IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 3, pp. 367–390. DOI:10.46586/tches.v2022.i3.367-390

SIPFA: Statistical Ineffective Persistent Faults Analysis on Feistel Ciphers

Nasour Bagheri[1,2], Sadegh Sadeghi[3,4], Prasanna Ravi[5], Shivam Bhasin[5] and Hadi Soleimany[6]

[1] CPS^2 lab., Shahid Rajaee Teacher Training University, Tehran, Iran
[2] School of Computer Science (SCS), Institute for Research in Fundamental Sciences (IPM),Tehran, Iran
nbagheri@sru.ac.ir
[3] Department of Mathematics, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan 45137-66731, Iran
[4] Research Center for Basic Sciences and Modern Technologies (RBST), Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan 45137-66731, Iran
s.sadeghi@iasbs.ac.ir
[5] Temasek Laboratories, NTU, Singapore
prasanna.ravi@ntu.edu.sg,sbhasin@ntu.edu.sg
[6] Cyber Research Center, Shahid Beheshti University, Tehran, Iran
h_soleimany@sbu.ac.ir

Abstract.
Persistent Fault Analysis (PFA) is an innovative and powerful analysis technique in which fault persists throughout the execution. The prior prominent results on PFA were on SPN block ciphers, and the security of Feistel ciphers against this attack has received less attention. In this paper, we introduce a framework to utilize Statistical Ineffective Fault Analysis (SIFA) in the persistent fault setting by proposing Statistical Ineffective Persistent Faults Analysis (SIPFA) that can be efficiently applied to Feistel ciphers in a variety of scenarios. To demonstrate the effectiveness of our technique, we apply SIFPA on three widely used Feistel schemes, DES, 3DES, and Camellia. Our analysis reveals that the secret key of these block ciphers can be extracted with a complexity of at most 2^{50} utilizing a single unknown fault. Furthermore, we demonstrate that the secret can be recovered in a fraction of a second by increasing the adversary's control over the injected faults. To evaluate SIPFA in a variety of scenarios, we conducted both simulations and real experiments utilizing electromagnetic fault injection on DES and 3DES.

Keywords: Fault Attack · Persistent Fault Analysis · Statistical Ineffective Fault Analysis · Feistel Ciphers · DES · 3DES · Camellia

1 Introduction

Fault attacks are a powerful type of physical attacks that can jeopardize the security of cryptographic systems significantly. Such attacks rely on an attacker's ability to inject a fault into the target device using a variety of tools such as clock glitches, Electromagnetic (EM) pulses, laser beam, etc. The faulty output (or some information about it) can be used to deduce critical information about the cryptographic algorithm and, ultimately, the secret key.

Starting with the seminal work by Boneh et al. on RSA [BDL97], numerous studies have assessed the security of cryptographic implementations against a wide range of fault

Received: 2022-01-15 Accepted: 2022-03-15 Published: 2022-06-08

attacks [BBB$^+$22]. According to the duration of the fault, the generated faults can be categorized into three classes. The transient fault has a temporary effect on the device. Most of the fault attacks proposed in the literature fall in this category. One can mention *Differential Fault Analysis* (DFA) [BS97], Fault Sensitivity Analysis (FSA) [LSG$^+$10], Differential Fault Intensity Analysis (DFIA) [GYTS14], Safe-Error Analysis (SEA) [YJ00], Ineffective Fault Analysis (IFA) [Cla07], Statistical Fault Analysis (SFA) [FJLT13], Statistical Ineffective Fault Analysis (SIFA) [DEK$^+$18], and so on. The permanent fault is a second sort of fault that has irreversible consequences that last during the lifetime of the device. Another sort of fault is the persistent fault, which persists but disappears when the target device is reset.

1.1 Previous Works

Schmidt et al. [SHP09] introduced the term “permanent” fault when they demonstrated an attack on AES using ultraviolet light to erase the non-volatile memory. In the follow-up work [ZLZ$^+$18], the Persistent Fault Analysis (PFA) technique was presented at CHES 2018, which is a novel and powerful analysis technique in which faults persist throughout the execution. PFA has several advantages over the previous fault attacks. The fault does not have to be injected at a particular time-synchronized to the encryption process. Additionally, PFA is capable of circumventing the majority of redundancy-based countermeasures for AES, such as detection- or infection-based countermeasures [ZLZ$^+$18]. The typical target for the persistent fault is memory holding, for example, the Sbox look-up table. The main shortcoming of the original PFA is that it assumes that the attacker is aware of the location and the value of the fault, which is hard to achieve in practice. Zhang et. Al. [ZZJ$^+$20] demonstrated that the requirement for the precise knowledge of the fault value can be relaxed. However, their solution is only applicable to a single fault and does not work in the case of multiple faults where the attacker does not know the faulty values. Inspired by SFA, Engels et al. [ESP20] presented Statistical Persistent Fault Analysis (SPFA) to perform PFA in the presence of multiple faults. Soleimany et al. proposed a framework to apply multiple persistent faults analysis without making any assumptions about the knowledge of faulty values [SBH$^+$21].

1.2 Motivation

Despite significant developments in the application of PFA to SPN ciphers, there is little study on its influence on Feistel ciphers. It appears that applying it to Feistel ciphers is challenging. Each round of an SPN cipher is composed of three layers: a non-linear (substitution) layer, a linear (permutation) layer, and a round key addition. In the majority of SPN ciphers, bijective Sboxes are applied in parallel to the entire state in the substitution layer. As a result, an unbiased distribution in the output of Sboxes caused by persistent fault can result in an unbiased distribution in the rounds’ outputs. This fact has been used directly in prior works to perform key recovery on SPN ciphers.

The Feistel ciphers divide the input state of the round X into two equal halves (X^L, X^R). In comparison to SPN, the round function of a Feistel cipher is applied to only half of the state $F_{sk_r}(X^L)$, where sk_r denotes the r-th round’s subkey. The result $F_{sk_r}(X^L)$ is then XORed with the other half of the state X^R. It means that the output of each round of Feistel ciphers is masked by the output of the preceding round, thereby destroying the statistical characteristic in the output of rounds employed in PFA. Specifically, the presence of an impossible value in the Sbox output does not imply the presence of an impossible value in the ciphertext. The reason for this is self-evident: first, the substation layer affects only half of the state. Second, the output of Sboxes is masked by unknown random values from the preceding round. As a result, techniques that rely on the use of impossible values will fail to succeed in case of Feistel ciphers. This is the reason that the proposed PFA techniques in [ZLZ+18, ZZJ+20, ESP20, SBH+21] are unable to recover the key from the Feistel ciphers.

Caforio and Banik [CB19] presented a version of PFA on DES. However, they assumed that the adversary had access to both faulty and fault-free ciphertexts in their study, which is comparable to DFA attacks. Furthermore, if there is a detection/countermeasure mechanism on the implemented cipher, the suggested attack would fail. PFA's two significant advantages are that it requires only faulty ciphertexts and it is capable of bypassing detection-based defenses. As a result, the proposed attack in [CB19] has a very limited impact. Due to this limitation, we investigate a statistical method for extending PFA's application to Feistel cipher s in the context of ciphertext-only attacks in the presence of popular countermeasures.

1.3 Our Contributions

Statistical Ineffective Fault Attacks (SIFA) [DEK+18] is a new type of fault attack that combines the concepts of Statistical Fault Attack (SFA) [FJLT13] and Ineffective Fault Attacks (IFA) [Cla07]. SIFA makes use of statistical information from instances in which the injected fault does not influence the output. If the transient fault has a temporary effect, SIFA can easily use the statistical distribution of an intermediate value impacted by the transient fault. However, when a persistent fault exists, applying SIFA can be challenging since the persistent fault affects not just one intermediate value, but all intermediate values in different rounds. This paper shows how to adopt the SIFA as a powerful method to tackle the challenges of applying PFA to Feistel ciphers. SIPFA has many of the same advantages as original PFA, such as the ability to circumvent detection- and infection-based countermeasures, and it can be used in the ciphertext-only scenario. Moreover,Pan *et al.* [PZRB19] demonstrated that masking countermeasures can be broken at any masking order, similar to SIFA which can overcome masked implementations [DEG+18]. Hence, SIPFA can also be applied in the presence of masking countermeasures. In comparison to SIFA, SIPFA has the following advantages:

- Dummy operations and countermeasure shuffling can affect SIFA's performance, but they have no effect on SIPFA's.
- A noisy setup has a stronger impact on SIFA compared to SIPFA, because SIFA cannot detect missed faults.

Our novel techniques enable the extension of PFA's applicability to several types of Feistel ciphers.We explore two distinct scenarios depending on whether the attacker is aware of the fault value or not. Then we apply our techniques to DES [S+77, DES99], 3DES, and Camellia [AIK+00]. To the best of our knowledge, this is the first use of PFA on these ciphers that can be carried out in the ciphertext-only attack scenario in a reasonable amount of time, even in the presence of detection- and infection-based countermeasures[1]. DES is the most well-known example of a Feistel cipher selected as a standard by the US government in 1976. Due to the fact that the key size of the original DES cipher was insufficient for the majority of applications, Triple DES is specified in multiple standards, including ISO/IEC and NIST. In the literature, triple DES is sometimes abbreviated as TDES, TDEA, and 3DES. 3DES has been widely employed in practical applications, although it has been replaced by other ciphers due to the presentation of the Sweet32 attack [BL16] and deprecated by NIST in 2017. Camellia is an ISO/IEC standard and CRYPTREC-portfolio cipher developed in collaboration between NTT and Mitsubishi.

Table 1 gives a summary of our results as well as a comparison with the relevant work [CB19]. It demonstrates that a single unknown persistent fault is sufficient to

[1]The source codes for our simulations in `C` language are publicly available at `https://github.com/sadeghi87/SIPFA.git`

Table 1: The comparison between PFA on DES, 3DES and Camellia-128/192/256 under different assumptions, where BDC stands for by-pass Detection Countermeasures

Algorithem	Attack Scenario	Unknown Fault	# Fault	BDC	Time	Data	Ref.
DES	Chosen Plaintext	✗	8	✗	2^{20}	656	[CB19]
	Ciphertext only	✓	8	✓	neg.	3120	Section 4.1
	Ciphertext only	✗	1	✓	$2^{42.97}$	2574	
3DES	Ciphertext only	✓	8	✓	neg.	5176	Section 4.2
	Ciphertext only	✗	1	✓	$2^{42.97}$	4255	
Camellia-128	Ciphertext only	✓	4	✓	neg.	7240	Section 4.3
	Ciphertext only	✗	1	✓	$2^{49.49}$	7483	
Camellia-192	Ciphertext only	✓	4	✓	neg.	7592	
	Ciphertext only	✗	1	✓	$2^{49.67}$	7843	
Camellia-256	Ciphertext only	✓	4	✓	neg.	7592	
	Ciphertext only	✗	1	✓	$2^{50.08}$	7843	

extract the master key of DES, triple DES (3DES), and versions of Camellia (Camellia-128/192/256) in a reasonable amount of time (almost 2^{50}) that is feasible in light of current technologies. If the attacker can inject eight controlled faults on the Sboxes of DES/3DES or four controlled faults on the Sboxes of Camellia (all versions, i.e., Camellia-128/192/256), the secret key can be extracted in a fraction of a second.

1.4 Outline

The remainder of this paper is structured as follows. Section 2 provides an overview of PFA and the block ciphers analyzed in this paper. In Section 3, we describe a new technique called SIPFA that enables the extension of the PFA algorithm to Feistel ciphers. Section 4 demonstrates the method's adaptability by describing its applicability to DES, 3DES, and Camellia in various scenarios. We present our simulation results in Section 5. We discuss the practical aspects of SIPFA in Section 6 by describing our practical fault injection experiments. Finally, in Section 7, we conclude.

2 Preliminaries

2.1 Notations

In this section, we introduce the notations used in this paper. We use b and k respectively to denote the block length and the key length of a target Feistel cipher. K and C respectively denote the master key and the ciphertext. n represents the total number of rounds of the cipher, and the round number is denoted by r and appears as a subscript of the target parameter, e.g., X_{n-1} denotes the intermediate value X at the penultimate round. Given that this study focuses on classic Feistels, L and R are used to refer to the left and right parts of the state or ciphertext, respectively, and always appear in superscript of the target parameter. For example, C^L refers to the left part of the ciphertext C. Since we may require N ciphertexts, we use j to distinguish them, e.g., C^j and $\left(C^j\right)^L$ respectively denotes j-th ciphertext and its corresponding left part. The target cipher may have several Sboxes, and the persistent fault may be injected into any of them. The number of distinct Sboxes is denoted by ℓ, and a ciphertext produced under fault injection in i-th Sbox is denoted by C_i. Hence, $(C_i^j)^L$ denotes the left part of the j-th ciphertext generated when a fault is injected into the i-th Sbox. Each round r of the target cipher uses a subkey denoted by sk_r.

2.2 Statistical Ineffective Fault Attack (SIFA)

SIFA [DEK+18] is empowered by SFA [FJLT13] and IFA [Cla07] attacks simultaneously. It is designed to take advantage of faulty encryptions in which the induced faults are ineffective, and therefore the corresponding ciphertexts are always correct. The redundancy-based countermeasures fail to identify, suppress, or infect this type of ciphertext. On the other hand, although the effect of an ineffective fault is not visible in its output, this does not rule out the possibility that it is leaking exploitable information. [DEK+18] demonstrates that the majority of practical fault inductions, whether effective or not, modify the target intermediate variable with a non-uniform probability distribution. SIFA exploits a bias in a target value's intermediate value over ineffective events. To apply a key-recovery attack, the adversary obtains the distribution for the target intermediate value by partially decrypting the ineffective ciphertexts for each key candidate. The adversary then attempts to retrieve the correct key from a list of key candidates using a statistical test. The statistical test is determined according to the attacker's extent of control over the injected fault. If the attacker is aware of the distribution of the faulty value, he can rank the key candidates using LLR. If the attacker does not know the distribution of the faulty intermediate value other than that it is biased over ineffective events, he can use the SEI or Pearson's chi-squared test. Given a large enough sample size of ineffective ciphertexts, the correct key's statistic is usually the highest.

2.3 Persistent Faults Analysis on SPN ciphers

Zhang et al. [ZLZ+18] considered an R-round word-oriented Substitution-Permutation Network (SPN) cipher $E_K(P)$, which accepts two inputs: b-bit plaintext block P and k-bit key K. SPN ciphers are constructed by repeatedly performing an invertible function known as round $\mathcal{R}$. Typically, the b-bit state X is composed of L words of the same size $m = b/L$ indicated by $X[i]$ where $0 \leq i \leq L-1$. The j-th word input and output of the substitution layer in the r-th round are denoted by $x_r[j]$ and $z_r[j]$, respectively. The j-th word of ciphertext equals to $C[j] = y[j] + sk_n[j]$, where $sk_n[j]$ is the last round key's j-th word. Assume that the induced fault into the look-up table of Sbox modifies the value v to the faulty value $v^* \neq v$. Thus, the value of v is missing from $y_n[j]$, while the value of v^* is supposed to occur twice as frequently in $y[j]$. While the probability distributions of $y_n[j]$ and $C[j]$ are uniform for the correct encryptions, $y[j]$ and $C[j]$ are not uniformly in case of faulty encryption. Given some faulty ciphertexts, the adversary can count the appearances of each of the possible values in $C[j]$ for all $0 \leq j \leq L-1$. By utilizing a sufficiently large number of ciphertexts N, it is possible to determine the minimal and maximal number of counts for each word of the ciphertext, which are denoted by $C_{min}[j]$ and $C_{max}[j]$, respectively. Given $C_{min}[j]$ is the value that is never observed in $C[j]$, the corresponding word of the last round key can be derived as $sk_n[j] = C_{min}[j] \oplus v$. Similarly, if $C_{max}[j]$ is the value that occurs twice as frequently as other values, the j-th work of the last round key can be retrieved as $sk_n[j] = C_{max}[j] \oplus v^*$. These methods are based on a disputable assumption that the adversary is aware of the precise position and the value of the fault. Zhang et al. followed the main idea proposed in the initial attack but used Maximum Likelihood Estimation to derive the key in the absence of knowledge of the fault location or faulty value [ZZJ+20].

2.4 Generalized Round of Feistel Ciphers

We consider a b-bit Feistel cipher E which includes n rounds. A round of this cipher is depicted in Figure 1, where X_r^L and X_r^R respectively denote the left and the right half of the input of the r-th round and:

$$X_{r+1}^L = F_r(X_r^L, sk_r) \oplus X_r^R, \quad (1)$$
$$X_{r+1}^R = X_r^L \quad (2)$$

In general, the r-th round-function of such block cipher consists of three layers, as shown in Figure 1, i.e., $F_{r,I}$ as the input layer, S as nonlinear layer, and $F_{r,O}$ as the output layer. We assume that $F_{r,I}$ is linear but do not have any restriction on $F_{r,O}$. Without the loss of generality, we assume that the Sbox layer includes ℓ calls to $S_1, \ldots, S_\ell$. Some of the Sboxes can be identical such as that of Camellia or they are all distinct Sboxes such as that of DES. Furthermore, we make no assumption about the bijectivity of the Sboxes when considering Feistel structure. We denote the input length of each Sbox by m. The input and output of $F_{r,I}$ are respectively denoted by x_r and y_r and the output of S is denoted by z_r. We denote the input and output of the i-th word of the Sbox layer, for $1 \leq i \leq \ell$, in the r-th round by $y_r[i]$ and $z_r[i]$, respectively. As a consequence, assuming that $F_{r,I}$ is a linear function, $x_n[i] = y_n[i] \oplus sk_n[i]$ can be determined from the ciphertext $\mathcal{C}$, where $sk_n[i]$ is the last round key.

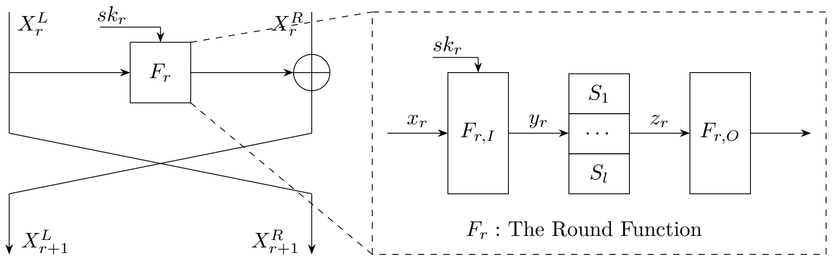


Figure 1: Generalized round of a Feistel cipher

2.5 Target ciphers

This section introduces the block ciphers that will be analyzed in this paper.

2.5.1 DES Cipher

DES is a 64-bit Feistel cipher. The key is supposed to be 64 bits long, but only 56 of them are utilized by the algorithm. The general structure of DES is composed of an initial permutation, sixteen identical rounds, and a final permutation. The round function F takes a 32-bit intermediate cipher word and a 48-bit round key as inputs. The F-function consists of four steps:

1) **Expansion:** The expansion permutation E expands a 32-bit input to a 48-bit output.

2) **Key addition:** The state is XORed with the 48-bit round key.

3) **Substitution:** The state is split into eight 6-bit blocks. Each block is sent into a separate 6-to-4 Sbox, generating one 4-bit output.

4) **Permutation:** The fixed 32-bit to 32-bit permutation is applied to the state

The DES key schedule is a linear function that derives the round keys by selecting 48 of the master key's 56 bits. We refer the interested readers to [S+77, DES99] for more details.

2.5.2 3DES Cipher

In the face of contemporary cryptanalytic techniques and super-computing capabilities, the Data Encryption Standard's 56-bit key is no longer deemed sufficient. Triple DES (3DES), an extended variant of DES, provides a straightforward way to increase the key size without requiring the design of a new block cipher. 3DES performs three iterations of DES using three distinct keys of 56 bits each: K_1, K_2, and K_3. The encryption algorithm is defined as $C = DES_{K_3}(DES^{-1}_{K_2}(DES_{K_1}(M)))$, and the decryption is the inverse of encryption: $M = DES^{-1}_{K_1}(DES_{K_2}(DES^{-1}_{K_3}(C)))$.

2.5.3 Camellia Block Cipher

Camellia [AIK+00] is a 128-bit block cipher with a Feistel structure with a key size range of 128, 192, and 256. Camellia includes 18 rounds for keys with a 128-bit length and 24 rounds for keys with 192- or 256-bit length. The process of encryption in the Camellia-128 (resp. 192/256-bit key) is as follows. The 128-bit plaintext M is first XORed with whitening keys $kw_1||kw_2$ and then split into two 64-bit data X^L_0 and X^R_0. Then, except for $r = 6$ and 12 (resp. $r = 6$, 12 and 18), the following operations are performed from $r = 1$ to 18 (resp. $r = 1$ to 24).

$$X^L_r = X^R_{r-1} \oplus F(X^L_{r-1}, sk_r), \qquad X^R_r = X^L_{r-1},$$

where sk_r is the r-th round key and X_{r-1} is the input of the the r-th round. The following operations are performed for $r = 6$ and 12 (resp. $r = 6, 12$, and 18):

$$\overline{X^L_r} = X^R_{r-1} \oplus F(X^L_{r-1}, sk_r), \qquad \overline{X^R_r} = X^L_{r-1}$$

$$X^L_r = FL(\overline{X^L_r}, kf_{r/3-1}), \qquad R_r = FL^{-1}(\overline{X^R_r}, kl_{r/3}).$$

Finally, using $kw_3||kw_4$ the 128-bit ciphertext C is calculated as $C = (X^R_{18}||X^L_{18}) \oplus (kw_3||kw_4)$ (resp. $C = (X^R_{24}||X^L_{24}) \oplus (kw_3||kw_4)$).

The round function F includes three layers: a key-addition layer, a substitution transformation layer S (non-linear part), and a diffusion layer P (linear part). Therefore, the round function F for the r-th round is computed as

$$y_r = X^L_{r-1} \oplus sk_r, \tag{3}$$
$$z_r = S(Y_r), \tag{4}$$
$$w_r = P(Z_r), \tag{5}$$

The nonlinear layer S is composed of four different s-boxes, S_1, S_2, S_3, and S_4. If we demonstrate y_r as 8-byte $(y[1], y[2], y[3], y[4], y[5], y[6], y[7], y[8])$, the output z_r is computed as

$$z_r = S_1(y[1]), S_2(y[2]), S_3(y[3]), S_4(y[4]), S_2(y[5]), S_3(y[6]), S_4(y[7]), S_1(y[8]).$$

We refer the interested readers to [AIK+00] for the details.

3 Statistical Ineffective PFA on Feistel ciphers

3.1 Main Idea

Statistical Ineffective Fault Attack takes advantage of faulty encryptions in which the induced faults are ineffective, and therefore the corresponding ciphertexts are always correct. It is shown in [DEK+18] that most transient fault inductions cause the targeted intermediate variable to shift into a new value with a non-uniform probability distribution. The persistent fault does not affect a single intermediate value, but affects all intermediate values in all rounds. As a result, when a persistent fault exists, utilizing ineffective faults should be done cautiously and after thorough analysis.

In contrast to first impressions, we demonstrate that ineffective faults can be leveraged to overcome the difficulties associated with applying PFA to Feistel ciphers after comprehensive investigation. In this section, we use the statistical characteristics of persistent faults to apply PFA on Feistel ciphers, but rather than considering statistical characteristics across all data we consider only those in which the fault is ineffective.

Let us assume that the inserted fault transforms the correct value $S_i(\delta_i) = v$ into the incorrect value $v^* = v \oplus \Delta_i$. If $y_r[i] = \delta_i$, for any $r \in \{1, \ldots, n\}$, the ciphertext is also affected by the fault with the probability 1, and the returned value is either an empty string $\perp$ or a random string ϕ, depending on the employed countermeasure. On the other hand, if $\{y_1[i], \ldots, y_n[i]\} \cap \delta_i = \varnothing$, it implies that the persistent fault is ineffective. As a result, the device returned the correct ciphertext. We denote the ciphertext that is not affected by the fault injected in the i-th Sbox by C_i.

If the round function makes a single call to the target Sbox per round, the rate of ineffective events can be estimated based on Equation (6).

$$\Pi_{\mathsf{c}} = (1 - 2^{-m})^n. \tag{6}$$

If the Sbox-layer of the round contains t calls to a single Sbox, the rate of ineffective events can be determined using Equation (7):

$$\Pi_{\mathsf{c}} = (1 - 2^{-m})^{t \cdot n} \tag{7}$$

In a ciphertext only scenario, given N correct ciphertexts C_i^1 where $1 \leq j \leq N$, the following relation holds:

$$\{y_1^1[i], \ldots, y_n^N[i]\} \cap \delta_i = \varnothing \tag{8}$$

Equation (8) not only implies that there exist a bias in the value of $y[i]$ over correct ciphertexts (resulted in ineffective events) but also a bias in the value of x. The precise relationship is determined by $F_{r,I}$. If $x_r[i] = y_r[i] \oplus sk_r[i]$, the following relation holds:

$$\{x_1^1[i], \ldots, x_n^N[i]\} \cap (\delta_i \oplus k_n[i]) = \varnothing \tag{9}$$

We assume that the implementation of the target block cipher includes a countermeasure based on the redundancy encryption in which different Sbox lookup tables are used (common faulty Sbox naturally bypasses several redundancy scheme [ZLZ+18]). Depending on the deployed countermeasure, i.e. detection-based or infection-based, the output value of faulty computation is either an empty string $\perp$ or a random string ϕ. In the case of detection-based countermeasure, given $\hat{N}$ queries, there exist $\Pi_{\mathsf{c}} \times \hat{N}$ correct ciphertext that fulfill Eqs. (8) and (9). In the case of infection-based countermeasure, there exist $\Pi_{\mathsf{c}} \times \hat{N}$ which satisfy Eqs. (8) and (9) but they are not distinguishable. The incorrect ciphertexts (resulted in effective fault) satisfy Eqs. (8) and (9) with the probability of 2^{-m}. As a result, for large values of $\hat{N}$, the distribution of $x_n[i] = \delta_i \oplus sk_n[i]$ retains a bias that can be exploited to perform key recovery using a statistical test.

Due to the fact that the bias exists over ciphertexts with ineffective faults and the injected fault is permanent, we refer to the attack that makes use of this source of information as Statistical Ineffective PFA, or SIPFA. The remainder of this section discusses our attack in depth against each of the detection- and infection-based countermeasures. While knowledge of the fault's location is a less likely assumption, it can provide more power to perform a more efficient attack. We will consider two situations depending on the assumption that the attacker is either aware or unaware of the fault's value.

3.2 SIPFA in the Presence of Detection-based Countermeasures

If $y_r[i] = \delta_i$, for any $r \in \{1, \ldots, n\}$, then the ciphertext will be affected by the fault and the returned value will be an empty string $\perp$, in the presence of a detection-based countermeasure. On the other hand, if the returned value $\mathcal{C} \neq \perp$, it implies that the ciphertext was not affected by the fault. As a result, if the adversary obtains N correct ciphertexts $C_i^1, \ldots, C_i^N$ then $\{x_1^1[i], \ldots, x_n^N[i]\} \cap (\delta_i \oplus k_n[i]) = \varnothing$ and for large values of N the target $\delta_i \oplus sk_n[i]$ can be determined uniquely. We consider two scenarios, in which the fault's value and position are known or unknown to the attacker.

3.2.1 Known Fault Value and Location

Provided $C_i^1, \ldots, C_i^N$ and knowledge of δ_i and i, the adversary can determine $x_n[i]$ for all the given ciphertexts. The values appearing at $x_n[i]$ cannot be candidates for $\delta_i \oplus sk_n[i]$; only the minimum values appearing at $x_n[i]$ will be considered. With a sufficiently large N, only one minimum value remains at $x_n[i]$ which fulfills the relation $x_n[i] = \delta_i \oplus sk_n[i]$ and consequently $sk_n[i]$ can be retrieved as $\delta_i = x_n[i] \oplus sk_n[i]$. To determine any other word of sk_n, e.g. $sk_n[j]$, the system will be reset, a specific fault δ_j is injected at S_j and the attack is repeated. Hence, we can retrieve $sk_n[i]$ for every $1 \leq i \leq \ell$ as well as the the entire sk_n. In addition, we can compute $(X_i^j)_n^L \| (X_i^j)_n^R$ for every ciphertext $C_i^j = (C_i^j)^L \| (C_i^j)^R$ given sk_n. Then, using the same procedure, we can determine sk_{n-1}. This method enables the determination of all round keys. The details of this procedure is represented in Algorithm 1.

We should have enough ciphertext N so that all possible values occur at $x_n[i]$ at least once. Following [SBH+21], the number of ciphertexts required to observe h values at $x_n[i]$ can be estimated as follows:

$$N = h \cdot \mathrm{H}(h) \tag{10}$$

where $\mathrm{H}(h)$ is the h-th harmonic number. To determine $\delta_i \oplus sk_n[i]$ uniquely we set $h = 2^m - 1$. The required data to obtain N ineffective ciphertexts can be estimated as follows:

$$\hat{N} = \frac{N}{\Pi_{\mathsf{c}}} = \frac{h \cdot \mathrm{H}(h)}{\Pi_{\mathsf{c}}} \tag{11}$$

Consequently, the required data to determine the subkeys for all ℓ' distinct Sboxes equals to $\ell' \cdot \hat{N}$.

3.2.2 Unknown fault value and location

When the adversary is unaware of both δ_i and i, he should first determine the fault location, i.e. i. The adversary can determine the location of the fault by considering the fact that the target intermediate value $x_n[i]$ (which corresponds to the faulty Sbox S_i), includes at least one minimum value regardless of the number of ciphertexts, whereas the other words at x_n may contain no minimum value for large N. Given the fault location, to determine the fault value δ_i, the adversary can guess the involved bits of subkeys to decrypt over one round to determine the minimum values at $x_{n-1}[i]$. The correct guess of the subkeys should lead to a minimum value at $x_{n-1}[i]$. In other words, any guess for which $x_{n-1}[i]$ does not have a minimum value is incorrect. Following this approach it is possible to determine the location and value of the fault, as well as certain bits of the last round's subkey. This technique is described in Algorithm 2.

The probability of a wrong key passing the specified filter in Step 15 is $(1 - 2^{-m})^N$, whereas the total number of remaining subkeys is $2^\kappa \cdot (1 - 2^{-m})^N$. If the adversary aims to gain a-bit advantage, then the expected candidates returned by Step 16 should be $2^{\kappa - a}$. Hence, $N = \frac{-a}{log_2(1-2^{-m})}$ is the expected value to achieve a bit advantage.

Algorithm 1 Key-recovery on a Feistel cipher with a detection-based countermeasure, assuming that δ_i is known for $1 \le i \le \ell$, where $C_i^j = (C_i^j)^L \| (C_i^j)^R$

```
Require: Ineffective ciphertexts $(C_i^1, \dots, C_i^N)$, for $1 \le i \le \ell$, that are all generated using the
    same master key $K$.
Ensure: The candidates for $sk_n, \dots, sk_1$
 1: for $i \leftarrow 1$ to $\ell$ do
 2:     $\mathcal{X} = \{0, \dots, 2^m - 1\}$
 3:     $S_i[\delta_i] \leftarrow S_i[\delta_i] \oplus v_i$
 4:     for $j \leftarrow 1$ to $N$ do
 5:         Get a correct ciphertext $C_i^j$
 6:         $x_n \leftarrow F_{n,I}((C_i^j)^L)$
 7:         $\mathcal{X} \leftarrow \mathcal{X} / x_n[i]$
 8:     $sk_n[i] \leftarrow \mathcal{X} \oplus \delta_i$                    // returns $sk_n[i]$
 9: for $i \leftarrow 1$ to $\ell$ do
10:     for $j \leftarrow 1$ to $N$ do
11:         $(X_i^j)_{n-1}^L \leftarrow (C_i^j)^L$
12:         $(X_i^j)_{n-1}^R \leftarrow (C_i^j)^R \oplus F_n((C_i^j)^L, sk_n)$
13: for $r \leftarrow n-1$ down to 1 do
14:     for $i \leftarrow 1$ to $\ell$ do
15:         $\mathcal{X} = \{0, \dots, 2^m - 1\}$
16:         for $j \leftarrow 1$ to $N$ do
17:             $x_r \leftarrow F_{r,I}((X_i^j)_r^L)$
18:             $\mathcal{X} \leftarrow \mathcal{X} / x_r[i]$
19:         $sk_r[i] \leftarrow \mathcal{X} \oplus \delta_i$
20:     for $i \leftarrow 1$ to $\ell$ do
21:         for $j \leftarrow 1$ to $N$ do
22:             $(X_i^j)_{n-1-h}^L \leftarrow (X_i^j)_r^R$
23:             $(X_i^j)_{n-1-h}^R \leftarrow (X_i^j)_r^L \oplus F_r((X_i^j)_r^R, sk_r)$
24: return $sk_n, \dots, sk_1$
```

If the round's Sbox-layer includes multiple applications of the faulty Sbox , e.g. $S_i = S_{i'}$, then it is possible to use other techniques to reduce the keyspace. For example, in this case, Step 9 returns 2 Sboxes with minimum value and they are linearly dependent as follows:

$$\left.\begin{array}{rcl} x_n[i] &=& \delta_i \oplus sk_n[i] \\ x_n[i'] &=& \delta_i \oplus sk_n[i'] \end{array}\right\} \implies x_n[i] \oplus x_n[i'] = sk_n[i] \oplus sk_n[i'] \tag{12}$$

Consequently, N ciphertext is sufficient to determine the subkeys of those words. More specifically, if the round function includes ℓ' distinct Sboxes instantiating and include exactly $t = \frac{\ell}{\ell'}$ calls to each distinct Sbox, the data complexity will be $\ell' \cdot (1 - 2^{-m})^{-t \cdot n} \cdot h \cdot \mathrm{H}(h)$.

3.3 Infection-based countermeasures

In the presence of infection-based countermeasures, the attacker is unable to detect the ciphertexts, resulting in an ineffective fault. To assess whether a ciphertext is correct (and corresponds to an ineffective event), one way is to encrypt a specific input in the chosen-plaintext model using both normal and faulty encryptions. However, the chosen-plaintext scenario stands in contrast to the aim of PFA, which is to be executed without input control. This situation complicates the application of SIPFA when an infection-based countermeasure is utilized. We make use of statistical tests to mount the attack in the ciphertext-only scenario. Similarly to statistical attacks, we decrypt the ciphertexts partially and determine the distribution of some intermediate values by guessing the involved subkeys. To narrow the candidates for the involved subkeys, we need to associate

Algorithm 2 Key-recovery on a Feistel cipher with a detection-based countermeasure, assuming that δ_i and i are unknown

```
Require: Ineffective ciphertexts $(C_i^1, \ldots, C_i^N)$ that are all generated using the same master key
    $K$ and a faulty Sbox $S_i^*$ in which $S_i[\delta_i] \leftarrow S_i[\delta_i] \oplus v_i$.
Ensure: The candidates for $\kappa$ bits of $sk_n$
 1: for $i \leftarrow 1$ to $\ell$ do
 2:     $\mathcal{X}[i] = \{0, \ldots, 2^m - 1\}$
 3: for $j \leftarrow 1$ to $N$ do
 4:     Get a correct ciphertext $C_i^j$
 5:     $x_n \leftarrow F_{n,I}((C_i^j)^L)$
 6:     for $i \leftarrow 1$ to $\ell$ do
 7:         $\mathcal{X}[i] \leftarrow \mathcal{X}[i] / x_n[i]$
 8: for $e \leftarrow 1$ to $\ell$ do
 9:      $\mathcal{X}[e] \neq \emptyset$ then $i \leftarrow e$              // $i$ gives the faulty Sboxs number
10: $\mathcal{X}[i] = \{0, \ldots, 2^m - 1\}$
11: for $k \leftarrow 0$ to $2^\kappa - 1$ do
12:     for $j \leftarrow 1$ to $N$ do
13:         $x_{n-1}[i] \leftarrow F_{n-1,I}((F_n(C_i^j)L, K) \oplus (C_i^j)^L)[i]$
14:         $\mathcal{X}[i] \leftarrow \mathcal{X}[i] / x_{n-1}[i]$
15:     if $\mathcal{X}[i] \neq \emptyset$ then
16:         return $K$                                  // It determines $\kappa$ bits of $sk_n$
```

a statistical scoring function. Squared Euclidean Imbalance (SEI) is a statistical scoring function that is defined as follows:

$$SEI(k) = \sum_{x \in \mathcal{X}} (p(x) - \theta(x))^2 \tag{13}$$

where, θ denotes uniform distribution which is 2^{-m} in our analysis. With $p(x)$ and θ being relatively close, the capacity $Cap(p, \theta)$ is defined as follows:

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

The required number of ciphertexts to get the success probability P_S of having the correct candidate among the first 2^a candidates out of all 2^κ candidates, can be estimated as follows [DEK+18, BGN12]:

$$N \approx \frac{\beta \cdot \Phi_{0,1}^{-1}(\alpha)}{C(p, \theta)} \qquad \text{for } P_\mathrm{S} = 0.5. \tag{14}$$

where $\alpha = 1 - 2^{-a}$ and $\beta = 2 \cdot m$ (when the target is a m-bit word). In this paper, we use SEI to rank candidates whenever applicable.

3.3.1 Known fault value and location

Let us assume that the attacker knows δ_i and i and N ciphertexts $C_i^1, \ldots, C_i^N$ are given. The adversary can determine $x_n[i]$ for each of the given ciphertexts and then count the occurrences of each possible value at $x_n[i]$. We expect the correct guess of $\delta_i \oplus sk_n[i]$ appears less frequently than other values at $x_n[i]$. More precisely, δ_i will never appear at the input of the faulty Sboxes for the correct ciphertexts, but will occur with a probability of 2^{-m} for the faulty ciphertext. Hence, the expected counter of the $x_n[i]$ corresponding to $\delta_i \oplus sk_n[i]$ is $N \cdot (1 - \Pi_\mathsf{c}) \cdot 2^{-m}$ while for any other value at $x_n[i]$ the counter is expected to be $N \cdot (\Pi_\mathsf{c} \cdot \frac{1}{2^m - 1} + (1 - \Pi_\mathsf{c}) \cdot 2^{-m})$. For large N, we expect that the counter's value for the target $\delta_i \oplus sk_n[i]$ be the minimum value. Given δ_i and the candidate(s) of $\delta_i \oplus sk_n[i]$, the possible value(s) of $sk_n[i]$ can be determined. The attack can be repeated to determine other words of sk_n as well. Given sk_n, it is possible to decrypt one round and repeat the attack using the gathered data to determine sk_n. The details of the procedure is represented in Algorithm 3.

Algorithm 3 Key-recovery on a Feistel cipher with an infection-based countermeasure, assuming that δ_i and i are known for $1 \leq i \leq \ell$

Require: Ciphertexts $(C_i^1, \ldots, C_i^N)$, for $1 \leq i \leq \ell$, all generated using the same master key K.
Ensure: candidates for $sk_n, \ldots . sk_1$

```
1: for $i \leftarrow 1$ to $\ell$ do
2:     for $h \leftarrow 0$ to $2^m - 1$ do
3:         $cnt[h] = 0$
4:     for $j \leftarrow 1$ to $N$ do
5:         Get a ciphertext $C_i^j$
6:         $x_n[i] \leftarrow F_{n,I}((C_i^j)L)$
7:         $cnt[x_n[i]] = cnt[x_n[i]] + 1$
8:     return $sk_n[i] = min(cnt[j]) \oplus \delta_i$          // It determines $n$ bits of $sk_n$
9: for $i \leftarrow 1$ to $\ell$ do
10:    for $j \leftarrow 1$ to $N$ do
11:        $(X_i^j)_{n-1}^L \leftarrow (C_i^j)^L$
12:        $(X_i^j)_{n-1}^L \leftarrow (C_i^j)^R \oplus F_n((C_i^j)^L, sk_n)$
13: for $r \leftarrow n-1$ down-to 1 do
14:    for $i \leftarrow 1$ to $\ell$ do
15:        for $h \leftarrow 0$ to $2^m - 1$ do
16:            $cnt[h] = 0$
17:        for $j \leftarrow 1$ to $N$ do
18:            $x_r \leftarrow F_{r,I}((X_i^j)_r^L)$
19:            $cnt[x_r[i]] = cnt[x_r[i]] + 1$
20:        $sk_r[i] = min(cnt[j]) \oplus \delta_i$          // It determines $n$ bits of $sk_r$
21: return $sk_n, \ldots, sk_1$
```

3.3.2 Unknown fault value and location

If δ_i and i are unknown, the adversary should first determine the location of the fault, i.e. i. As with Section 3.3.1, we expect the correct value of $\delta_i \oplus sk_n[i]$ appears less than other values at $x_n[i]$. It leads to a biased distribution in $x_n[i]$ and a uniform distribution in other words of x_n. Hence, we can compute $x_n^j[i]$, for $1 \leq j \leq N$ and $1 \leq i \leq \ell$, as well as the distribution of each words in x_n. Given these distributions in hand, we can use SEI to identify the biased word of x_n that corresponds to the fault location. After finding the location of the fault, the adversary should determine the fault value δ_i. The adversary guesses the involved subkeys to decrypt over one round and determine the distributions of $x_{n-1}[i]$ for each guess. The correct guess results in a biased distribution at $x_{n-1}[i]$, and a wrong guess results in a uniform distribution at $x_{n-1}[i]$. Hence, we rank the distributions corresponding to each guess using SEI, and select the candidates with the highest SEI as possible candidates of the fault value and the involved bits of subkey(s). Algorithm 4 illustrates this method to determine the fault value and location and recover the bits of subkeys.

It is worth noting the expected value of SEI for the wrong guess of the faulty Sbox is zero, while for the correct guess, it is a non-zero value that can be estimated as follows:

Algorithm 4 Key-recovery on a Feistel cipher with an infection-based countermeasure, assuming that δ_i and i are unknown

Require: Ciphertexts $(C_i^1, \ldots, C_i^N)$ all generated using the same master key K and a faulty Sbox S_i^* in which $S_i[\delta_i] \leftarrow S_i[\delta_i] \oplus v_i$.
Ensure: candidates for κ bits of sk_n

```
1: for $i \leftarrow 1$ to $\ell$ do
2:     $SEI[i] = 0$
3:     for $h \leftarrow 0$ to $2^m - 1$ do
4:         $cnt[i][h] = 0$
5: for $j \leftarrow 1$ to $N$ do
6:     Get a ciphertext $C_i^j$
7:     $x_n \leftarrow F_{n,I}((C_i^j)^L)$
8:     for $i \leftarrow 1$ to $\ell$ do
9:         $cnt[i][x_n[i]] = cnt[i][x_n[i]] + 1$
10: for $i \leftarrow 1$ to $\ell$ do
11:     for $h \leftarrow 0$ to $2^m - 1$ do
12:         $q[i][h] = \frac{cnt[i][h]}{N}$
13:         $SEI[i] = SEI[i] + (q[i][h] - 2^{-m})^2$
14: return $i$ such that $SEI[i]$ is maximum        // $i$ determines the faulty Sbox
15: for $k \leftarrow 0$ to $2^\kappa$-1 do
16:     $SEI[k] = 0$
17:     for $h \leftarrow 0$ to $2^m - 1$ do
18:         $cnt[h] = 0$
19:     for $j \leftarrow 1$ to $N$ do
20:         $x_{n-1}[i] \leftarrow F_{n-1,I}((F_n(C_i^j)L, K) \oplus (C_i^j)^L)[i]$
21:         $cnt[x_{n-1}[i]] = cnt[x_{n-1}[i]] + 1$
22:     for $h \leftarrow 0$ to $2^m - 1$ do
23:         $p[h] = \frac{cnt[h]}{N}$
24:         $SEI[k] = SEI[k] + (p[h] - 2^{-m})^2$
25: return $k$ such that $SEI[k]$ is maximum        // It determines $\kappa$ bits of $sk_n$
```

$$\begin{aligned} SEI(k) &= (2^m - 1) \cdot ((\Pi_c \cdot \frac{1}{2^m - 1} + (1 - \Pi_c) \cdot 2^{-m}) - 2^{-m})^2 + ((1 - \Pi_c) \cdot 2^{-m} - 2^{-m})^2 \\ &= (2^m - 1) \cdot (\Pi_c \cdot \frac{1}{2^m - 1} - \Pi_c \cdot 2^{-m})^2 + (\Pi_c \cdot 2^{-m})^2 \end{aligned} \quad (15)$$

The expected SEI for the correct guess of the κ bits of sk_n is identical to Section 3.3.2.

4 Application of SIPFA on DES, 3DES and Camellia

In this section, we apply the introduced attacks in Section 3 on various well-known block ciphers, i.e., DES, 3DES, and the Camellia family.

4.1 Application on DES

To map a round of DES to Figure 1, we can consider the expansion layer in conjunction with the add round key as $F_{r,i}$ and the permutation P as the $F_{r,o}$. $S_1, \ldots, S_8$ forms the Sbox layer.

Assume that the fault is injected into $S_1[\delta_1]$. Each Sbox is called once during each round of the cipher. Since DES has 16 rounds, and the input length of Sbox is $m = 6$ bits, the probability of obtaining fault-free ciphertext on each encryption can be estimated as follows (based on Equation (6)):

$$(1-\frac{1}{2^6})^{16} = 0.777 \tag{16}$$

As a result, the average rate at which a detection-based countermeasure returns a non-empty output is 77.7%. On average, an infection-based countermeasure generates correct ciphertexts and random ciphertexts by the probability of 77.7% and 22.3%, respectively.

Assuming that the fault's value and location are known and a detection-based countermeasure is used, we can use Algorithm 1 to determine sk_{16}. Following Equation (10), to achieve the only minimum value at the last round of DES the value of N is $64 \cdot H(64) = 64 \cdot 4.74 \approx 304$ expected correct ciphertexts. As a result, we can identify six bits of the last round's key given the fault value and almost 304 ciphertexts that are not impacted by the fault. The rate of fault-free ciphertext in this framework, on the other hand, is 77.7%. As a result, the overall complexity is about 390 DES calls, following Section 3.1. Following Algorithm 1, we may apply fault on other Sboxes also, i.e., $S_2, \dots, S_8$, and discover sk_{16} step by step to determine whole sk_{16}. The attack on average has a total complexity of $8 \cdot 390 = 3120$ calls to DES. It is also feasible to obtain sk_{15} and the master key using sk_{16} and those ciphertexts. As a result, obtaining the DES master key using a detection-based countermeasure requires 3120 calls to DES under eight faults, assuming we know the fault values. This attack has negligible time complexity.

To relax the fault model we consider Algorithm 2, where we assumed the attacker does not know where the fault is injected or even which Sbox is defective. We count the possible values at $x^j_{16}[1], \dots, x^j_{16}[8]$ for $1 \le j \le N$, given the N ciphertexts. The $x^j_{16}[i]$ that includes a minimum value, can determine the faulty Sbox. Let us assume that S_1 is the faulty Sbox. Following Algorithm 2, we guess the active bits of sk_{16} to determine $x^j_{15}[1]$ for $1 \le j \le N$. Given that bits 32, 1, 2, 3, 4, and 5 determine the input of S_1 and those bits of $x_{15}[1]$ are mapped to bits 25, 17, 7, 20, 21, and 23 of y_{16} in DES, therefore, we need to estimate 36 bits of sk_{16} to identify $x^j_{15}[1]$ values. We do not expect to see a minimum value at $x_{15}[1]$ if we guess those bits incorrectly, but the correct guess always returns a minimum value at $x_{15}[1]$. The chance of an incorrect guess to return a minimum value is calculated as follows:

$$p_w = 64 \cdot (1-\frac{1}{2^6})^N \tag{17}$$

For $N = 2000$, we have $p_w = 2^{-39.44}$. Hence, given 2000 fault-free ciphertexts, which costs 2574 calls to DES, we can determine 36 bits of sk_{16} uniquely. The rest of the key bits, i.e., 20 bits, can be found by exhaustive search. The total time complexity on average is $2000 \cdot 2^{36} \cdot \frac{1}{16} + 2^{20} \approx 2^{42.97}$ DES computations.

When an infection-based countermeasure is utilized, we can follow Algorithm 3 (resp. Algorithm 4) if the fault's value and location are known (resp. unknown) and SEI can be used to determine the rank of the candidate subkey's.

4.2 Application on 3DES

Due to the fact that 3DES utilizes the same Sboxes as DES, the attack may simply be extended to this cipher as well. However, because the cipher has 48 rounds, the chance of obtaining a fault-free ciphertext for each encryption is as follows (based on Equation (6)):

$$(1-\frac{1}{2^6})^{48} = 0.47 \tag{18}$$

As a result, if the aim is to use Algorithm 1 or Algorithm 2, the average rate of returned values by a detection-based countermeasure is 47%. In an infection-based countermeasure, where we can use Algorithm 3 or Algorithm 4, the rate of accurate ciphertexts is 47% and the rate of random ciphertexts is 53%, on average.

Assuming that the fault's value and location are known, given the fault value and approximately 304 ciphertexts that are not impacted by the fault, using Algorithm 1 we may deduce 6 bits of the last rounds key, resulting in a total complexity of almost 647 3DES calls. We may also apply faults on other Sboxes, i.e., $S_2, \dots, S_8$, and discover sk_{48} step by step. The attack on average has a total complexity of $8 \cdot 647 = 5176$ calls to 3DES. It is also feasible to obtain sk_{47}, sk_{46}, $\dots$ similarly. Assuming that we know the fault values, the overall complexity of getting the master key of 3DES with detection-based countermeasure on average is 5176 calls to this cipher under 8 faults. The time complexity of this attack is likewise negligible.

Given N ciphertexts and no knowledge of the fault location or even the defective Sbox, we may use Algorithm 2 to compute $x^j_{48}[1], \dots, x^j_{48}[8]$, for $1 \leq j \leq N$, and identify the faulty Sbox. Assuming that S_1 is faulty, we use $x^j_{47}[1]$, for $1 \leq j \leq N$, to identify 36 bits of sk_{48} .Given a unique candidate for 36 bits of sk_{48}, finding sk_{47} using $x^j_{46}[1]$ for $1 \leq j \leq N$ requires only 2^{20} guesses. In this method, the 56 bits of the used key at the outer DES are determined on average with a complexity of $2000 \cdot 2^{36} \cdot \frac{1}{48} + 2000 \cdot 2^{20} \cdot \frac{2}{48} \approx 2^{41.38}$. The outer DES is then inverted for all available ciphertexts. Then, for $1 \leq j \leq N$, we may apply the same technique as for the outer DES to calculate 36 bits of sk_{32} based on $x^j_{31}[1]$. Given a unique candidate for the 36 bits of sk_{32}, the complexity of identifying sk_{31} using $x^j_{30}[1]$ for $1 \leq j \leq N$ is around only 2^{20} guesses. In this method, the 56 bits of the involved key at the middle DES is found with a complexity of $2000 \cdot (\frac{16}{48} + 2^{36} \cdot \frac{1}{48} + 2^{20} \cdot \frac{2}{48}) \approx 2^{41.38}$. We similarly can determine the involved key in the first DES with an average complexity of $2^{41.38}$, if it is distinct from the previous keys. As a result, the overall complexity of getting 3DES full keys is $3 \cdot 2^{41.38}$, and the data complexity is 4255 calls to 3DES.

If the fault's value and location are known (resp. unknown), we may use Algorithm 3 (resp. Algorithm 4) to calculate the rank of the candidate subkeys when an infection-based countermeasure is applied.

4.3 Application on Camellia

When utilizing 128-bit keys, Camellia has 18 rounds, and when using 192- or 256-bit keys, it has 24 rounds. To map a round of Camellia to Figure 1, we can use the expansion add round key of the cipher, along with the whitening key of the last round, as $F_{r,i}$ and the P-function as $F_{r,o}$. The Sbox layer is made up of $S_1, S_2, S_3, S_4, S_2, S_3, S_4, S_1$.

Assume the fault's value and location are known and the fault is injected on the $S_1[\delta_1]$. Given that each Sbox is called twice in each round of the cipher, Camellia-128 has 18 rounds, and the Sboxs input length is 8 bits, the probability of achieving fault-free ciphertext for each encryption is (based on Equation (6)):

$$(1 - \frac{1}{2^8})^{18 \cdot 2} = 0.8686 \tag{19}$$

and it is 0.8287 for Camellia-192/256. As a result, the average rate of returned values by a detection-based countermeasure for Camellia-128 (resp. 192/256) is 86.86% (resp. 82.877%).

When we use Algorithm 1 or Algorithm 2, the expected number of ciphertexts N to observe exactly the proper minimum value in the input of each S_1 at round 18 (resp. 24) of Camellia-128 (resp. 192/256) is $N = 256 \cdot H(256) = 256 \cdot 6.1243 \approx 1572$, according to Section 4. Hence, given the fault value and nearly 1572 ciphertexts that are not influenced by the fault, we can use Algorithm 1 to identify 16 bits of Camellia-128's $sk_3 \oplus sk_{18}$ and $sk_3 \oplus sk_{24}$. In this framework, the rate of fault-free ciphertext is 86.86% (vs. 82.87%). As a result, there are around 1810 (resp. 1898) Camellia-128 (resp. 192/256) calls in total. We may also apply fault to other Sboxes, such as S_2, S_3 and S_4, to gradually discover $sk_3 \oplus sk_{18}$ (resp. $sk_3 \oplus sk_{24}$). The attack has (on average) a total complexity of $4 \cdot 1810 = 7240$ (resp. 7592) calls to Camellia-128 (resp. 192/256). Using $sk_3 \oplus sk_{18}$ and those ciphertexts, it is also possible to derive $sk_4 \oplus sk_{17}$ of Camellia-128 and the master key. Similarly, the master key of Camellia-192/256 can be determined step by step. Hence, utilizing a detection-based countermeasure and following Algorithm 1 to retrieve the Camellia-128 (resp. 192/256) master key requires 7240 (resp. 7592) calls to Camellia-128 (resp. 192/256) under 4 distinct faults, assuming we know the fault values and location. Likewise, this attack has negligible time complexity.

To use Algorithm 2, we relax the fault model and assume that the attacker does not know the fault value or the faulty Sbox. Following the proposed approach in Algorithm 2, given the N ciphertexts the values of $x^j_{18}[1], \ldots, x^j_{18}[8]$ are computed for $1 \le j \le N$. Following the structure of Sbox layer, we should have minimum values on one of the pairs $(x_{18}[1], x_{18}[8])$, $(x_{18}[2], x_{18}[5])$, $(x_{18}[3], x_{18}[6])$ or $(x_{18}[4], x_{18}[7])$ which is enough to identify the faulty Sbox. Let's assume that S_1 is the faulty Sbox, the minimum values of $y_{18}[1]$ and $y_{18}[8]$ should be the same and we have the minimum values of $x_{18}[1]$ and $x_{18}[8]$. Hence, we can retrieves 8 bits of the secret key, i.e., $sk_3[1] \oplus sk_{18}[1] \oplus sk_3[8] \oplus sk_{18}[8]$. Next, We guess $sk_3[1] \oplus sk_{18}[1]$, $sk_3[4] \oplus sk_{18}[4]$, $sk_3[5] \oplus sk_{18}[5]$, $sk_3[6] \oplus sk_{18}[6]$ and $sk_3[7] \oplus sk_{18}[7]$ to determine $x^i_{17}[8]$ for $1 \le j \le N$,which costs 40 bits guessing. For wrong guesses, we do not expect to observe a minimum value at $x_{17}[8]$, but the correct guess always returns a minimum value at $x_{17}[8]$. The following formula is used to calculate the probability of an incorrect key returning a minimum value:

$$p_w = 256 \cdot (1 - \frac{1}{2^8})^N \tag{20}$$

We have $pw = 2^{-28.7}$ for $N = 6500$. As a result, given 6500 fault-free ciphertexts and 7483 calls to Camellia, the key candidates of 48 bits of $sk_3 \oplus sk_{18}$ can be reduced from 2^{48} to $2^{11.29}$. The remaining round key bits, $sk_3[2] \oplus sk_{18}[2]$ and $sk_3[3] \oplus sk_{18}[3]$ plus the remaining 11.29 bit entropy, i.e., 11.29+16=27.29 bits of $sk_3 \oplus sk_{18}$, can be guessed and the minimum value of $x^j_{17}[1]$ for $1 \le j \le N$ can be determined. A wrong guess will not return a minimum value at $x_{17}[1]$, but the correct guess always produces a minimum value at $x_{17}[1]$. The total time complexity of determining $sk_3 \oplus sk_{18}$ on average $6500 \cdot (2^{40} + 2^{27.29}) \cdot \frac{1}{18} \approx 2^{48.49}$ Camellia-128 computations. We can also partially decrypt $\mathcal{C}_1, \ldots, \mathcal{C}_N$ over one round with $sk_3 \oplus sk_{18}$ and use the same approach to retrieve $sk_4 \oplus sk_{17}$. The complexity of this step of the attack is also $2^{48.795}$, implying that finding Camellia-128's master key is on average $2^{49.49}$ and the data complexity is 7483 calls to the cipher. Camellia 192 and Camellia 256 have time complexities of $2^{49.67}$ and $2^{50.08}$, respectively, while the data complexity is 7844.

If an infection-based countermeasure is used, we can use Algorithm 3 (resp. Algorithm 4) to calculate SEI and rank the candidate subkeys.

5 Simulation Results

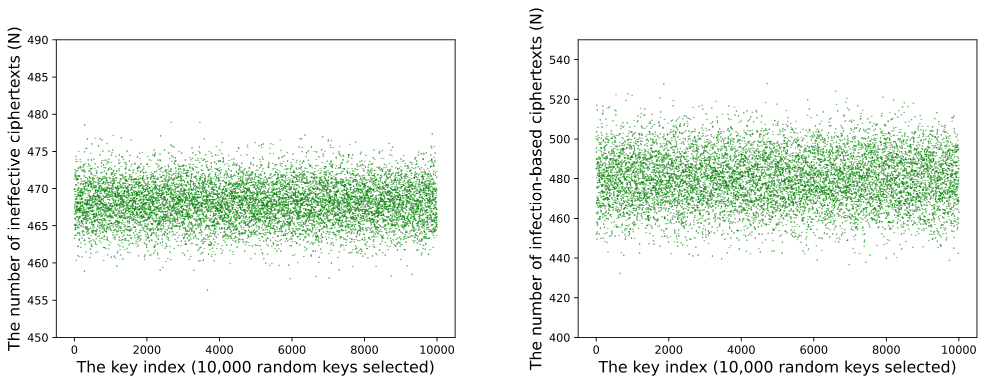
To evaluate the efficiency of the proposed framework, we performed extensive simulations of Algorithm 1 to Algorithm 4 by choosing DES as the target cipher. The results of each algorithm's simulation are discussed individually in the remainder of this section[2].

5.1 Simulation of Algorithm 1

In Algorithm 1, we assume that the fault location is known and attempt to recover $sk_n, \cdots, sk_1$ in the detection-based countermeasure scenario. We generated 10,000 random keys. Then the average number of ineffective ciphertexts required to retrieve the key is calculated for each of the selected keys. The average is taken for 100 different sets of random ineffective ciphertexts. Figure 2a illustrates the details of this simulation. The

[2]The source code for our simulations in `C` language is publicly available at `https://github.com/sadeghi87/SIPFA.git`

horizontal axis represents the selected random key's index, and the vertical axis shows the average number of ineffective ciphertexts required to recover the chosen key.



(a) The simulation of Algorithm 1 on DES **(b)** The simulation of Algorithm 3 on DES.

Figure 2: Distribution of the required number of ciphertexts to determine a word of sk_n, when the fault location and value are known

5.2 Simulation of Algorithm 2

In Algorithm 2, we assume that the fault location and its value are unknown to the adversary, and the purpose is to recover the κ bits of sk_n for a detection-based countermeasure. To simulate Algorithm 2, we assumed that only 12 bits of the subkey sk_{16} were unknown, and we generated N detection-based countermeasure ciphertexts, all of which were obtained using the same master key and a faulty Sbox. We repeated the experiments 100 times for each N to obtain the average number of key candidates. Figure 3 illustrates the results related to the simulation of Algorithm 2. N_0 and nk_0 denote the number of ineffective ciphertexts and the average number of key candidates, respectively.

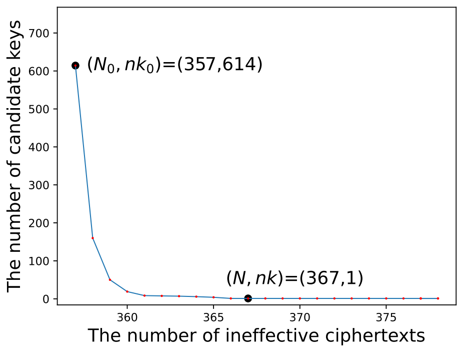
5.3 Simulation of Algorithm 3

To verify Algorithm 3, around 10,000 random keys were selected. Then for each of the selected keys, we repeated Algorithm 3 for 100 different sets of random ciphertexts, and computed the average number of ciphertexts required to recover the key. Figure 2b illustrates the details of this simulation. The horizontal axis represents the index of the selected random key, while the vertical axis represents the average number of ciphertexts required to recover the secret key.

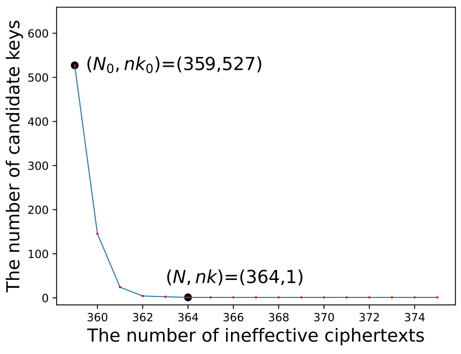
5.4 Simulation of Algorithm 4

We performed the simulation of Algorithm 4 in two parts: first, we simulated steps 1 to 14 to check whether the algorithm would correctly detect the fault location or not. For this purpose, one of the Sboxes is randomly selected as the faulty Sbox, and then by increasing the value of N (the number of infection-based ciphertexts). As shown in Figure 4, the faulty Sbox in both tests is correctly identified by increasing the value of N.

In the second part, we simulated Steps 15 to 25 in Algorithm 4 to verify the key-recovery attack. Due to time constraints, we assumed only 12 bits of sk_{16} were unknown. We repeated the key-recovery attack for 100 random secret keys with $N = 3000, 3500, \cdots, 20000)$ and counted the average number of the ranks of the candidate keys. Figure 5 illustrates the results for the different numbers of N.

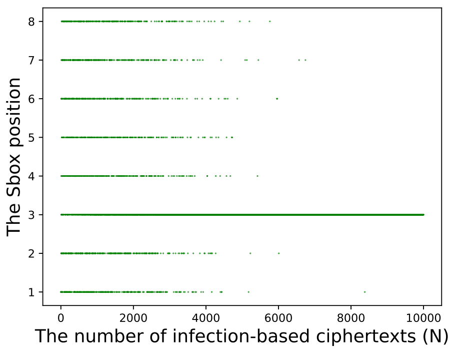


(a) Fault in S_1.

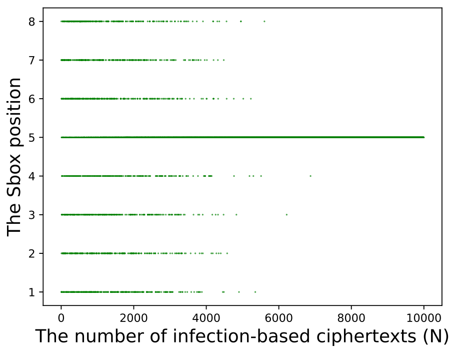


(b) Fault in S_6.

Figure 3: The simulation of Algorithm 2. The simulation is performed independently when the fault is injected in S_1 and S_6.



(a) The fault's location detection, when the fault is injected in S_3



(b) The fault's location detection, when the fault is injected in S_5

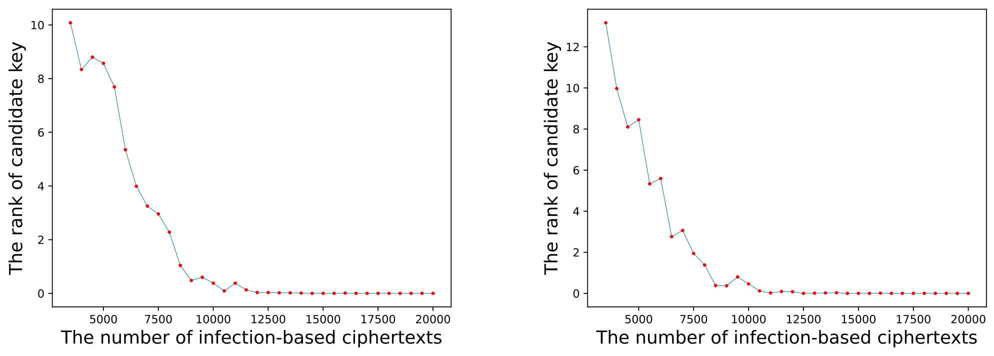
Figure 4: The simulation of the first part of Algorithm 4 for determining the fault location in DES in infection-based assumption

6 Experimental Results

This section reports the results of practical electromagnetic fault injections (EMFI) performed on 3DES. EMFI offers the advantage of non-invasive manner to inject faults compared to other approaches like laser fault injection where chip decapsulation might be required, while providing a good precision in location and timing as compared to voltage/clock glitching. We investigate the feasibility of single persistent fault injection in Sboxes of Feistel ciphers when executed on modern microcontrollers and corresponding key recovery complexity.

6.1 Experimental Platform

The EMFI platform used for the following experiments comprises of three main components: (1) a high-voltage pulse generator capable of generating pulses up to 200V (in either polarity) with a very low rise time under 4ns; (2) a hand-crafted electromagnetic probe designed as a simple loop antenna; and (3) a motorised XYZ table to position the probe over the device under test (DUT). An optional oscilloscope is also used for verification of pulse strength and timing characteristics. A software controls the DUT and the EMFI setup. It triggers



(a) Fault in S_4 **(b)** Fault in S_8

Figure 5: Simulation results of Algorithm 4 on DES. The simulation is performed independently when the fault is injected in S_4 and S_8.

the pulse injection synchronised with a feedback signal from the DUT. For automation of the whole system and additional relay switches are used for automated power-on reset of the DUT.

The target board for the following experiments is a STM32F4DISCOVERY evaluation board, where the prime target is a 32-bit ARM Cortex-M4 microcontroller (STM32F407VG). The microcontroller executes a publicly available 3DES implementation[3], compiled using the `arm-none-eabi-gcc` compiler with optimization level-`O3`. The communication with the DUT is done using UART. The flash configuration is performed through OpenOCD framework and on-chip hardware debugging is done with GNU debugger for ARM (`arm-none-eabi-gdb`). ST-LINK/v2.1 add-on board enables these communications. The faults target the boot-up time Sbox transfer from flash to RAM, similar to other works reporting practical PFA [MBD+19, SBH+21].

6.2 Experimental Results

We consider a scenario where the target cryptographic software utilizes the Sbox that is stored as part of the program code within flash memory. Upon power up, the Sbox is retrieved from flash and is stored in a designated location in the main memory (RAM). The software subsequently utilizes the Sbox stored in RAM for multiple executions of the encryption procedure. This approach is particularly advantageous in embedded microcontrollers, as it leads to reduced Sbox access delays, thereby improving the overall performance. If an attacker is able to fault the transfer of Sbox from flash to RAM, then it leads to a persistent fault in the Sbox, since the same faulted Sbox is used for multiple encryption calls. A similar fault model has been reported by Menu et al. [MBD+19], where they demonstrated the ability of EMFI to inject precise bit-set and bit-reset faults on data retrieved from the flash memory. Current scenario of targeting Sbox transfer from flash to RAM at boot-up is motivated by the public implementations we target in the following. Different implementations would need different strategy to achieve persistent faults in the Sbox.

While injecting a persistent fault in cryptographic software has been previously demonstrated in several previous works [ZLZ+18, ZZJ+20, MBD+19, SBH+21], all these experiments were performed on SPN block ciphers. When moving to Feistel ciphers like DES, 3DES and Camellia, few challenges arise. Unlike previously studied SPN ciphers, DES/3DES/Camellia have multiple distinct Sboxes. This would require achieving a precise control on the timing of the fault to inject faults in different Sboxes for different set of

[3]https://github.com/lbeatu/The-TRIPLE-DES-Algorithm-Illustrated-for-C-code.git

experiments. Moreover, an optimal way to exploit multiple persistent faults is not yet known for Feistel ciphers. In this case, a precise control over the position of the probe is required to trigger faults which affect up to 4/4/8 bits (DES/3DES/Camellia) of a single Sbox entry in a 32-bit transfer from flash to RAM over the data bus. Lack of precise control will lead to multiple faults which are known to be more likely than single faults when considering similar platforms [MBD+19, SBH+21].

For our experiments, we consider the 3DES cipher with 8 distinct Sboxes of size 6×4. Upon power up, our target implementation of 3DES loads the Sbox from flash memory into registers using multiple 32-bit LDR.W load instructions in an iterative manner, and subsequently moves them to a designated location in RAM using multiple 32-bit STR.W store instructions. For fault injection, we fix the voltage of the EM pulse to about 190 v and the pulse width to 7 ns (nanoseconds), as we are able to observe reliable faults with these parameters. We then perform a detailed fault injection campaign, sweeping over the entire surface of the target chip as well different injection delays to achieve different types of faults over the Sbox values.

We recall that our attack requires to independently fault single entries of all the eight distinct Sboxes of the 3DES cipher, unlike a single Sbox in other SPN ciphers like AES, PRESENT [ZZJ+20]. Thus, our main focus is to evaluate the practicality of faulting single entries of all the 8 Sboxes, independently. Once a particular Sbox is faulted (say S_1), corresponding faulty ciphertext are collected, followed by a reset of the chip and injection of fault into another Sbox (say S_2) and so on. Moreover, we only focus on the number of faulted Sbox entries, since the exact value of the faulted Sbox entry is not relevant for our analysis. We refer to the fault on a single SBox entry as a *single fault* while any fault affecting more than one SBox entry as *multiple faults*. Note that our analysis mainly requires single fault for an effective key recovery. Refer to Figure 6(a) for the distribution of single versus multiple faults observed from our detailed fault injection campaign. Among all the observed faults, the number of multiple faults was $\approx 17\times$ more than single faults. This correlates well with the work of Soleimany et al. [SBH+21] who also observed a very similar distribution of single versus multiple Sboxes when faulting Sboxes of SPN block ciphers. However, we observed a few sweet spots with respect to probe location and injection timing, where single faults on the different Sboxes were obtainable with a reasonably high repeatability. The repeatability was as high as 80% for certain Sboxes.

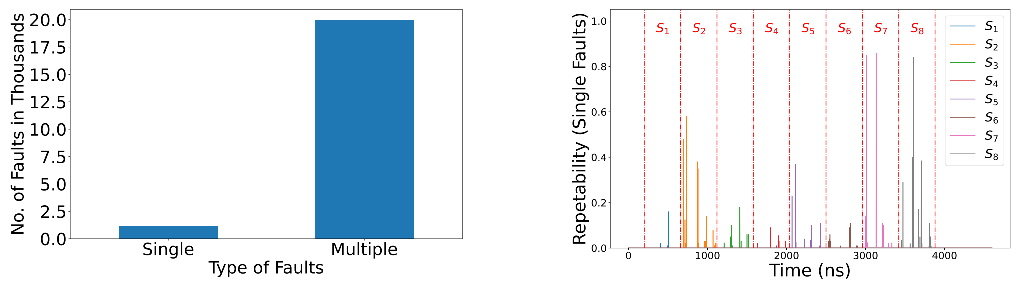


Figure 6: (a) Distribution of no. of faulted entries of the Sbox - Single vs Multiple Faults (b) Repeatability of single faults on the distinct Sboxes of 3DES over time

Refer Figure 6(b) for the ability to independently achieve single faults on all the eight distinct Sboxes with respect to the time of fault injection. We can see that single faults on all the Sboxes occur at distinct non-overlapping time windows. Thus, an attacker who can precisely control the timing of fault injection can achieve single faults on a targeted Sbox, while not affecting the other Sboxes. We also observe that the ability of obtaining single faults varies based on the targeted Sbox. We observe high repeatability of about 70-80%

for single faults on S_2, S_7, S_8, while the repeatability of single faults drops to about 20% for S_3, S_4. We could probably achieve high fault repeatability for all the Sboxes with a more intensive profiling and a better EM fault injection setup.

Refer Figure 7 for the sensitive areas on the chip surface to EMFI to achieve single faults on all the eight distinct Sboxes of 3DES. Similar to the distinct time windows of fault injection for the different Sboxes, we also observe varied sensitivity of the probe location to single faults for the different Sboxes. Thus, an attacker with precise control over the probe location as well as injection timing can inject single faults into all the eight distinct Sboxes of 3DES. The same fault characteristics can also be obtained for other Feistel ciphers such as Camellia. However, the fault experiments for Camellia are not presented, due to its similarity with AES with 8×8 Sboxes which does not present any additional challenge with respect to EMFI. Also, in a memory efficient version of Camellia, the Sboxes S_2, S_3, S_4 can be simply calculated from S_1, thus needing only S_1 in RAM.

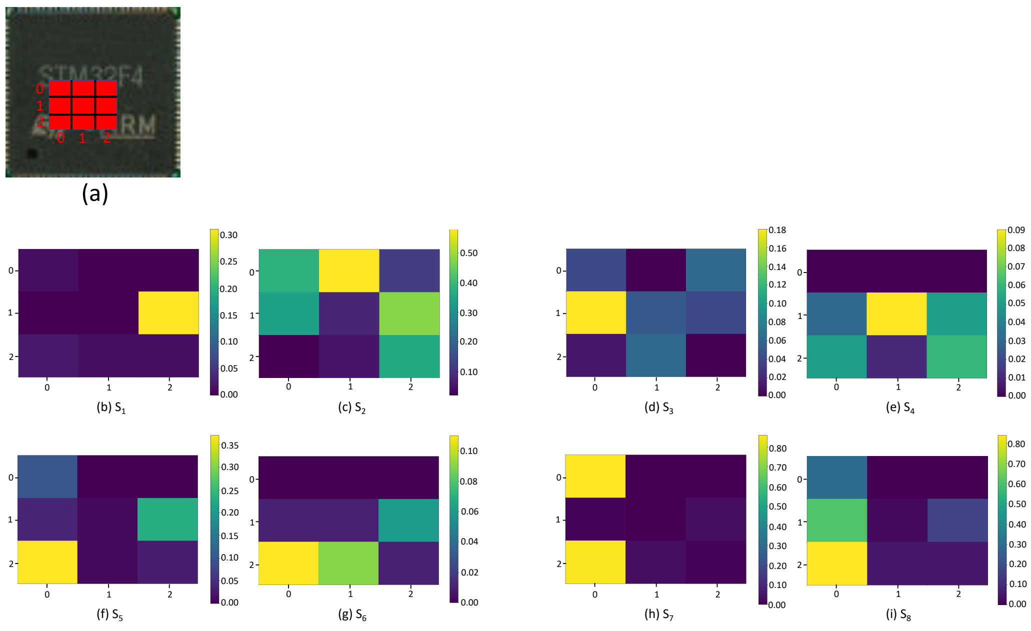


Figure 7: Sensitive area to EMFI for single faults (a) Physical location of sensitive spots to EMFI on the target chip (STM32F407VG) (b-i) Repeatability of Single Faults on the eight distinct Sboxes of 3DES

7 Conclusion

In this paper, we discussed critical limits on the application of PFA to Feistel ciphers and demonstrated how they might be overcome. We introduced Statistical Ineffective Persistent Faults Analysis (SIPFA), which enables us to apply PFA efficiently to Feistel ciphers in a range of scenarios. The proposed techniques were applied to DES, 3DES, and Camellia. SIPFA can be quite efficient against Feistel ciphers, as illustrated by the findings. Additionally, we conducted both simulations and real experiments on DES and 3DES using electromagnetic fault injection.

Acknowledgement

The authors wish to thank Florian Mendel and the anonymous reviewers for their detailed comments and helpful suggestions. Prasanna Ravi and Shivam Bhasin acknowledge the support from the Singapore National Research Foundation (“SOCure” grant NRF2018NCR-NCR002-0001 – www.green-ic.org/socure).

References

[AIK+00] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-bit block cipher suitable for multiple platforms—design andanalysis. In *International workshop on selected areas in cryptography*, pages 39–56. Springer, 2000.

[BBB+22] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Dirmanto Jap, and Dhiman Saha. A survey on fault attacks in symmetric key cryptosystems. ACM Computing Survey, March 2022.

[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.

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-) security of 64-bit block ciphers: Collision attacks on http over tls and openvpn. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 456–467, 2016.

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

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

[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.

[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.

[DES99] NIST DES. Fips publication 46-3-data encryption standard, 1999.

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

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

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

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

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

[PZRB19] Jingyu Pan, Fan Zhang, Kui Ren, and Shivam Bhasin. One fault is all it needs: Breaking higher-order masking with persistent fault analysis. In Jürgen Teich and Franco Fummi, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, pages 1–6. IEEE, 2019.

[S+77] Data Encryption Standard et al. Federal information processing standards publication 46. *National Bureau of Standards, US Department of Commerce*, 23, 1977.

[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.

[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.

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

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

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