[15] P. Derbez, P. Fouque, and D. Leresteux, “Meet-in-the-middle and impossible differential fault analysis on AES,” in *CHES*, ser. Lecture Notes in Computer Science, vol. 6917. Springer, 2011, pp. 274–291.

[16] X. Zhao, S. Guo, F. Zhang, Z. Shi, C. Ma, and T. Wang, “Improving and evaluating differential fault analysis on LED with algebraic techniques,” in *FDTC*. IEEE Computer Society, 2013, pp. 41–51.

[17] F. Zhang, X. Zhao, S. Guo, T. Wang, and Z. Shi, “Improved algebraic fault analysis: A case study on piccolo and applications to other lightweight block ciphers,” in *COSADE*, ser. Lecture Notes in Computer Science, vol. 7864. Springer, 2013, pp. 62–79.

[18] F. Zhang, S. Guo, X. Zhao, T. Wang, J. Yang, F. Standaert, and D. Gu, “A framework for the analysis and evaluation of algebraic fault attacks on lightweight block ciphers,” *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 5, pp. 1039–1054, 2016.

[19] N. F. Ghalaty, B. Yuce, M. Taha, and P. Schaumont, “Differential fault intensity analysis,” in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2014, pp. 49–58.

[20] J. Blömer and V. Krummel, “Fault based collision attacks on AES,” in *FDTC*, ser. Lecture Notes in Computer Science, vol. 4236. Springer, 2006, pp. 106–120.

[21] Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta, “Fault sensitivity analysis,” in *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, ser. Lecture Notes in Computer Science, S. Mangard and F. Standaert, Eds., vol. 6225. Springer, 2010, pp. 320–334.

[22] S. Saha, A. Bag, D. B. Roy, S. Patranabis, and D. Mukhopadhyay, “Fault template attacks on block ciphers exploiting fault propagation,” in *EUROCRYPT (1)*, ser. Lecture Notes in Computer Science, vol. 12105. Springer, 2020, pp. 612–643.

[23] N. Vafaei, S. Zarei, N. Bagheri, M. Eichlseder, R. Primas, and H. Soleimany, “Statistical effective fault attacks: The other side of the coin,” *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 1855–1867, 2022.

[24] T. Fuhr, É. Jaulmes, V. Lomné, and A. Thillard, “Fault attacks on AES with faulty ciphertexts only,” in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, W. Fischer and J. Schmidt, Eds. IEEE Computer Society, 2013, pp. 108–118.

[25] S.-M. Yen and M. Joye, “Checking before output may not be enough against fault-based cryptanalysis,” *IEEE Transactions on computers*, vol. 49, no. 9, pp. 967–970, 2000.

[26] C. Clavier, “Secret external encodings do not prevent transient fault analysis,” in *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, ser. Lecture Notes in Computer Science, P. Paillier and I. Verbauwhede, Eds., vol. 4727. Springer, 2007, pp. 181–194.

[27] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas, “SIFA: Exploiting ineffective fault inductions on symmetric cryptography,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 3, pp. 547–572, 2018.

[28] B. Selmke, S. Brummer, J. Heyszl, and G. Sigl, “Precise laser fault injections into 90 nm and 45 nm sram-cells,” in *CARDIS*, ser. Lecture Notes in Computer Science, vol. 9514. Springer, 2015, pp. 193–205.

[29] B. Selmke, J. Heyszl, and G. Sigl, “Attack on a DFA protected AES by simultaneous laser fault injections,” in *FDTC*. IEEE Computer Society, 2016, pp. 36–46.

[30] A. Baksi, S. Bhasin, J. Breier, M. Khairallah, and T. Peyrin, “Protecting block ciphers against differential fault attacks without re-keying,” in *HOST*. IEEE Computer Society, 2018, pp. 191–194.

[31] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*, ser. Information Security and Cryptography. Springer, 2002.

[32] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, “Present: An ultra-lightweight block cipher,” in *International workshop on cryptographic hardware and embedded systems*. Springer, 2007, pp. 450–466.

[33] G. Barbu, L. Castelnovi, and T. Chabrier, “Generalizing statistical ineffective fault attacks in the spirit of side-channel attacks,” in *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*, ser. Lecture Notes in Computer Science, vol. 12910. Springer, 2021, pp. 105–125.

[34] E. Trichina and R. Korkikyan, “Multi fault laser attacks on protected CRT-RSA,” in *FDTC*. IEEE Computer Society, 2010, pp. 75–86.

[35] A. Dehbaoui, J. Dutertre, B. Robisson, P. Orsatelli, P. Maurine, and A. Tria, “Injection of transient faults using electromagnetic pulses - practical results on a cryptographic system-,” *IACR Cryptol. ePrint Arch.*, p. 123, 2012.

[36] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz, “Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller,” in *FDTC*. IEEE Computer Society, 2013, pp. 77–88.

[37] A. Dehbaoui, J. Dutertre, B. Robisson, and A. Tria, “Electromagnetic transient faults injection on a hardware and a software implementations of AES,” in *FDTC*. IEEE Computer Society, 2012, pp. 7–15.

[38] J. Balasch, B. Gierlichs, and I. Verbauwhede, “An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus,” in *FDTC*. IEEE Computer Society, 2011, pp. 105–114.

[39] T. Korak and M. Hoefler, “On the effects of clock and power supply tampering on two microcontroller platforms,” in *FDTC*. IEEE Computer Society, 2014, pp. 8–17.

[40] S. Endo, N. Homma, Y. Hayashi, J. Takahashi, H. Fuji, and T. Aoki, “A multiple-fault injection attack by adaptive timing control under black-box conditions and a countermeasure,” in *COSADE*, ser. Lecture Notes in Computer Science, vol. 8622. Springer, 2014, pp. 214–228.

[41] B. Yuce, N. F. Ghalaty, and P. Schaumont, “Improving fault attacks on embedded software using RISC pipeline characterization,” in *FDTC*. IEEE Computer Society, 2015, pp. 97–108.

[42] B. Yuce, N. F. Ghalaty, H. Santapuri, C. Deshpande, C. Patrick, and P. Schaumont, “Software fault resistance is futile: Effective single-glitch attacks,” in *FDTC*. IEEE Computer Society, 2016, pp. 47–58.

[43] J. Schmidt and C. Herbst, “A practical fault attack on square and multiply,” in *FDTC*. IEEE Computer Society, 2008, pp. 53–58.

[44] J. Breier, D. Jap, and C. Chen, “Laser profiling for the back-side fault attacks: With a practical laser skip instruction attack on AES,” in *CPSS@ASIACSS*. ACM, 2015, pp. 99–103.

[45] K. Fukushima, R. Xu, S. Kiyomoto, and N. Homma, “Fault injection attack on salsa20 and chacha and a lightweight countermeasure,” in *TrustCom/BigDataSE/ICESS*. IEEE Computer Society, 2017, pp. 1032–1037.

[46] P. Pessl and L. Prokop, “Fault attacks on cca-secure lattice kems,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 2, pp. 37–60, 2021.

[47] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, “Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures,” *Proc. IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[48] A. Menu, J.-M. Dutertre, O. Potin, J.-B. Rigaud, and J.-L. Danger, “Experimental analysis of the electromagnetic instruction skip fault model,” in *2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2020, pp. 1–7.

[49] C. Harpes, G. G. Kramer, and J. L. Massey, “A generalization of linear cryptanalysis and the applicability of matsui’s piling-up lemma,” in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 921. Springer, 1995, pp. 24–38.

[50] S. Patranabis and D. Mukhopadhyay, *Idempotent Instructions to Counter Fault Analysis Attacks*. Cham: Springer International Publishing, 2019, pp. 195–208.

[51] C. Herbst, E. Oswald, and S. Mangard, “An AES smart card implementation resistant to power analysis attacks,” in *ACNS*, ser. Lecture Notes in Computer Science, vol. 3989, 2006, pp. 239–252.

[52] J. Daemen, C. Dobraunig, M. Eichlseder, H. Groß, F. Mendel, and R. Primas, “Protecting against statistical ineffective fault attacks,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 508–543, 2020.