Linked Fault Analysis

Ali Asghar Beigizad[1], Hadi Soleimany[1], Sara Zarei[1] and Hamed Ramzanipour[2]

[1] Cyber Research Center, Shahid Beheshti University, Tehran, Iran
beigizad@yahoo.com, h_soleimany@sbu.ac.ir, sarazareei.94@gmail.com
[2] Shahid Rajaee Teacher Training University

Abstract.
Numerous fault models have been developed, each with distinct characteristics and effects. These models should be evaluated in light of their costs, repeatability, and practicability. Moreover, there must be effective ways to use the injected fault to retrieve the secret key, especially if there are some countermeasures in the implementation. In this paper, we introduce a new fault analysis technique called "linked fault analysis" (LFA), which can be viewed as a more powerful version of well-known fault attacks against implementations of symmetric primitives in various circumstances, especially software implementations. For known fault analyses, the bias over the faulty value or the relationship between the correct value and the faulty one, both produced by the fault injection serve as the foundations for the fault model. 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: $u' = l(v)$. We show that LFA lets the attacker perform fault attacks without the input control, with much fewer data than previously introduced fault attacks in the same class. Also, we show two approaches, called LDFA and LIFA, that show how LFA can be utilized in the presence or absence of typical redundant-based countermeasures. Finally, we demonstrate that LFA is still effective, but under specific circumstances, even when masking protections are in place. We performed our attacks against the public implementation of AES in ATMEGA328p to show how LFA works in the real world. The practical results and simulations validate our theoretical models as well.

Keywords: Fault Analysis · Linked Fault

1 Introduction

Fault attacks are a type of physical attack in which the attacker deliberately induces a fault in the targeted device with the aim of obtaining information about its secret key. This is accomplished by watching how the device responds to the fault. According to the duration of the fault, it is possible to classify induced faults into three categories. The vast majority of proposed fault attacks are based on transient faults, which only temporarily affect the system. This form of fault attack was first introduced by Boneh et al. with its application to RSA [BDL97]. The second type is known as "permanent fault", where the faults' effects on the target device are permanent for the rest of its lifetime. Persistent faults are the third form of fault. They persist but can be removed by resetting the target device. Schmidt first introduced the basic concept [SHP09], which was then taken by Zhang et al., who offered a more specialized framework [ZLZ+18]. Although his framework has garnered a lot of interest recently [PZRB19, GPT19, CB19, SBH+21, ESP20, CGR20, ZZJ+20, BSR+22], it is outside the purview of our study. In this paper, we will concentrate on transient faults, the second type.

The great majority of research on transient fault attacks against symmetric primitives focuses on the ability of this category of attacks to transform an intermediate value into a faulty one. The attacker injects a fault to value u to turn it into the faulty value u'. Most of the time, it is assumed that the obtained faulty value and its corresponding correct value have some extractable relationships. One can mention fault models like stuck-at, bit-flips, random fault, random-AND fault, biased fault, etc., where the referred interconnection is distinct in each case. In this work, we acquire an unprecedented perspective to shine light on an out-of-sight sort of relation and propose a novel fault-based attack dubbed "linked fault analysis"(LFA) based on it. LFA can be interpreted as a more powerful variation of several well-known fault attacks against software implementations of cryptographic primitives in various scenarios. While in a traditional fault attack, the fault model is defined based on the relation between the correct value and the faulty one, the LFA leverages a model in which 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 would be the change of the u value to faulty value u' according to the v value. We entitle this event as a linked change and take its advantage toward our intended novel analysis.

1.1 Overview of Prior Transient Fault Attacks

Classifications of fault attacks appear to be rational starting points to avoid getting lost between the various and varied fault attacks that have been proposed thus far and their attributes and application. Therefore, adhering to conventions, we, too, opt for this entryway to open our new outlook on this world and determine where our newly launched attack will land. However, before discussing different existing categorizations, we must first answer a critical question: Which criterion yields a more precise or thorough classification? Fortunately or not, this is not a single-answer question. Numerous variables can play a major or minor role in certain applications or settings. Different fault attack tactics include different demands and assumptions, each advantageous and appropriate in a particular context. The quantity of faulty and correct ciphertexts required to carry out a successful attack is, for example, a typical criterion used in the literate for comparing fault attacks. Though this is quite an essential attribute, especially when the number of queries is limited (for instance, due to the fresh-key policy), in practical situations, the fewer ciphertexts may be of minor priority, and other factors gain heavier consideration. Consider, for instance, the attacker's ability to encrypt the same input several times or performing the attack while encountering the underlying countermeasures. Our surveys indicate that considering these last two factors simultaneously, will cover a good number of the most famous introduced attacks in the field. It can also pinpoint the precise location of *LFA*. Thus, in what follows, we attempt to provide more information about these two benchmarks and classify well-known existing fault attacks based on them.

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. In some of these attacks, this fixed input is presumed to be chosen by the attacker, although in others, this is not a requirement. In each case, the underlying assumption is that the attacker has access to the input and is continually asking for the same plaintext to be encrypted.

Bypassing vs. losing to countermeasures In the presence of typical redundant-based countermeasures (infection-based [GST12, TBM14, FCL+20] or detection-based [BCN+06]), the adversary is unable to detect defective ciphertexts. Hence, it would not be possible for him/her to execute attacks that rely on the faulty ciphertext. However, there are faults that do not alter any intermediate value and are therefore invisible to embedded countermeasure systems. This phenomenon is known as an "ineffective fault". Some attacks are known to have the ability to circumvent standard countermeasures because they exclusively employ such kinds of events.

Table 1: Classifications of Prior Transient Fault Attacks on Block Ciphers

		Input's Control	
		✓	✗
Countermeasure Bypass	✗	DFA, IDFA, DFIA, CFA	SFA, LDFA*
	✓	FTA, FSA, SEFA	IFA, SIFA, LIFA**

*,** Linked-DFA and Linked-IFA, proposed in this research (**??**).

Table 1 represents the four subcategories created by the two abovementioned criteria, along with a list of well-known attacks that fall within each group. The first group contains Differential Fault Analysis (DFA) [BS97], one of the most prominent and leading 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 [SMC09, AM12, AM13, AMT13], authenticated encryption schemes [RCC+16] and even stream ciphers [DCAM15]. Extensive research has been conducted on DFA, and multiple generalized and automated frameworks have been developed to identify vulnerabilities against this attack methodology [SMD18, BHL18, HBZL19]. Similar to the classical type of differential cryptanalysis, some other fault attacks are diverging from the main attack. The Impossible Differential Fault Attack (IDFA) is a differential-based fault attack based on an impossible characteristic [DFL11]. Algebraic Fault Attack (AFA) is similar to DFA in that analyzing both faulty and correct outputs [ZGZ+13, ZZG+13, ZGZ+16]. 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) [GYTS14] 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) [BK06] is another fault attack belonging to the first group of attacks. It applies when an attacker can encrypt two related inputs, injects a fault into one of the computations, and then exploits a collision that may occur (usually in the outputs). Therefore, redundant-based countermeasures can prevent these attacks, and the attacker must also control the input to perform these attacks.

Then we have the second group, attacks that require input control but are able to bypass countermeasures. Fault Sensitivity Analysis (FSA) [LSG+10] needs to encrypt the same plaintext during the profiling phase in order 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) [SBR+20] is another promising technique that can circumvent redundant-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 SFA and FTA can circumvent redundant-based countermeasures, but in practice, the attacker must control the input to repeat the encryption process of the same input. The last noteworthy member of this group is the newly introduced Statistical Effective Fault Analysis (SEFA) [VZB+22]. A dual for the most famous SIFA attack into which we will concentrate on the fourth group and also in the Section 2.

The last two groups comprise attacks not requiring input control. Statistical Fault Analysis (SFA) [FJLT13] eliminates the requirement of repeated encryptions of a particular plaintext by means of employing statistics, rather than the rigid and frequently impracticable regulating conditions of the first two groups. On the other hand, it demands faulty ciphertexts; therefore, it can be thwarted by redundant-based countermeasures.

Last but not least, the fourth group includes attacks that can get over redundant-based

countermeasures while not requiring input control. These attacks exclusively target cases in which the error has no effect on the computation. Safe Error Attack (SEA) [YJ00] is typically relevant to public-key cryptographic systems. Ineffective Fault Analysis (IFA) [Cla07] and its statistical counterpart (SIFA) [DEK+18] 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 [SBHS15, SHS16].

1.2 Our Contributions

In certain applications, it is not possible to encrypt (or sign) the same message more than once. A prominent example of such situations 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 1's first two groupings. 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). Despite the fact that it is sometimes assumed that the underlying block cipher is utilized in ECB mode, alternative modes like CTR, OFB, 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 value of the counter [BBB+18].

On the other hand, as was indicated in the preceding subsection, techniques that do not necessitate input control by the attacker usually rely on specific statistical properties (groups 3, and 4 in Table 1). As a result, compared to attacks with input control, the key-recovery strategy frequently requires more samples to be able to statistically separate the correct key from incorrect ones. It was also mentioned that the other side of the spectrum is DFA, which uses a lot fewer data than SFA at the cost of involving input control. Both tiny amounts of data and control-free scenarios can be important in different real-world situations, however, there is no free launch. None of them are possible for free in the DFA, SFA, and their other family members. We fill this critical gap in this paper by proposing a new fault technique that allows performing fault attacks without the need for input control while using significantly less data than previously presented fault attacks in groups 3 and 4. Our novel method is able to work both in the presence or absence of countermeasures.

In the absence of countermeasures: LDFA

For DFA and CFA to work, associations between two intermediate values—one from the correct computation process and the other from the faulty computation—must exist. Despite the limited availability of data, this attribute enables the attacker to utilize these relationships 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 strategic difference: our approach is based on correlations between the intermediate values of only faulty computations. In other words, we employ a relation attribute that is intrinsic to the faulty system and does not depend on access to the correct computation. Consequently, our proposed method can be viewed within the framework that 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).

In the presence of countermeasures: LIFA

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 focusing on ineffective ciphertexts. These ciphertexts' strength lies in their ability to circumvent redundant-based defenses. We demonstrate that our method is also applicable to ineffective ciphertexts. Also, our method gives the attacker the ability to find missed faults under certain conditions, which is important in real-world applications because missed faults can influence the performance of SIFA. We also demonstrate that LFA might be applicable in the presence of first-order masking and detection-based countermeasures.

1.3 Outline

The remaining sections of this paper are organized as follows. Section 2 gives the necessary background for this paper. In Section 3, we describe a novel technique known as LFA and analyze its application in various scenarios. In Section 4, we investigate the effects of conventional countermeasures on LFA. In Section 5, we provide our simulation and experimental results. Finally, we conclude in Section 6.

2 Preliminaries

2.1 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 cipher 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 $\mathcal{L}$ (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 S-boxes. 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 $1 \leq r \leq R$. Given that $\mathcal{L}$ is linear, it might be dropped or changed with a less complex version in the final round. Subkeys can be combined into states via a group operation, such as addition or XOR.

We concentrate on AES because it is the most frequently utilized cipher in real-world applications, even though our methods apply to any block cipher. AES is a 128-bit block cipher with a key size of 128, 192, or 256 bits with a substitution-permutation network (SPN) structure. AES 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. x_r represents the state input of round r, while y_r, z_r, and w_r represent the states after applying the SB, SR, and MC operations of round r, respectively. Since the output of the $r-$th round is the input of the $(r+1)-$th round, we represent it with the notation x_{r+1}.

There are various ways to implement AES. In addition to conventional implementation, the combination of SubByte, ShiftRow, and MixColumn operations can be precomputed and saved as 8×32 lookup tables T_0, T_1, T_2, and T_3 as shown in Equation (1).

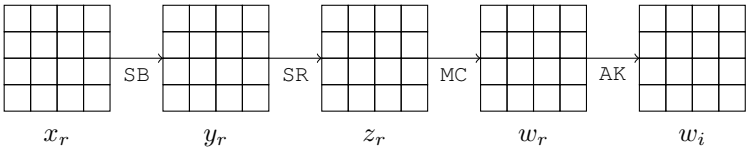

Figure 1: AES round function

$$T_0 = \begin{bmatrix} 02.S(z) \\ S(z) \\ S(z) \\ 03.S(z) \end{bmatrix}, T_1 = \begin{bmatrix} 03.S(z) \\ 02.S(z) \\ S(z) \\ S(z) \end{bmatrix}, T_2 = \begin{bmatrix} S(z) \\ 03.S(z) \\ 02.S(z) \\ S(z) \end{bmatrix}, T_3 = \begin{bmatrix} S(z) \\ S(z) \\ 03.S(z) \\ 02.S(z) \end{bmatrix} \tag{1}$$

Since the last round does not include MixColumn, the output of T-tables or another lookup table such as T_4 can be utilized to implement the final round. Since timing attacks can be applied to the T-table implementation, it should be used in practice cautiously.

2.2 DFA

The most widely used technique for fault analysis is differential fault analysis, which was first introduced in [BS97]. The assumption is that the attacker can inject a difference at a certain intermediate value in a particular round. In other words, the attacker is aware of the difference between the intermediate value's faulty and correct values. 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 in order 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. This method sometimes makes use of the precise relationship between the input difference and output difference of S-boxes.

2.3 SIFA

The Statistical Fault Attack (SFA) employs a biased distribution across an intermediate value. The attacker decrypts provided faulty ciphertexts with the key candidates across 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. Different statistical tests can be applied given N samples, depending on whether the attacker is knowledgeable or oblivious of the distribution p of the incorrect value.

The log-likelihood ratio (LLR) statistic can be applied as stated in Equation (2) if the distribution of the faulty value p is known. θ indicates the uniform distribution in the equation.

$$\mathrm{LLR}(k) = N \sum_{x \in \mathcal{X}} \hat{p}_k(x) \cdot \log_2 \frac{p(x)}{\theta(x)} \,. \tag{2}$$

If the faulty distribution is biased, but unknown to the attacker, Squared Euclidean Imbalance (SEI) can be used as shown in Equation (3).

$$\mathrm{SEI}(k) = \sum_{x \in \mathcal{X}} (\hat{p}_k(x) - \theta(x))^2 \,. \tag{3}$$

As a result, the attacker either uses $\mathrm{LLR}(k)$ or $\mathrm{SEI}(k)$ to rank the key candidates.

SIFA is simultaneously enabled by SFA and IFA methods. It utilizes the biased distribution of the intermediate value over ineffective faults, and is able to evade detection- and infection-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 phenomenons.

2.4 Limitations of Instruction Skip Attacks in Block ciphers

Skipping the execution of a single or multiple microprocessor instructions is a highly effective way of modifying processed values. A variety of laser pulses [BCN+06, TK10], electromagnetic pulses [DDR+12, MDH+13a, DDRT12], clock glitches [BGV11, KH14, EHH+14, YGS15, YGS+16], and power glitches [SH08] have been suggested as methods for skipping one or more instructions on microcontrollers. The majority of these attacks on symmetric-key primitives need input control, and more significantly, they cannot circumvent common countermeasures by nature. The following is a brief discussion of instruction-skipping attacks against block ciphers.

The loops and functions can be easily disrupted by a fault injection; hence, it is feasible to bypass a significant portion of the encryption and perform standard cryptanalysis, such as differential cryptanalysis, on the output [MDH+13b]. It is doubtful that a reduced-round cipher will yield ineffective output, preventing this type of attack from evading redundant-based countermeasures.

Another possible attack is to bypass the addition instruction of the final round key, which is added to the state via an exclusive-or operation [BGV11, BJC15]. The last round subkey can be easily 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 [FXKH17]. Pessl and Prokop recently demonstrated at CHES 2021 that addition instruction skip can be used to attack lattice-based KEMs [PP21]. Despite the public key schemes [SH08, BBKN12], this approach is not typically relevant to symmetric primitives in the presence of typical redundant-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.

Another method is to skip the equality check by skipping the conditional branching following the redundant computation [EHH+14]. One should note that certain faults, such as those 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 faulting targets with additional protections.

3 Linked Fault Analysis (LFA)

3.1 Fault Model

We target software-based implementations on microcontrollers and introduce a new fault analysis that takes advantage of instruction skipping, which is a common way to cause faults on these devices. As we will demonstrate in the following sections, our method is effective and practical, can be conducted with inexpensive tools, and can circumvent countermeasures based on redundancy. In our attack model, we assume that the attacker can control the intensity and duration of the fault. Besides, we presume, similar to the majority of proposed fault attacks, that the attacker is aware of the timing. It should be noted that in most circumstances, it is assumed that the fault's location (time) is known to the attacker, but this can be established by trial and error. Although later in Section 3.4.2, we show that the assumption of precise knowledge about timing can be relaxed under some circumstances. We assume that the target is a word-oriented block cipher and that

the nonlinear layer is implemented using one or more pre-computed look-up tables that are known to the attacker and that each table is called several times throughout round function execution.

We use the instruction skip technique 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. We demonstrate that, under certain conditions, if v and u are processed sequentially, the value of faulty u (i.e., u') becomes identical to the value of v. This fault can be induced using cheap instruments using well-timed power spikes or clock glitches. The links created across intermediate values may result in linked words in the ciphertexts, which an attacker can benefit from to obtain information about the secret key. The following describes the working mechanism of instruction skip to lead to linked fault.

The nonlinear layer of block ciphers typically consists of S-boxes. The S-box (or S-boxes) are progressively invoked with different values to carry out the nonlinear layer. Alternatives include utilizing larger, pre-computed look-up tables for both nonlinear and linear layers. The vast majority of block ciphers include the ability to load the output of such look-up tables, which can be exploited to cause a linked fault in the called function's input or output. In other words, the skip instruction can be used with these load instructions to target either the instruction that selects the address of the data or the instruction that settles the values of these data into RAM or flash memory.

For the time being, we assume that the instruction skip occurs perfectly. We will discuss the effects of missed or unwanted faults later in the last subsection and explain how to deal with them.

3.2 Linked Differential Fault Attack (LDFA)

In the absence of redundant-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 process of 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 this same idea, but instead, our method is based on relations between the intermediate values of the faulty computation itself. Hence, our method does not require input control. For the sake of clarity, we will describe our attack against AES first, and then we will describe a general method to recover the secret key from any cipher.

Assume that the byte $x[j]$ becomes equal to $x[i]$ or $y[j]$ becomes equal to $S(x[i])$ when an instruction skip is occurred during the execution of Sboxes. This fault causes two bytes (with unknown values), $z'_{10}[SR[i]]$ and $z'_{10}[SR[j]]$, to become equal after the ShiftRow operation. The corresponding bytes of ciphertexts $C'[SR[i]]$ and $C'[SR[i]]$ are not necessarily equal after adding the subkey sk_{10}, but they can be used to retrieve information about the subkey. Figure 2 demonstrates the propagation of a linked fault for $i = 1$ and $j = 5$. Given a faulty ciphertext C', the attacker can guess the last round's two bytes, $sk_{10}[SR[i]]$ $sk_{10}[SR[j]]$, and partially decrypt the ciphertext C' to compute intermediate values $z'_{10}[SR[i]]$ and $z'_{10}[SR[j]]$ for the guessed key. If $z'_{10}[SR[i]] \neq z'_{10}[SR[j]]$, the guessed key is wrong; otherwise, it could be a possible candidate for the correct key. The probability that a wrong key survives is 2^{-8}. Since there are 2^{16} possible values for $sk_{10}[SR[i]]$ $sk_{10}[SR[j]]$, only $2^{16} \cdot 2^{-8} = 2^8$ candidates remain. Hence, the number of remaining key candidates is reduced by a factor of 2^8 by evaluating the faulty ciphertext of each linked fault. In other words, the attacker can retrieve 8-bit information about the last round key by injecting only one fault and utilizing only one faulty ciphertext. It should be noted that this attack is applicable to any table-based AES implementation. If byte $x[j]$ becomes equal to $x[i]$, the output of the round is changed correspondingly in the T-table implementation of AES. Given h distinct linked faults and faulty ciphertexts,

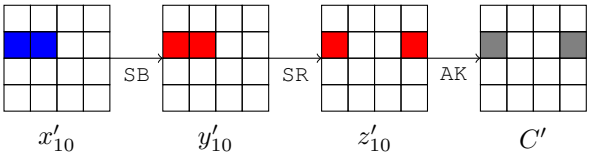

Figure 2: Last Round of AES: Similar-colored bytes are linked together.

the residual key entropy reduces from k bits to $(k - 8 \times h)$ bits. As previously noted in Section 3.1, the actual number of linked faults that an attacker is able to inject relies on the implementation and, more specifically, the number of lookup tables that are called throughout the round function's execution.

While we presented the attack for the AES cipher, it may be applied to any cipher by adopting a similar technique that has been utilized for key recovery in various differential fault attacks. 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, by guessing the value of these subkey bits, the intermediate values u' and v can be determined. 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 see if the relation $u' = l(v)$ holds. This technique is illustrated in Algorithm 1. The relation $u' = l(v)$ holds in all cases for the correct key, but with a probability of $2^{-m \cdot N}$ for the wrong key. Equation (4) can be used to compute the number of faulty ciphertexts needed to eliminate all the wrong key candidates.

$$\tau \cdot 2^{-m \cdot N} < 1 \Rightarrow 2^{\kappa - m \cdot N} \leq 1 \Rightarrow \kappa - m \cdot N \leq 0 \Rightarrow N \geq \frac{\kappa}{m} \tag{4}$$

Algorithm 1 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

```
for ℓ = 0 to (τ − 1) do
    for h = 1 to N do
        (u′, v) ← E^{-1}_{k_ℓ}(C′_h)        ▷ Partial decryption
        if u′ = l(v) then
            cnt[ℓ] = cnt[ℓ] + 1
        end if
    end for
end for
return argmax_ℓ(cnt[ℓ])
```

One should note that, in contrast to DFA, this method does not require the correct ciphertext. Hence, the attacker does not need to repeat the encryption process twice for a specific input value, which requires control over the input. We call this technique the Linked Differential Fault Attack (LDFA) because it leverages the difference between the bytes of ciphertext that are linked in some way. However, the benefit of LDFA over other approaches such as DFA and SFA is evident. LDFA uses linked words to exploit some advantages of known attacks (like how DFA only needs a small number of data and SFA doesn't need to control input) and overcome their limitations (like how DFA and CFA need to control input and SFA needs a lot of data).

Similar to attacks such as DFA and SFA, LDFA cannot circumvent redundant-based countermeasures. In the next part, we show that linked fault analysis is flexible enough that it can be used in applications that use conventional countermeasures.

3.3 Linked Ineffective Fault Attack (LIFA)

Some proposed attacks concentrate on ineffective faults. As they let the attacker circumvent several fault countermeasures, such as detection-based and infection-based countermeasures, these attacks have an obvious benefit. Injecting a stuck-at fault that is proposed in IFA is not a simple process, and it is not always practicable due to the need for costly equipment and specific knowledge about the target. SIFA utilizes an unbiased distribution of a targeted intermediate value over ineffective events. SIFA does not necessitate a complicated mechanism to inject the fault, but it usually cannot be performed with a few ineffective ciphertexts. To distinguish the correct key from a wrong key, sufficient ineffective ciphertexts must be collected in order to detect the desired bias in the distribution of the targeted intermediate value.

In contrast to IFA, LFA does not require complex equipment. LFA, unlike SIFA, does not rely on a static that cannot be distinguished by a small number of ciphertexts. Therefore, linked fault analysis appears to be a suitable technique when a redundant 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.

In the same manner as in the previous subsection, we begin by describing a linked fault in AES that causes the byte $x'[j]$ to become equal to $x[i]$ or $y'[j]$ to become equal to $S(x[i])$. The faulty ciphertext C' is unavailable to the attacker when redundant-based countermeasures are present. Nevertheless, if $y_{10}[i]$ is equal to $y_{10}[j]$, the fault has no effect on the processed data. In other words, if the relation $y_{10}[i] = y_{10}[j]$ 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^{-8}. This indicates that, on average, the attacker must repeat the process 2^8 times for each fault. After that, the attacker can retrieve 8-bit information about the key in a similar manner to that explained in the previous subsection by observing an ineffective event.

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 of the ciphers). Adopting LFA over ineffective events is therefore highly efficient, as one ineffective fault happens on average for every 2^m faulty computations. 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.

3.4 Utilizing a Link Between the Words of Faulty Ciphertext

As previously discussed, LFA has an intriguing feature that produces a link between the faulty value (u') and another value (v). The existence of such a link may be conveyed to the faulty ciphertext (C') in the sense that one portion of the faulty ciphertext (e.g. $C'[i]$) has a key-dependent link to another portion of the faulty ciphertext (e.g. $C'[j]$), although likely with a more complex relation (e.g. $C'[i] = f(C'[j], sk_R)$) holds for a known function f). This part demonstrates that this link can be utilized in two situations. First, we demonstrate that it is likely to be employed for a key-recovery attack that is faster. Then, we demonstrate that it assists the attacker in determining the location of the flaw if the attacker lacks detailed knowledge of the implementation. In order to demonstrate the advantage of a key-dependent link in a faulty ciphertext, we will analyze the LFA on the AES example from the preceding sections.

3.4.1 More Efficient Key Recovery

In previous subsections, the secret key was retrieved in a conventional manner. Peeling of the last round for a guessed key and checking a characteristic for the intermediate value(s) is a key-recovery method that dates back to the 1990s, prior to the introduction of fault

attacks. In this section, we demonstrate that it may be possible to retrieve the key more efficiently without requiring the partial decryption of (faulty) ciphertexts in a traditional manner.

For the sake of clarity, we will study the attack on the final round of AES to explain how one portion of the faulty ciphertext can be linked to another via a key-dependent relationship. As shown in Equation (5), it is straightforward to verify that the bytes of faulty ciphertexts $C'[SR[i]]$ and $C'[SR[j]]$ are linked based on the value $\Delta sk_{10}[i,j] \triangleq sk_{10}[SR[i]] \oplus sk_{10}[SR[j]]$.

$$C'[SR[i]] \oplus C'[SR[j]] = (z'_{10}[SR[i]] \oplus sk_{10}[SR[i]]) \oplus (z'_{10}[SR[j]] \oplus sk_{10}[SR[j]]) = \Delta sk_{10}[i,j] \tag{5}$$

Equation (5) states that the relation between the linked bytes of ciphertext depends on the value of the key and it is deterministic.

$$\Pr(C'[SR[i]] \oplus C'[SR[j]] = \alpha) = \begin{cases} 1 & \alpha = \Delta sk_{10}[i,j], \\ 0 & \alpha \neq \Delta sk_{10}[i,j]. \end{cases} \tag{6}$$

As illustrated in Algorithm 2, the relation given in Equation (6) allows an attacker to extract one byte of information about the final subkey with only one faulty ciphertext and without the need to perform costly key guessing.

It is worth mentioning that in the case of ineffective fault, $z_{10}[SR[i]]$ and $z_{10}[SR[j]]$ have a similar relationship (after the ShiftRow operation) and Algorithm 2 can be used similarly but over ineffective ciphertexts.

$$\Pr(C[SR[i]] \oplus C[SR[j]] = \alpha | \mathrm{i}) = \begin{cases} 1 & \alpha = \Delta sk_{10}[i,j], \\ 0 & \alpha \neq \Delta sk_{10}[i,j]. \end{cases} \tag{7}$$

Algorithm 2 Efficient key-recvoery attack on last round AES

Require: Faulty ciphertexts $C'_1, C'_2, \ldots, C'_N$ ▷ In case of LIFA, only ineffective ciphertexts are considered
Ensure: Correct value of $\Delta sk_{10}[i,j]$
1: **for** $h = 1$ to N **do**
2: $\quad \delta \leftarrow C'_h[SR[i]] \oplus C'_h[SR[j]]$
3: $\quad cnt[\delta] = cnt[\delta] + 1$
4: **end for**
5: **return** $argmax_\delta(cnt[\delta])$

3.4.2 Unknown Fault's Location

In this section, we illustrate that the assumption of precise knowledge of the loading order of intermediate values can be relaxed if some ciphertext words have a specified relationship similar to the preceding section's illustration. If we suppose that in general, an m-bit relation $C'[i] = f(C'[j], sk_R)$ holds, the attacker may be able to determine the values of i and j. The basic idea is to look at the relationship $C'[i] = f(C'[j], sk_R)$ for all possible i and j values across all key candidates. This process is illustrated in Algorithm 3. If we take two portions of ciphertexts that are not affected by the fault, this relation is expected to hold with a probability of 2^{-m} even for the correct key, whereas it always holds for a right guess of i and j for the correct key. Given N faulty ciphertexts and assuming that there are ℓ_1, ℓ_2, and τ candidates for i, j, and sk_R, respectively, the relation holds about $\ell_1 \cdot \ell_2 \cdot \tau \cdot 2^{-m}$ times for an incorrect guess. Equation (8) can be used to compute the number of faulty ciphertexts required to eliminate all wrong guesses. For example, the last round attack on AES has respectively $\ell_1 = 16$, $\ell_2 = 16$, and $\tau = 256$ possibilities for i, j, and sk_R (here it refers to $\Delta sk_{10}[i,j]$), while $m = 8$-bit filtering is provided. Given N=3

faulty ciphertexts, the attacker is able to determine not only the value of $\Delta sk_{10}[i,j]$, but also the location of the fault and how it is linked to the other word. 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 ciphertexts.

$$\ell_1 \cdot \ell_2 \cdot \tau 2^{-m \cdot N} < 1 \Rightarrow 2^{log(\tau+\ell_1+\ell_2)-m \cdot N} \leq 1 \Rightarrow \log(\tau+\ell_1+\ell_2) - m \cdot N \leq 0 \Rightarrow N \geq \frac{\log(\tau+\ell_1+\ell_2)}{m} \tag{8}$$

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 correct value of $\Delta sk_{10}[i,j]$

```
for h = 1 to N do
    for i = 0 to 15 do
        for j = i + 1 to 15 do
            δ ← C'_h[SR[i]] ⊕ C'_h[SR[j]]
            cnt_{i,j}[δ] = cnt_{i,j}[δ] + 1
        end for
    end for
end for
return argmax_{i,j,δ}(cnt_{i,j}[δ])
```

3.5 Affect of Missed and Unwanted Faults

Until now, we have assumed that fault injection is perfect, which is rarely the case in practice. It is possible that the fault will not be injected as desired due to a number of 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.

3.5.1 Unwanted Faults

In certain instances, the instruction skip may occur in an undesirable location. An "unwanted fault" describes a situation in which the fault is not injected at the desired time (or location). Injecting a fault at the incorrect time can result in an undesirable fault. This can occur due to either the unintentional noise that occurs naturally in practical experiments or the intentional noise that results from hiding countermeasures. Hiding countermeasures, such as shuffling and dummy operations, are intended to make fault injection into a specific place more complex by restricting the attacker's knowledge of the precise moment of instruction execution and processed data. Dummy operations incur significant additional expenses, and their impact against attacks like SEFA is limited. However, shuffling appears to be a successful, low-overhead countermeasure against the vast majority of fault attacks.

3.5.2 Missed Faults

There may be instances in which no instruction skip occurs. These instances, which cannot be distinguished from ineffective events, are referred to as "missed faults". Even with robust equipment and perfect setup, it is possible to have missed faults. Due to the fact that missed faults might be interpreted as ineffective faults, they have a substantial effect on fault attacks that employ ineffective events. This fact stimulated the development of novel attacks [PP21, VZB+22] that utilize correct ciphertexts that correspond to effective faults, called non-faulty effective ciphertexts. However, an attacker must perform the encryption twice to obtain non-faulty effective ciphertexts: once to compute the faulty

encryption to determine if it is effective, and again to obtain the corresponding correct ciphertext.

3.5.3 LFA in the Presence of Undesired Cases

Most fault attacks, if not all, 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.

In this part, we solely discuss situations where extra countermeasures, such as shuffling, are not employed. 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. Since no fault occurs or it 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}. The relation $u' = l(v)$ is not deterministic, in contrast to the situation discussed in Section 3.2 and Section 3.3. However, as shown in Equation (9), $u' = l(v)$ satisfies with different probabilities for the correct key and a wrong key.

$$\Pr(u' = l(v)) = \begin{cases} (1 - (\Pi_m + \Pi_u)) + 2^{-m} \cdot (\Pi_m + \Pi_u) & \text{Correct key} \\ 2^{-m} & \text{Wrong key.} \end{cases} \tag{9}$$

Hence, the attack can apply Algorithm 1 similarly, but he requires more data. Similar discussion holds for the key recovery attack presented in Section 3.4.1 in which we assume a key-dependent relation like $C'[i] = f(C'[j], sk_R))$ holds for two parts of faulty ciphertext where f is a known function. For instance, we can again consider the attack on the last round of AES. Since no fault occurs or it occurs in an incorrect location for other $N \times (\Pi_m + \Pi_u)$ outputs, the value of $C[SR[i]] + C[SR[j]]$ is a random 8-bit value. Hence, the distribution is not uniform as it is indicated in Equation (10).

$$\Pr(C[SR[i]] \oplus C[SR[j]] = \alpha) = \begin{cases} (1 - (\Pi_m + \Pi_u)) + 2^{-8} \cdot (\Pi_m + \Pi_u) & \alpha = \Delta sk_{10}[i,j], \\ 2^{-8} & \alpha \neq \Delta sk_{10}[i,j]. \end{cases} \tag{10}$$

Consequently, Algorithm 2 can be applied similarly here, but the required data increases.

4 Countermeasures and Further Discussions

4.1 Shuffling

Shuffling is a simple and effective countermeasure against fault attacks. The main principle behind shuffling is to randomize the order of operations that can be executed in any order. Sboxes can be executed in any order when it comes to block ciphers, for example. In the presence of shuffling, an attacker cannot predict the order of the calling look-up tables because it is randomized in each execution. As a result, injecting a fault at a certain time does not necessarily affect a fixed intermediate value. This is why the performance of the majority of fault attacks is severely impacted by shuffling. Even SEFA, which is substantially less susceptible to noise than SIFA, is shown to be equally affected by the shuffling.

As explained in Section 3.4.2, in the key-recovery procedure of LFA, the attacker need not necessarily know which byte has been altered and how it is linked to other bytes. Even in this instance, it is presumed that the fault location and its linked word have been fixed (but unknown). In the key-recovery procedure of LFA, the attacker must be aware of which byte has been altered and how it is linked to other bytes. The input state contains L words that can be processed in $L(L-1)$ distinct ways. To generate a linked fault like $u' = l(v)$, the values of u and v must be processed sequentially and at the time duration when the attacker injects the fault (see Section 3.1). However, this is not the case when shuffling is present. 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 during the injection of the fault. 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 probability $1 - \frac{2}{L(L-1)}$. The relation $u' = l(v)$ may be randomly satisfied with a probability of 2^{-m} in these cases. As a result, as shown in Equation (11), $u' = l(v)$ satisfies with different probabilities for the correct key and a wrong key.

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

Equation (11) indicates that the attacker can use Algorithm 1 to retrieve the key, but he needs more data. Similarly, Algorithm 2 can be used if one portion of the faulty ciphertext is linked to another via a key-dependent relationship. The number of wrong keys and the probability $\Pr(u' = l(v))$ for the correct and wrong key determine the amount of data N required to extract the correct key. Using Algorithm 1 of [BGN12], the required data can be computed. For example, to obtain the correct key in the LFA on the final round of AES with probablity 95%, an attacker needs approximately 2^{14} faulty ciphertexts. 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 demonstrates that shuffling is far less effective against LFA than statistical fault analysis techniques such as SFA, SIFA, and SEFA. It is important to note that even SEFA, which is less affected by missed faults than SIFA, can be prevented by the shuffling method [VZB+22]. This advantage stems from the fact that LFA employs a deterministic relation, which is less affected by the noise introduced by shuffling.

4.2 Repetition of instructions

The linked fault presented in this research can be generated in practice by skipping instructions. Various instruction-level countermeasures for defending the implementation against fault attacks have been researched and presented in prior studies. Idempotent instructions (such as the move instruction) can be implemented multiple times without affecting the output [PM19]. This countermeasure should be utilized with caution, as there are some security concerns regarding the protection offered by multiple copies of the instructions. While it is sometimes assumed that these countermeasures can only be compromised by injecting several faults, it is demonstrated that their security can be compromised by injecting a single clock glitch with a low-cost tool [YGS+16]. Despite these issues, it is undeniable that instruction-level countermeasures such as the repeating of idempotent instructions can make the application of LFA substantially more difficult, particularly when combined with other countermeasures such as shuffling.

4.3 Masking

Side-channel attacks pose a severe threat to devices such as smart cards and IoT devices. Masking is the most widely used protection against side-channel attacks in the imple-

mentation of cryptographic algorithms. Boolean masking is the most common method for splitting a sensitive intermediate variable x into multiple shares x_i where $0 \leq i \leq d$ so that $x = \sum_{i=0}^{d} x_i$ holds. Similarly, a primary function f is decomposed into so-called component functions $f_j()$ so that the original function can be evaluated using shared variables: $f = \sum_{j=0}^{s}$. The majority of fault attacks target intermediate values, which are not processed when masking is employed. Hence, masking can make it difficult to apply the majority of fault attacks in practice, although it is not designed to prevent fault attacks. One exception is a variation of SIFA that has been demonstrated to be applicable to implementations protected by both masking and detection-based countermeasures [DEG+18]. Observing an ineffective event after injecting a fault into one of the shares of an intermediate variable may result in the disclosure of the unmasked value of another variable. This technique is sometimes referred to as SIFA2 because it differs slightly from the initial idea. Daemen et al. proposed a sophisticated SIFA-resistance masking at CHES 2020 [DDE+20] as a response to this challenge. A single fault on a share of an intermediate variable in this construction is either effective and detectable by a countermeasure based on redundancy, or ineffective but dependent on an incomplete set of shares for a single variable.

We demonstrate in this part that LIFA may be applicable in the presence of first-order masking and detection-based countermeasures.

4.3.1 Precomputed masked Sbox

A straightforward masking implementation of a Sbox in software is possible [HOM06] 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 as well as the amount of memory required to store the tables also increases. 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 3.3 when using the first-order masking implementation of AES provided in [HOM06], .

4.3.2 SIFA-Protected Sbox

Algorithm 4 SIFA-protected masked implementation of $\chi 3$

Require: $(a_0, a_1, b_0, b_1, c_0, c_1)$
Ensure: $(r_0, r_1, s_0, s_1, t_0, t_1)$

$T_0 \leftarrow b_0 c_1$; $T_2 \leftarrow a_1 b_1$
$T_1 \leftarrow b_0 c_0$; $T_3 \leftarrow a_1 b_0$
$T_0 \leftarrow T_0 \oplus a_0$; $T_2 \leftarrow T_2 c_1$
$r_0 \leftarrow T_0 \oplus T_1$; $t_1 \leftarrow T_2 T_3$

$T_0 \leftarrow c_0 a_1$; $T_2 \leftarrow b_1 c_1$
$T_1 \leftarrow c_0 a_0$; $T_3 \leftarrow b_1 c_0$
$T_0 \leftarrow T_0 \oplus b_0$; $T_2 \leftarrow T_2 a_1$
$r_0 \leftarrow T_0 \oplus T_1$; $t_1 \leftarrow T_2 T_3$

$T_0 \leftarrow a_0 b_1$; $T_2 \leftarrow c_1 a_1$
$T_1 \leftarrow a_0 b_0$; $T_3 \leftarrow c_1 a_0$
$T_0 \leftarrow T_0 \oplus c_0$; $T_2 \leftarrow T_2 b_1$
$r_0 \leftarrow T_0 \oplus T_1$; $t_1 \leftarrow T_2 T_3$

SIFA is proven to be applicable even when masking is present. A SIFA-protected

masking that can be used in both software and hardware implementations was proposed by [DDE+20]. Algorithm 4 illustrates a SIFA-protected masked implementation of *chi*3, 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. 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 true 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 S-box. 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 [DDE+20], 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.

4.4 Automated Tools for Finding LFA vulnerabilities

Several automated tools for analyzing the fault attack vulnerabilities of software or hardware implementations have been presented. Because LFA does not rely on the relationship between faulty and correct computations of encryption, techniques like the one described in [HBZL19], that mean to detect differential-based fault attacks are incapable of identifying linked faults. In addition, LFA is also utilizable in cases of ineffective events. Some tools have been proposed, which employ sophisticated techniques to find weaknesses in these situations. However, the majority of them are designed intending hardware implementations. Below is a discussion of why even current cutting-edge techniques cannot discover LFA vulnerabilities.

VerFI [AWMN20], is one of the automated fault detection systems, that evaluates countermeasure strategies for determining the location, cycle, and type of injected faults. This tool analyzes DFA, AFA, SEA, and SIFA attacks in 4 fault models, including biased and uniform faults in implementations with design flaws; a univariant fault in one clock cycle; and multivariant faults in several clock cycles, and stuck at 0.1 and bit-flip. This tool's novelty is "fault Coverage", which is dependent on three factors: detected fault, undetected fault, and ineffective fault, the latter of which is essential for the SIFA scenario evaluation. On the other hand, LFA depends on two (linked) intermediate values of the same state. Note that despite the fact that it employs two values, it uses a type of single-fault attack. According to the predetermined VerFI models, both values must be submitted as detected faults for this fault model to be evaluated. This will drastically lower the overall fault coverage.

FIVER [RSS+21] is an additional verification tool for fault injection countermeasures. In addition to addressing some of VerFI's shortcomings, this technique provides a model for analyzing protection implementations known as Binary Decision Diagrams (BDDs). The fault model it uses is no longer restricted to some fixed cases, in return, the modeling of each injected fault depends on the attack scenario and is in some ways generalized. According to BDD's optimizations, the fault model represented by the FIVER tool uses the parameters n, t, and l to describe the total number of injected faults, the fault type (bit flip or trapped at 0.1), and the fault location. In order to provide a more comprehensive picture, the developers of FIVER have included a fourth parameter for modifying the

status of boolean functions. This parameter indicates that after a fault is injected, each logic gate transforms into another kind. The primary objective of verification in the FIVER tool is to evaluate as rapidly as possible the number of possible combinations resulting from the insertion of faults into one of the provided models. In none of the situations analyzed by the FIVER, there is no coverage for the definition of the LFA fault model. In other words, even if the dependence between graphs is implemented to find the best possible combinations or if all possible states are analyzed to change the output state of a logic gate after fault injection in each model, the FIVER tool will still examine one of the two values of the LFA scenario.

5 Experiments

In this section, the feasibility of LFA on the final round of AES when run on a microcontroller is investigated. We provide the results of a practical LFA performed by injecting a single clock glitch, which is non-invasive and requires a low-cost instrument.

5.1 Experimental Setup

We come up with the realization of the linked fault by means of a basic yet typical setup. We set our target system's (μController's) clock to be fed by an external clock. The clock is then built by utilizing a Field-Programmable Gate Array (FPGA). By doing so, 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). Take an ATMEGA328p case, we have an increase from 16MHz 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. In fact, the FPGA's high synchronicity with the μController allows us, as attackers, 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 (supposed be the 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* value, respectively. In what follows, we describe and report the results of our first experiment.

We consider an 8-bit AVR μController (ATMEGA328p) on which the AES algorithm is running. We target the start of the last AES round. The AES implementation is carried out through a conventional C code with standard libraries in which is a publicly available AES implementation[1]. Since our attack involves only a single instruction skip, the performance of the attack will not be significantly affected by the usage of assembly-optimized code.

5.2 Application of LDFA and LIFA

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 successful linked faults in nearly 99.1% of them. Besides, we observed ineffective faults, missed faults, and unwanted faults in 108, 42, and 54 of the experiments, respectively. We repeated the fault injection for various values and discovered that the fault injection time may be modified to create linked faults that are similar and have a high likelihood of succeeding. Our trials' effective rate ($108/23862 = 2^{-7.79}$) is quite close to our prediction (2^{-8}). This result demonstrate that LIFA is also relevant if LDFA works. We repeated Algorithm 2, 20 times and were able to obtain the correct value of $\Delta_{0,4} = sk_{10}[0] \oplus sk_{10}[4]$ in all cases. This is not unexpected given that our

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

fault configuration provides us a high success rate with a minimal number of missing or undesirable faults.

As stated in Section 3.4.2, the attacker might not be aware of the precise location of the fault or the term associated with the target value. Even though we were aware of this information, we assumed the attacker testing Algorithm 3 wouldn't be able to access them. Given one faulty ciphertext, Algorithm 3 is unable to determine the correct values for i and j. However, we performed it for more than one faulty ciphertext and observed that as the number of faulty ciphertexts rises, the success rate of the Algorithm 3 is also improved. 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.

5.3 LDFA and LIFA in the Presence of Countermeasure

To assess the formalization presented in Section 4.1, we performed LDFA in the presence of shuffling. We looked at $N = 2^{14}$ faulty ciphertexts by assuming that shuffling is used before the AES nonlinear layer. We ran the tests 1,000 times and discovered that the correct key can be identified exclusively 98.46% of the time. The success probability based on the available number of faulty ciphertexts is demonstrated in Table 2.

Table 2: Success probability of LDFA in the presence of shuffling

Number of faulty ciphertexts (N)	2^8	2^{10}	2^{12}	2^{14}
Success probability	66.54%	73.93%	86.48	98.46

We applied LFA on a publicly available byte-masked implementation [2] to check if LFA works with masked look-up tables in real life. 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 also applies in this instance.

Additionally, we applied LFA to the SIFA-protected, masked implementation of χ, which is publicly accessible[3]. We tried to apply LFA to the given source, but it does not work because the variables we wanted to be linked are not processed in order (see Section 4.3.2). But to see if LFA could be applied to SIFA-protected masking implementations, we moved code line 351 to line 347 so that two shares of a single variable could be processed successively. This modification does not alter the SIFA-masked implementation principle described in [DDE+20], but it makes it so that two shares of the same variable (like a) are handled one after the other. Then we used LFA on this changed implementation and saw that the target values, which are called a_0 and a_1, were linked. 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 as, in the event of ineffective events, information regarding an intermediate value is revealed. This does not negate the security assertion made in [DDE+20], as stated in Section 4.3.2, but it provides compelling proof that even SIFA-protected masking must be used with caution due to LIFA's impact on instruction order.

6 Conclusions

In this study, we present a new technique for fault analysis that employs a link between a faulty intermediate value and a second intermediate value. Our method, unlike most

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

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

fault analysis techniques, does not rely on a biased fault model and has the potential to circumvent many countermeasures. We proved the impact of LFA by suggesting two key-recovery approaches. Our first technique, LDFA, is applicable to unprotected implementations of block ciphers that require an extremely minimal amount of data. LIFA, our second method, can circumvent conventional detection- and infection-based countermeasures. LIFA is less affected by missed faults or unwanted faults compared to SIFA, although, unlike SEFA, it does not require input control.

References

[AM12] Subidh Ali and Debdeep Mukhopadhyay. Differential fault analysis of twofish. In *Inscrypt*, volume 7763 of *Lecture Notes in Computer Science*, pages 10–28. Springer, 2012.

[AM13] Subidh Ali and Debdeep Mukhopadhyay. Improved differential fault analysis of CLEFIA. In *FDTC*, pages 60–70. IEEE Computer Society, 2013.

[AMT13] Subidh Ali, Debdeep Mukhopadhyay, and Michael Tunstall. Differential fault analysis of AES: towards reaching its limits. *J. Cryptogr. Eng.*, 3(2):73–97, 2013.

[AWMN20] Victor Arribas, Felix Wegener, Amir Moradi, and Svetla Nikova. Cryptographic fault diagnosis using verfi. In *HOST*, pages 229–240. IEEE, 2020.

[BBB+18] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Mustafa Khairallah, and Thomas Peyrin. Protecting block ciphers against differential fault attacks without re-keying. In *HOST*, pages 191–194. IEEE Computer Society, 2018.

[BBKN12] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proc. IEEE*, 100(11):3056–3076, 2012.

[BCN+06] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proc. IEEE*, 94(2):370–382, 2006.

[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.

[BGN12] Céline Blondeau, Benoît Gérard, and Kaisa Nyberg. Multiple differential cryptanalysis using LLR and χ^2 statistics. In Ivan Visconti and Roberto De Prisco, editors, *Security and Cryptography for Networks – SCN 2012*, volume 7485 of *LNCS*, pages 343–360. Springer, 2012.

[BGV11] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus. In *FDTC*, pages 105–114. IEEE Computer Society, 2011.

[BHL18] Jakub Breier, Xiaolu Hou, and Yang Liu. Fault attacks made easy: Differential fault analysis automation on assembly code. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):96–122, 2018.

[BJC15] Jakub Breier, Dirmanto Jap, and Chien-Ning Chen. Laser profiling for the back-side fault attacks: With a practical laser skip instruction attack on AES. In *CPSS@ASIACSS*, pages 99–103. ACM, 2015.

[BK06] Johannes Blömer and Volker Krummel. Fault based collision attacks on AES. In *FDTC*, volume 4236 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2006.

[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.

[BSR+22] Nasour Bagheri, Sadegh Sadeghi, Prasanna Ravi, Shivam Bhasin, and Hadi Soleimany. Sipfa: Statistical ineffective persistent faults analysis on feistel ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):367–390, Jun. 2022.

[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.

[DCAM15] Prakash Dey, Abhishek Chakraborty, Avishek Adhikari, and Debdeep Mukhopadhyay. Improved practical differential fault analysis of grain-128. In *DATE*, pages 459–464. ACM, 2015.

[DDE+20] Joan Daemen, Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Florian Mendel, and Robert Primas. Protecting against statistical ineffective fault attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):508–543, 2020.

[DDR+12] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, P. Orsatelli, Philippe Maurine, and Assia Tria. Injection of transient faults using electromagnetic pulses -practical results on a cryptographic system-. *IACR Cryptol. ePrint Arch.*, page 123, 2012.

[DDRT12] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic transient faults injection on a hardware and a software implementations of AES. In *FDTC*, pages 7–15. IEEE Computer Society, 2012.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. 11273:315–342, 2018.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting ineffective fault inductions on symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):547–572, 2018.

[DFL11] Patrick Derbez, Pierre-Alain Fouque, and Delphine Leresteux. Meet-in-the-middle and impossible differential fault analysis on AES. In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 274–291. Springer, 2011.

[EHH+14] Sho Endo, Naofumi Homma, Yu-ichi Hayashi, Junko Takahashi, Hitoshi Fuji, and Takafumi Aoki. A multiple-fault injection attack by adaptive timing control under black-box conditions and a countermeasure. In *COSADE*, volume 8622 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2014.

[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.

[FCL+20] Jingyi Feng, Hua Chen, Yang Li, Zhipeng Jiao, and Wei Xi. A framework for evaluation and analysis on infection countermeasures against fault attacks. *IEEE Trans. Inf. Forensics Secur.*, 15:391–406, 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.

[FXKH17] Kazuhide Fukushima, Rui Xu, Shinsaku Kiyomoto, and Naofumi Homma. Fault injection attack on salsa20 and chacha and a lightweight countermeasure. In *TrustCom/BigDataSE/ICESS*, pages 1032–1037. IEEE Computer Society, 2017.

[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.

[GST12] Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In *LATINCRYPT*, volume 7533 of *Lecture Notes in Computer Science*, pages 305–321. Springer, 2012.

[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.

[HBZL19] Xiaolu Hou, Jakub Breier, Fuyuan Zhang, and Yang Liu. Fully automated differential fault analysis on software implementations of block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):1–29, 2019.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, 2006.

[KH14] Thomas Korak and Michael Hoefler. On the effects of clock and power supply tampering on two microcontroller platforms. In *FDTC*, pages 8–17. IEEE Computer Society, 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.

[MDH+13a] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In *FDTC*, pages 77–88. IEEE Computer Society, 2013.

[MDH+13b] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In *FDTC*, pages 77–88. IEEE Computer Society, 2013.

[PM19] Sikhar Patranabis and Debdeep Mukhopadhyay. *Idempotent Instructions to Counter Fault Analysis Attacks*, pages 195–208. Springer International Publishing, Cham, 2019.

[PP21] Peter Pessl and Lukas Prokop. Fault attacks on cca-secure lattice kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):37–60, 2021.

[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.

[RCC+16] Debapriya Basu Roy, Avik Chakraborti, Donghoon Chang, S. V. Dilip Kumar, Debdeep Mukhopadhyay, and Mridul Nandi. Fault based almost universal forgeries on CLOC and SILC. In *SPACE*, volume 10076 of *Lecture Notes in Computer Science*, pages 66–86. Springer, 2016.

[RSS+21] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. FIVER - robust verification of countermeasures against fault injections. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):447–473, 2021.

[SBH+21] Hadi Soleimany, Nasour Bagheri, Hosein Hadipour, Prasanna Ravi, Shivam Bhasin, and Sara Mansouri. Practical multiple persistent faults analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):1–24, 2021.

[SBHS15] Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise laser fault injections into 90 nm and 45 nm sram-cells. In *CARDIS*, volume 9514 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2015.

[SBR+20] Sayandeep Saha, Arnab Bag, Debapriya Basu Roy, Sikhar Patranabis, and Debdeep Mukhopadhyay. Fault template attacks on block ciphers exploiting fault propagation. In *EUROCRYPT (1)*, volume 12105 of *Lecture Notes in Computer Science*, pages 612–643. Springer, 2020.

[SH08] Jörn-Marc Schmidt and Christoph Herbst. A practical fault attack on square and multiply. In *FDTC*, pages 53–58. IEEE Computer Society, 2008.

[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.

[SHS16] Bodo Selmke, Johann Heyszl, and Georg Sigl. Attack on a DFA protected AES by simultaneous laser fault injections. In *FDTC*, pages 36–46. IEEE Computer Society, 2016.

[SMC09] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. A diagonal fault attack on the advanced encryption standard. *IACR Cryptol. ePrint Arch.*, page 581, 2009.

[SMD18] Sayandeep Saha, Debdeep Mukhopadhyay, and Pallab Dasgupta. Expfault: An automated framework for exploitable fault characterization in block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):242–276, 2018.

[TBM14] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Destroying fault invariant with randomization - A countermeasure for AES against differential fault attacks. In *CHES*, volume 8731 of *Lecture Notes in Computer Science*, pages 93–111. Springer, 2014.

[TK10] Elena Trichina and Roman Korkikyan. Multi fault laser attacks on protected CRT-RSA. In *FDTC*, pages 75–86. IEEE Computer Society, 2010.

[VZB+22] Navid Vafaei, Sara Zarei, Nasour Bagheri, Maria Eichlseder, Robert Primas, and Hadi Soleimany. Statistical effective fault attacks: The other side of the coin. *IEEE Trans. Inf. Forensics Secur.*, 17:1855–1867, 2022.

[YGS15] Bilgiday Yuce, Nahid Farhady Ghalaty, and Patrick Schaumont. Improving fault attacks on embedded software using RISC pipeline characterization. In *FDTC*, pages 97–108. IEEE Computer Society, 2015.

[YGS+16] Bilgiday Yuce, Nahid Farhady Ghalaty, Harika Santapuri, Chinmay Deshpande, Conor Patrick, and Patrick Schaumont. Software fault resistance is futile: Effective single-glitch attacks. In *FDTC*, pages 47–58. IEEE Computer Society, 2016.

[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.

[ZGZ+13] Xinjie Zhao, Shize Guo, Fan Zhang, Zhijie Shi, Chujiao Ma, and Tao Wang. Improving and evaluating differential fault analysis on LED with algebraic techniques. In *FDTC*, pages 41–51. IEEE Computer Society, 2013.

[ZGZ+16] Fan Zhang, Shize Guo, Xinjie Zhao, Tao Wang, Jian Yang, François-Xavier Standaert, and Dawu Gu. A framework for the analysis and evaluation of algebraic fault attacks on lightweight block ciphers. *IEEE Trans. Inf. Forensics Secur.*, 11(5):1039–1054, 2016.

[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.

[ZZG+13] Fan Zhang, Xinjie Zhao, Shize Guo, Tao Wang, and Zhijie Shi. Improved algebraic fault analysis: A case study on piccolo and applications to other lightweight block ciphers. In *COSADE*, volume 7864 of *Lecture Notes in Computer Science*, pages 62–79. Springer, 2013.

[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.