Transform-and-Encode: A Countermeasure Framework for Statistical Ineffective Fault Attacks on Block Ciphers

Sayandeep Saha*, Dirmanto Jap†, Debapriya Basu Roy*, Avik Chakraborti‡, Shivam Bhasin†, and Debdeep Mukhopadhyay*
*Department of Computer Science and Engineering, IIT Kharagpur, India
† Physical Analysis & Cryptographic Engineering (PACE) Labs, Nanyang Technological University, Singapore
‡ NTT Secure Platform Laboratories, Tokyo, Japan
{sahasayandeep, dbroy, debdeep}@iitkgp.ac.in, {djap, sbhasin}@ntu.edu.sg, avikchkrbrti@gmail.com

Abstract—Right from its introduction by Boneh et al., fault attacks (FA) have been established to be one of the most practical threats to both public key and symmetric key based cryptosystems. Statistical Ineffective Fault Analysis (SIFA) is a recently proposed class of fault attacks introduced at CHES 2018. The fascinating feature of this attack is that it exploits the correct ciphertexts obtained during a fault injection campaign, instead of the faulty ciphertexts. The SIFA has been shown to bypass almost all of the existing fault attack countermeasures even when they are combined with provably secure masking schemes for side-channel resistance. The goal of this work is to propose a countermeasure for SIFA. It has been observed that a randomized domain transformation of the intermediate computation combined with bit-level error correction can throttle SIFA. The randomized domain transformation can be achieved by standard masking schemes. In fact, we prove that if biased faults are injected at the state register of a block cipher at a target round, then masking is sufficient to protect against SIFA, until all the shares for a specific bit are corrupted. However, masking alone cannot prevent SIFA if the faults are injected at certain specific locations inside the S-Boxes. To address this issue, we incorporate a bit-level error-correction mechanism. The strongest advantage of the proposed countermeasure, called AntiSIFA, is that it provides provable and quantifiable security guarantees. Proof-of-concept evaluations were performed on software implementations of the block cipher PRESENT, which correlates with the theoretical results.

Index Terms—Fault Attack, Block Cipher, Masking

1 Introduction

The pervasive use of embedded electronics has opened the avenue for various implementation-based attacks. Fault attacks (FA) are considered among the forerunner in the arsenal of these implementation attacks, which are capable enough to allow practical key recovery [1], [2]. FAs are a suite of active perturbation attacks, where errors are intentionally injected to create security vulnerabilities. These vulnerabilities can be of several types including bypassing critical verification steps, learn sensitive information etc. In context of ciphers, FA are widely deployed for practical key recovery attacks. Take AES-128 for example. The best known theoretical attack needs a brute-force of $\approx 2^{126}$ (which is not practical), whereas a single fault injection can bring the brute-force complexity to as low as 2^8 (which can be done on a standard computer). Thus, the practicality of FA motivates an adversary to invest in fault injection capabilities and furthermore, a designer to implement relevant protection measures.

Several different families of fault analysis techniques and the underlying fault model required have been introduced in literature [1]–[7]. The common feature among all the fault analysis techniques is that they utilize the statistical bias introduced by the fault into the cipher computation. Such statistical bias is often called as distinguishers as they filter out wrong key guesses and help in determining the correct key. The most common analysis technique is differential fault analysis (DFA) [2], [3], which exploits the fault differentials observed through correct-fault ciphertext pair. An automated version exploiting the power of SAT solvers also exist, known as algebraic fault analysis (AFA) [8]. Most of differential attacks assume a generic and localized random fault model, which makes the analysis extremely powerful. Moreover, the required number of faults are the lowest in DFA compared to other fault analysis techniques. The most common strategy to throttle DFA is to incorporate redundancy in computation [9], [10] to detect the presence of a fault. Various countermeasures have been proposed which use redundancy in either space, time [9] or information [11]. These countermeasures do result in a non-negligible overhead but can lead to an effective solution. The most commonly used one from this class is duplication followed by output matching. In case of a mismatch, the out-

put is either suppressed or randomized, preventing further analysis. Another popular class of DFA countermeasures are the so-called infection countermeasures which avoids the explicit detection of faults altogether, and incorporates a randomized infection function to "infect" (randomize) the computation upon the injection of a fault [10], [12].

The other family of FAs are rather based on statistically biased fault model. The biased model are linked to device physics where some faulty values occur more often than the other, like a bit reset ($1 \rightarrow 0$) is more likely than bit set ($0 \rightarrow 1$) when using overclocking based injection. Such biased faults have given rise to multiple simpler albeit effective classes of analysis mechanisms, namely Statistical Fault Analysis (SFA) [4], Differential Fault Intensity Analysis (DFIA) [5], Ineffective Fault Analysis (IFA) [13] etc. Most importantly, biased faults can evade several FA countermeasures as shown in [14], [15]. However, all these attacks are statistical in nature and requires a good number of faulty ciphertexts for the key recovery.

The most recent inclusion in the family of biased fault attacks is the so called Statistical Ineffective Fault Analysis (SIFA) [6]. SIFA combines the concepts of SFA attacks with that of the Ineffective Fault Analysis (IFA) proposed by Clavier [13]. As the name suggests, SIFA exploits ineffective faults. Ineffective faults are those which have no impact on the output. Let us take the previous example of bit reset ($1 \rightarrow 0$) being more likely compared to a bit set. Now if the initial value of the target bit is already 0, the fault has a higher chance of resulting in $0 \rightarrow 0$) rather than a ($0 \rightarrow 1$), having no impact on the target variable leading to correct computation. If any redundancy based countermeasure is in-place, it would not detect this fault and lead to a correct ciphertext. The adversary can build a statistical distinguisher based on probability of seeing a correct ciphertext, despite a (ineffective) fault, which leads to key recovery. A detailed overview of SIFA is given in the following sections.

Contrary, to earlier attacks, SIFA exploits correct ciphertext rather than faulty ciphertext, making it a direct threat to conventional FA countermeasures which are mostly based on the fundamental operation of detecting faults. It has been practically shown in [7] that SIFA can evade all existing FA countermeasures even while they are combined with provably secure side-channel (SCA) countermeasures like masking [16], [17] or threshold implementation (TI). In a different work, SIFA was exploited for breaking two authenticated encryption (AE) schemes [18]. All existing FA countermeasures, being fundamentally dependent on their fault detection step fails to detect SIFA. Moreover, SIFA, unlike certain other classes of biased FAs like DFIA, does not require an explicit knowledge on the nature of the bias present in the fault distribution.

The focus of this paper is to propose effective countermeasures against SIFA. We first study the mathematical foundation of SIFA to understand the root cause and in particular the underlying biased distribution which SIFA exploits. As a next step, we present a generic framework, referred to as *Transform-and-Encode* (TaE), for throttling SIFA. The proposed framework applies two basic primitives, namely *Domain Transformation* (or simply Transform) and Encode on any block cipher construction to make it SIFA protected, and is generic in the sense that the two aforementioned primitives can be realized by various means. There are two versions of SIFA which were previously proposed. The first version, further referred as SIFA-1, assumes that the fault is injected in the state variable and it is statistically biased. The later version, referred as SIFA-2, considers random faults in internal sub-operations of the cipher like substitution box (S-Box). We theoretically derive the conditions to prevent both versions of SIFA. As shown later, SIFA-2 is harder to prevent compared to SIFA-1. In fact, we found that the Transform primitive is sufficient to prevent SIFA-1, which can be realized by any standard masking scheme. On the other hand, protecting against SIFA-2 requires both Transform and Encode to be applied, where the Encode step can be realized by bit-level Error-Correcting-Codes (ECC). In summary, the contributions of this work are as follows:

- We propose the first generic framework TaE to counter SIFA. Unlike a protocol or system level countermeasure (e.g. re-keying or self-destruction upon observing a few faults), we take one of the most reasonable approach, that is to offer a cipher level countermeasure against FA or SCA.
- The Transform step in the proposed framework provides provable security against the SIFA-1 model. In fact, it is found that any secured masking scheme for SCA protection is a concrete realization of the Transform. **The realization of Transform can provide security against any multi-bit biased fault provided the fault corrupts a cipher state and it does not affect all shares of a bit, simultaneously.** From a practical perspective, this fault model is fairly reasonable. Furthermore, the realization of Transform with masking incurs zero-overhead for an implementation which is supposed to be protected against both SCA and SIFA-1.
- The SIFA-2 model exploits relatively precise faults within the S-Box computation to come up with successful attacks. In order to throttle SIFA-2, we incorporate bit-level error correction via the Encode step of the proposed framework. As a concrete example of the Encode, we use simple *Duplication Code* in a per-bit manner. The proposed ECC provides quantifiable security assurance with respect to the fault model. More concretely, to achieve t bit error correction we need to employ a code length of $2t + 1$, resulting in a overhead of $2t + 1$ times. Although the overhead may seem significant, it is still affordable, given its strong security guarantees. Observing the fact that a straightforward application of error correction could be fatal, we take necessary precautions to make it sufficiently robust. **More precisely, the error-correction operation we use is implemented in a fault-tolerant way, and is applied at precise places without causing any new vulnerability.**
- A proof-of-concept evaluation has been made on the PRESENT block cipher [19] which is a potential candidate for lightweight cryptography. Extensive evaluations have been made for the protected implementation which completely correlates with the theoretical results. For most of our experiments we

stick to simulated faults as they can give very precise idea about the capability of the countermeasure. To be precise, we are able to prevent the SIFA successfully.

The rest of the paper is organized as follows. In Sec. 2, we provide the necessary background on FAs in general with a focus on SFA and SIFA. We also briefly describe the PRESENT cipher [19], which is used for proof-of-concept evaluation. Sec. 3 presents the mathematical model for SIFA along with the necessary and sufficient conditions for preventing it. A sketch of our main idea is given in Sec. 4. The actual TaE framework is outlined next, in Sec. 5, followed by concrete realizations for both Transform and Encode operations. Sec. 6 describes the proof-of-concept implementation for PRESENT, with the experimental evaluations of our claims on it. Finally, Sec. 7 concludes the paper.

2 PRELIMINARIES

In this section, we provide the necessary background on FA. Given the present context, the main focus is on SIFA. Furthermore, a brief description of the PRESENT block cipher [19], which is utilized for a proof-of-concept evaluation, is provided.

2.1 Fault Attacks on Block Ciphers

Malicious exploitation of faults in the context of cryptography dates back to 1997 due to Boneh et. al., who first demonstrated FA on public key cryptosystems [2]. The concept of DFA was first introduced by Eli Biham and Adi Shamir for the Data Encryption Standard (DES) [1], and got readily extended for ciphers like Advanced Encryption Standard (AES) [20], PRESENT [21], LED [22]. So far, AES is the most studied one in the context of fault attacks. In particular, it has been demonstrated that a single random byte fault can recover the entire 128-bit key of AES [3] within minutes. The basic principle behind any FA is to exploit the information leakage caused by the malicious aberration in the normal execution of a cipher due to a fault. In an ideal situation, at any time, the intermediate state of a block cipher should be uniform (U_n) if random plaintexts are being encrypted. However, a computational fault may force the intermediate state to deviate from U_n (larger statistical distance). Interestingly, such statistical bias becomes visible only for the correct key (or possibly for a small set of candidate keys containing the correct key), and disappears for most of the wrong key choices. Using this property one can build so-called wrong-key distinguishers to identify the correct key. The concept of a wrong key distinguisher (or simply distinguisher) is fundamental in every fault attack proposed till date.

In DFA, a distinguisher is represented analytically, and a system of equations is solved, either to extract the key or to reduce the keyspace to a size where exhaustive search becomes trivial [3]. The low fault complexity and extremely relaxed fault model of DFA makes it the most preferred and explored attack strategy so far. However, the attack algorithm is specific to the structure of the target cipher and fairly challenging to discover. In [23], Saha et. al. presented a framework to construct such attack algorithms automatically, given any block cipher.

A significantly different approach is adapted for fault exploitation in the case of so-called SFA attacks [4]. The main crux of these attacks are the statistical bias present in the fault itself, resulting from the physical characteristics of the target device. The bias caused by the injected fault is exploited statistically in this case. More specifically, one can guess the partial last round key of a cipher and go back to the fault injection point by means of partial decryption of the faulty ciphertexts. Under the correct key guess the bias present in the fault distribution would be visible, whereas for a wrong key guess it would be very close to a uniform distribution. In general, statistical tests like Squared-Euclidean-Imbalance (SEI) are utilized to detect the bias and hence the correct key. One advantage of this attack strategy with respect to DFA is that the attack algorithm is fairly simple and generic. In fact, the correct ciphertext corresponding to a faulty one is not required in some cases [22]. However, the number of required faults are often significantly high due to the statistical nature of the attack. There exists several different flavours of this general strategy. One popular example is the DFIA [22], which utilizes a slightly different distinguisher based on the Hamming distance (HD) between two faulty intermediate states [22].

2.2 Statistical Ineffective Fault Analysis (SIFA)

The SIFA is classified as an instance of SFA where the correct ciphertexts are utilized for attack [6], instead of the faulty ciphertexts. The main observation here is that under a typical biased fault model, some of the injections fail to alter the value of the target intermediate state resulting in a correct output. As a simple example, consider a variable A over $\{0,1\}^4$ at some intermediate state of a block cipher. If a stuck-at-0 fault is injected at the Least Significant Bit (LSB) of A, the fault will corrupt only the values having 1 at the LSB position. In other words, it remains ineffective if $LSB(A) = 0$ and results in a correct ciphertext. One may observe that *the target intermediate A will assume only 8 possible values instead of 16.* This is indeed a statistical bias, and it becomes visible with the correct key guess if the correct ciphertexts are partially decrypted up to the fault injection point.

The most fascinating feature of SIFA is that it is somewhat agnostic to the type of bias caused due to fault injection. In practice, any biased intermediate distribution can be exploited for key recovery. Further, it was shown in [7], that the intended statistical bias can be caused even by a random fault if utilized carefully. More specifically, *if the computation of a bijective S-Box is corrupted in a way so that its bijectivity gets somehow hindered, the S-Box output assumes a biased distribution.* This typical fault model for corrupting S-Box computation is able to bypass state-of-the-art masking schemes used as SCA countermeasures. It has been shown in [7], that this specific fault model is easily realizable for software implementations. As already pointed out in the introduction, we clearly differentiate between these two fault models. In particular,

- The biased fault model corrupting the intermediate states is referred to as SIFA-1.
- The fault model corrupting the S-Box computation is referred to as SIFA-2.

As we shall show later in this paper, the requirements for countering these two fault models are quite distinct. Fortunately, the countermeasure framework we are going to propose can provide provable and quantifiable security against both of these models.

2.3 The PRESENT Cipher

PRESENT is a block cipher having a Substitution-Permutation-Network (SPN) based structure [19]. The most attractive feature of PRESENT is its extremely low hardware/software footprint, which establishes it as a strong candidate for lightweight cryptography. The original proposal describes two different versions namely, PRESENT -80 and PRESENT -128. However, for our purpose, we shall use the first one having a 64-bit block size, 80-bit master key and 64-bit key per round derived from the master key. The iterative construction of PRESENT -80 repeats a round function (consisting a key addition layer, confusion layer, and a diffusion layer), 31 times to generate the ciphertext. The key addition layer (`addRoundKey`) is the first operation in the round function of PRESENT , which is eventually followed by the substitution (confusion) layer and the diffusion layer. The substitution layer (`sBoxLayer`) consists of 16 identical 4×4 S-Boxes having good cryptographic properties. The diffusion layer (`pLayer`) of PRESENT is constructed with a simple bit-permutation operation. Note that the bit-permutation is highly favourable for lightweight hardware implementations as it requires only wirings to be made without any logic gates. The key schedule of the cipher generates total 32 round keys, each having 64-bit length, from a key state of 80 bits. The key schedule of PRESENT is not important in the current context and hence we do not describe it here. We refer to [19] for further details on the cipher.

PRESENT has been targeted previously with different classes of FAs. In [24] and [21], authors described DFA attacks on PRESENT . SFA attacks were proposed in [25] with faulty ciphertexts only. Another distinct class of attack, which utilizes both fault and power analysis was proposed in [26]. This new class of attack typically exploits the properties of the bit-permutation layer and is closer to the classical DFA attacks in principle. However, no result has been reported so far in the context of SIFA. From this perspective, our work presents the first analysis of a bit-permutation-based, lightweight block cipher with respect to SIFA. However, the results are very similar to the attacks on AES, in general.

3 Mathematical Analysis on SIFA

In this section, we formalize the main idea behind SIFA. The formalization presented here is subsequently utilized for proving the security claims made in this work. Before going to the details, we summarize the mathematical notations used in the rest of the paper in Table 1 for quick reference.

TABLE 1: List of Notations

Symbol	Definition
$"+"$	Bitwise XOR
n	Block size of a block cipher
$\mathcal{X}_g = \{0,1\}^g$	State space of g bits
X_g	Random variable over $\mathcal{X} = \{0,1\}^g$
X_g'	Random variable representing the faulty value over a state-space $\mathcal{X} = \{0,1\}^g$
x	Fixed valuation over $\mathcal{X}_g = \{0,1\}^g$
S	State of a block cipher (n bits).
S_g	Part of a block cipher state having g bits.
$S_g = \langle b_0, b_1, \cdots, b_{g-1} \rangle$	Bit-wise representation of S_g.
U_g	Uniformly distributed random variable over a given state-space $\mathcal{X}_g$
$\leftarrow$	Represents assignment.
$[x_1 \cdots x_{\lceil \lvert x \rvert / m \rceil}] \xleftarrow{m} x,$	Parse a string into m bit blocks. ($\lvert x_i \rvert = m$ for $1 \leq i \leq \lceil \lvert x \rvert / m \rceil$ and $\lvert x_{\lceil \lvert x \rvert / m \rceil} \rvert \leq m$.)

3.1 How to Represent Faults

With the informal introduction from the last section, now we construct a mathematical model for the SIFA. Let S be the intermediate state register of n bits. The fault injection is supposed to corrupt S (from its actual value) at a specific time stamp. Also, we assume that the attacker can control the timing of the fault. Controlling the fault timing is trivial for modern fault injection setups.

A crucial point here is how S gets corrupted by the fault injection. Most of the practical fault models in FA assume that the fault is localized within a specific region of the entire state, and repeatedly affects the same region. For example, one may assume that a specific nibble or byte within the state S is getting affected. It is well-established that localized faults are preferable for doing practical fault attacks. Following this trend we consider that the fault impact is limited within a contiguous chunk of w-bits (we call w as width) of the state register S[1]. Also, in practical fault attack experiments, even a localized fault may corrupt different parts of S at different injection events. However, the analyses must focus at a specific w-bit chunk and measure the bias at that point. The rest of the cases, where the fault impacts other places are considered as noise.

Let us denote the state space associated with the entire S as $\mathcal{X} = \{0,1\}^n$. Our analysis will be focused on a specific w-bit chunk from $\mathcal{X} = \{0,1\}^n$. For convenience, we denote the state space of interest as $\mathcal{X}_w = \{0,1\}^w$, and the part of S associated with it as S_w. However, not all the bits of S_w are equally likely to be affected by fault injection. In the most general and practical scenario, the fault may only impact a fixed subset of bits within S_w repeatedly. A pictorial illustration of such fault behaviour is provided in Fig. 1. Furthermore, even if a bit is impacted, it may or may not change its value depending on whether the fault is ineffective or not. One convenient way of modelling this whole scenario is to assign probabilities to each of the bits within S_w, depending on how it gets affected by the fault injection.

1. Localized faults are even more important in the case of statistical fault attacks. The reason is statistical fault attacks usually measure the deviation of a part of the intermediate state from uniform random distribution to distinguish the keys. If the fault is not localized the analysis would require to measure the entire S for its deviation from uniformity. This will become computationally expensive as the number of samples required could be formidable, and also one need to simultaneously guess a large number of key bits.

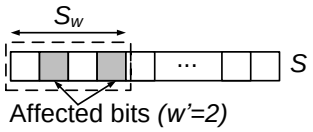

Fig. 1: State fault scenario in SIFA.

Let X_w denote a random variable associated with S_w, and assuming values from $\mathcal{X}_w$. Further, S_w can be represented as $S_w = \langle b_0, b_1, \cdots b_{w-1}\rangle$ with $b_i \in \{0,1\} \forall i$. We characterize the impact of the faults by transition probability of a value under the influence of a fault. To represent the faulty valuation of S_w, we use another random variable $X_w^{'}$, assuming values from the same state space $\mathcal{X}_w$. The transition probability of a value under the influence of a fault can be represented as:

$$p_x(x') = \mathbb{P}[X_w^{'} = x' \mid X_w = x] \tag{1}$$

Further, considering the bitwise representation of S_w, where each bit can be considered to be statistically independent, $p_x(x')$ can be further expressed as:

$$p_x(x') = \prod_{i=0}^{w-1} \mathbb{P}[b_i' = x_i' | b_i = x_i] = \prod_{i=0}^{w-1} p^i_{x_i \to x_i'}, \tag{2}$$

Here, for simplicity, we represent each bit of S_w and its associated random variable with the same symbol b_i (for the faulty case, the variables are $b_i^{'}$). It is also worth mentioning that x_i' ($\in \{0,1\}$) can either be equal to x_i or $1 \oplus x_i$. Since, in SIFA the transition probability of a bit b_i depends upon the value it assumes (i.e. x_i), the probability expressions in our case are parameterized with x_i's rather than b_i's. This slight abuse of notation helps us to keep the modeling simple.

As it has been already pointed out, given $S_w = \langle b_0, b_1, \cdots, b_{w-1}\rangle$, many of the bits are not supposed to get impacted by fault in an ideal noise-free case. For these bits, the valuation does not matter as they never come under the influence of the fault. We model this situation as follows:

$$p^j_{x_j \to x_j^{'}} = \begin{cases} 1, & \text{if } x_j = x_j^{'} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Here $p^j_{x_j \to x'_j}$ takes the value 1 if the j^{th}-bit of S_w does not get affected by the fault. To be precise, the probability of x_j, changing to a different valuation is 0,[2]. However, for the attacks to work, it is required that $\exists j \in \{0, \cdots, w-1\}$, such that $p^j_{x_j \to x_j^{'}} \neq 0$, for $x_j \neq x_j'$.

We now derive the necessary an sufficient condition for SIFA to work. As we shall show, this condition is going to be the key factor in preventing SIFA.

3.2 Condition for SIFA

Referring to Eq.(1) and (2), we can write $p_x(x) = \prod_{i=0}^{w-1} p^i_{x_i \to x_i}$, which denotes the transition probability of a state in the case of ineffective faults. For simplicity, we use $p^i_{x_i}$ instead of $p^i_{x_i \to x_i}$. Note that, for several $i \in \{0, 1, \cdots w-1\}$: $p^i_{x_i \to x_i} = 1$. Without loss of generality, consider that w' non-contiguous bits within a w-bit contiguous chunk ($w' \leq w$) get affected by the fault injection due to spatial locality. Assume the indices for these w' bits (confined within contiguous w bits) are $i_0, i_1, \cdots, i_{w'-1}$. Let $S_{w'} = \langle b_{i_0}, b_{i_1}, \cdots, b_{i_{w'-1}}\rangle$, and the two associated random variables representing the correct and faulty valuation of $S_{w'}$ are $X_{w'}$ and $X_{w'}^{'}$, respectively. The number of possible values of $S_{w'}$ thus reduces to $2^{w'}$ and we denote this reduced state space by $\mathcal{X}_{red}$. We also assume that $b_{i_0}, b_{i_1}, \cdots b_{i_{w'-1}}$ are uniform over $\{0,1\}$ and w'-wise independent. We would like to emphasize that SIFA works when the injected fault (on the w' bits) is ineffective for some specific state values. We would now like to state and prove the following theorem:

Theorem 1. *Statistical ineffective fault attacks happen if and only if* $\exists\, x, y \in \mathcal{X}_{red} : p^*_x(x) \neq p^*_y(y)$, *where* $p^*_x(x) = \prod_{j=0}^{w'-1} p^{ij}_{x_{ij}}$.

Proof. Let us first define the *ineffectivity rate* $\pi_=$ as $\pi_= = \mathbb{P}[X_{w'} = X'_{w'}]$. Clearly,

$$\begin{aligned} \pi_= &= \sum_{x \in \mathcal{X}_{red}} \mathbb{P}[X_{w'} = x].\mathbb{P}[X'_{w'} = x \mid X_{w'} = x] \\ &= \sum_{x \in \mathcal{X}_{red}} \frac{p^*_x(x)}{2^{w'}} = \sum_{x \in \mathcal{X}_{red}} \frac{\prod_{j=0}^{w'-1} p^{ij}_{x_{ij}}}{2^{w'}} \end{aligned} \tag{4}$$

Next, we derive the probability distribution of ineffective faults denoted by $p_=$ as:

$$\begin{aligned} p_=(x) &= \mathbb{P}[X'_{w'} = x \mid X'_{w'} = X_{w'}] = \frac{p^*_x(x)}{2^{w'}\pi_=} \\ &= \frac{\prod_{j=0}^{w'-1} p^{ij}_{x_{ij}}}{2^{w'}\pi_=} = \frac{p^*_x(x)}{\sum\limits_{x \in \mathcal{X}_{red}} p^*_x(x)} \end{aligned} \tag{5}$$

One important fact here is that if $p_=(x)$ becomes statistically indistinguishable from an uniform distribution $\theta(x)$ (that is the statistical bias is negligible) even for the correct key guess, then the key recovery becomes impossible. This directly follows from the principle of any distinguishing attack on block ciphers.

We next prove the converse of the above statement that is if the SIFA attack is throttled, then $\forall x, y \in \mathcal{X}_{red} : p^*_x(x) = p^*_y(y)$. The only cause of SIFA being throttled is that the statistical distance between $p_=(x)$ and $\theta(x)$ is negligibly small or zero. The uniformity of $p_=(x)$ implies that $\forall x, y \in \mathcal{X}_{red} : p^*_x(x) = p^*_y(y)$ and vice versa (because with this condition $p_=(x) = 2^{-w'}$). The other side of the proof is fairly straightforward and we do not state it here. □

4 Preventing SIFA: The Main Idea

The key idea behind preventing SIFA is to make the probability distribution $p_=(x)$ uniformly random or at least sufficiently close to uniform so that number of samples (i.e. correct ciphertexts) required for attack goes beyond any practical limit. [3] It is well-understood that conventional fault attack countermeasures are not so useful in this context. A potential strategy could be to transform the domain of computation during the execution of the cipher in a

2. It is worth mentioning that for all the bits outside S_w, $p^j_{x_j \to x_j^{'}} = 1$, while $x_j = x_j^{'}$.

3. Another alternative could be to make $p_=(x)$ negligibly small $\forall x$. However, this property strongly depends on the nature of the fault injected, and hence not a very good choice.

way so that even if faults are injected at desired locations in the transformed computation state, their effect on the actual computation state causes an unbiased ineffective distribution. The most straightforward way of realizing this transformed domain of computation is to use some randomized encoding strategy. One should note that by the word "encoding" here we do not refer to any conventional error-correcting-code (ECC). Rather, the domain transformation encoding is aimed towards randomizing the state at each cipher execution in an way, so that the bias introduced in the state by means of a state-level fault gets mitigated. Fortunately, it has been observed that the idea of transforming the computation domain (from now onward, we call it as *domain transformation*) actually works. At the beginning of the next section, we shall theoretically analyze what kind of transformation may throttle SIFA, at least for certain reasonable fault models.

One important observation in this work is that state randomization indeed works till the fault is injected in the state register. In fact, provable security guarantees can be achieved even for multi-bit faults with certain reasonable restrictions. However, if the fault is induced into the intermediate computations (for example, in S-Box computation) the domain transformation is not sufficient. Putting it differently, *if the correct computation itself becomes biased, domain-transformation cannot mitigate it.* One option at this point is to incorporate hiding schemes such as shuffling or infection countermeasures. However, both of them is found to have limited success in the sense that they increases the ciphertexts requirements only quadratically. Another alternative is to correct the induced errors, so that the adversary becomes unaware if the fault happened or not. This seems to be a more reasonable approach for achieving security rather than hiding. Note that correction ensures the uniformity of $p_{=}(x)$ by mitigating the effect of faults. Moreover, the adversary will also have practical limitations in injecting targeted faults at intermediate computations in a useful manner. Hence, even an error-correction mechanism having limited correction power may reasonably solve the issue. This brings up one practical question in this context, that is up to what granularity level the error correction is required. As we shall show later in this paper, the error correction is required at the granularity of each bit processed within the cipher. Consequently, we require a number of redundant bits corresponding to each bit of the computation. Another important point here is that what happens if the error correction logic becomes the target of the attacker. We show that this case is equivalent to a fault injection at the state, and thus can be handled by the domain transformation strategy.

5 The Countermeasure Construction: Transform-and-Encode framework

In this section, we propose the Transform-and-Encode (TaE) framework to harden a given block cipher against SIFA attacks. First, we explain the general properties expected from the Transform strategy and present some instantiations for the same with varying security guarantees. The most interesting observation in this context is that *any secure masking scheme can be used as Transform operation with reasonably strong security assurance.* Next, we formally present the Encode operation and propose a simple albeit generic instantiation for it.

The high-level algorithmic representation of the overall framework is given in Algorithm 1. We begin by describing the Transform in the following subsection.

Algorithm 1 Transform-and-Encode (TaE)

Input: $S \in \{0,1\}^n, d, \gamma \in \mathbb{Z}$ where $d \leq n$
Output: $S^{enc} \in \{0,1\}^{\gamma n}, r \in \{0,1\}^d$
1: $(S^{tra}, r) \leftarrow$ Transform(S, d)
2: $S^{enc} \leftarrow$ Encode(S^{tra}, γ)
3: **Return** (S^{enc}, r)

Algorithm 2 Transform

Input: $S \in \{0,1\}^n, d \in \mathbb{Z}$ where $d \leq n$
Output: $S^{tra} \in \{0,1\}^n, r \in \{0,1\}^d$
1: $[S_1 \ \cdots \ S_d] \xleftarrow{\lceil n/d \rceil} S$
2: $r \in_R \{0,1\}^d$
3: $S^{tra} \leftarrow$ DomTr$([S_1 \ \cdots \ S_d], r)$
4: **Return** (S^{tra}, r)

Algorithm 3 DomTr

Input: $[S_1 \ \cdots \ S_d] \in \{0,1\}^n, r \in \{0,1\}^d$
Output: $S^{tra} \in \{0,1\}^n$
1: **for i = 1 to d**: $S_i^{tra} \leftarrow (r_i = 1) \ ? \ S_i \ : \ \overline{S_i}$
2: $S^{tra} \leftarrow [S_1^{tra} \ \cdots \ S_d^{tra}]$
3: **Return** S^{tra}

Algorithm 4 Encode

Input: $S \in \{0,1\}^*, \gamma \in \mathbb{Z}$
Output: $S^{enc} \in \{0,1\}^{\gamma|S|}$
1: $S^{enc} \leftarrow$ ECC(S, γ)
2: **Return** S^{enc}

5.1 The Transform Operation

The Transform operation aims to make the cipher state unbiased by random coin tossing to resist SIFA that tries to exploit non-uniformity at the state resulting from biased fault injection (SIFA-1). The state value should be accompanied with the output of the random coin tosses for decryption. Note that, in this case we only intend to protect against faults which are injected at the state of the cipher and does not affect the computation of any intermediate function like an S-Box. The abstract representation of Transform is presented in Algorithm. 2 and 3. Here the main idea is to parse an n-bit cipher state in d chunks each having length of $\lceil n/d \rceil$ bits. The d is the security parameter in this case, and different valuations of d leads to varying levels of security assurance. Next, a random bit r is generated corresponding to each $\lceil n/d \rceil$ bit chunk of the state. These random bits together construct the encoding for the randomized domain transform. The actual domain transformation algorithm is outlined in Algorithm. 3. Each $\lceil n/d \rceil$-bit state chunk is considered here one at a time, and depending on the value of the associated random bit, the chunk is either inverted or left unaltered.

There are certain generic security requirements expected from the Transform operation. One may observe that a biased fault injection at a state reduces its entropy. As a simple example, consider a stuck-at-0 fault injected at the

LSB of a 4-bit state chunk. It is easy to see that under the influence of this fault the state can only assume 8 values from total 16 possible values of it. In this simplest case we consider that the stuck-at-0 happens with probability 1. Even if this condition is relaxed (that is, stuck-at-0 happens with some probability < 0.5 or > 0.5), the state will still have some statistical bias and as a result, the entropy will be less than its maximum possible entropy. The security requirement for Transform can be stated in terms of entropy. Informally speaking:

In order to ensure security against biased state faults, it is essential that each chunk after domain transformation (S_i^{tra}) shows $\lceil n/d \rceil$ bit entropy even after the fault is injected.

The security criteria may seem a little non-intuitive. However, the main crux here is the usage of random bits (coin tosses) for constructing the Transform. This extra randomness helps us to satisfy the above-mentioned security criteria efficiently. Further details on this is provided in the next few subsections, where we detail some of the concrete instantiations of the Transform operation.

5.2 Instantiations of Transform

As mentioned in the previous subsection, different realizations of Transform is feasible with varying security guarantees. In this subsection, we explore some of such possible instantiations in detail. We begin with the simplest possible instantiation of $d = 1$, which is also referred to as Randomized-Reverse-Transform (RRT) in this paper. We show that *RRT ensures security against SIFA-1, when the width of the injected fault is one bit.* Next, we elaborate the most general case, where $d = n$. As it was found that *this generic transform is practically realizable with any state-of-the-art masking scheme.*

5.2.1 Transform with $d = 1$

The Transform operation with $d = 1$, also referred to as RRT, can be defined as follows:

Definition 1. *Given a state register S of length n, and a 1-bit random coin r_{RRT}, the RRT transform $\mathcal{C}_{RRT}(S)$ of S is defined as:*

$$\mathcal{C}_{RRT}(S) = \begin{cases} S, & \text{if } r_{RRT} = 0. \\ \overline{S}, & \text{otherwise.} \end{cases} \tag{6}$$

In other words, RRT implies that depending on a random coin toss either S or its reverse will be processed. Here we use only one random coin corresponding to the entire state. If the state of a cipher is encoded with RRT, then we can achieve SIFA protection against all possible single bit faults affecting the state. We provide a formal argument on the security through the following theorem:

Theorem 2. *If the fault influences one bit within S then RRT ensures SIFA protection.*

Proof. Without loss of generality, we assume that the i-th bit in S is influenced by the fault. We also assume that the $p^i_{x_i} \neq p^i_{\overline{x_i}}$, which ensures that the fault is biased (while ineffective). Now, we know

$$p_{=}(x) = p^*_x(x) / \sum_{x \in \mathcal{X}_{red}} p^*_x(x).$$

(ref. Theorem. 1) Also, since the fault is single bit $|\mathcal{X}_{red}| = 2$. Let, b_i^* be the i^{th} bit of $\mathcal{C}_{RRT}(S)$. Now, with RRT transform we have,

$$p^{'i}_{x_i} = \sum_{x \in \{x_i, \overline{x_i}\}} \mathbb{P}[b_i^* = x] \cdot p^i_x = \frac{p^i_{x_i} + p^i_{\overline{x_i}}}{2} \tag{7}$$

where, $p^{'i}_{x_i}$ is the ineffective transition probability of the i-th bit in the transformed domain ($p^{'i}_{x_i} = p^{'i}_{\overline{x_i}}$). Further,

$$p_{=}(x) = p^{'i}_{x_i} / (p^{'i}_{x_i} + p^{'i}_{\overline{x_i}}) = \frac{1}{2}, \tag{8}$$

which indicate that the distribution is uniform. The security against 1-bit fault injections is thus established. □

It is worth mentioning that, RRT cannot provide security if the fault affect w' non-contiguous bits within a specific w bit chunk S_w of S. We elaborate this by a counterexample as follows:

Example: Consider two consecutive bits b_i and b_{i+1} of S influenced with the biased ineffective faults. Without loss of generality, let us consider two values of S as x and x' where for x, $(x_i, x_{i+1}) = (0, 0)$ and for x', $(x'_i, x'_{i+1}) = (1, 0)$. Rest of the bit positions of x and x' assume same values. With the RRT transform (x_i, x_{i+1}) will be in two possible states $(0, 0)$ and $(1, 1)$ in x. Likewise, it will be in two possible states $(0, 1)$ and $(1, 0)$ in x'. Clearly, transition probabilities for x and x' will be different while (b_i, b_{i+1}) will be influenced by ineffective faults. More precisely, for the first case, the transition probability will be $\frac{p^i_0 p^{i+1}_0 + p^i_1 p^{i+1}_1}{2}$, whereas for the second case it will be $\frac{p^i_0 p^{i+1}_1 + p^i_1 p^{i+1}_0}{2}$. Clearly, the distribution $p_{=}(x)$ will not be uniform.

5.2.2 Transform with $d = n$

It is apparent that SIFA protection strongly depends on the distribution $p_{=}(x)$. Any protection mechanism should ensure the uniformity of this distribution. One interesting and relevant question in this context is whether there exist domain transformations which can provide provable security against multi-bit ineffective faults. Here we show that the answer to this question is positive at-least for most of the reasonable multi-bit faults. The key idea in this case is to define a Transform instance with $d = n$. Before, going into the details of one such instantiation, we elaborate why Transform with $d = n$ provides multi-bit security.

An obvious interpretation of the domain transformation strategy is as a mapping from a normal state space to a transformed state space. Let us denote this mapping as, $\mathcal{T}_D : \mathcal{X}_n \mapsto \mathcal{D}$, $\mathcal{D}$ being the transformed domain. Referring to the RRT transform, any value $x \in \mathcal{X}_n$ maps to one of the two values in $\mathcal{D}$ depending on a random coin r_{RRT}. One mandatory property for any such transform is that it must have a unique reverse transform and hence the mapping must be one-to-one (in other words $|\mathcal{X}_n| = |\mathcal{D}|$).

The SIFA security strongly depends on the number of options that a value $x \in \mathcal{X}_n$ may have for getting mapped. We formalize this in the following theorem.

Theorem 3. *A domain transformation strategy $\mathcal{T}_D$ will provide perfect SIFA security against multi-bit faults if and only if an element x in the domain $\mathcal{X}_n$ of $\mathcal{T}_D$ can assume any value from the range set $\mathcal{D}$, in a uniformly random manner.*

Proof. Let us consider a valuation x for the normal state S (no transformation applied). We also assume that $x \in \mathcal{X}_n$. The encoding $\mathcal{T}_D$ strategy maps x over the entire range set $\mathcal{D}$ of size $|\mathcal{D}|$, decided by random coins (note that, $|\mathcal{D}|= |\mathcal{X}_n|$). The transition probability of x can be written as:

$$p_x(x) = \frac{1}{|\mathcal{D}|} \sum_{x_d \in \mathcal{D}} p'_{x_d}(x_d) \tag{9}$$

where $x_d = \mathcal{T}_D(x)$ and $p'_{x_d}(x_d)$ is the ineffective transition probability in the encoded space. Now it can be observed that, $p_x(x) = \frac{1}{|\mathcal{D}|} \sum_{x_d \in \mathcal{D}} p'_{x_d}(x_d)$ are equal for $\forall x \in \mathcal{X}_n$. Using Theorem. 1, we can thus conclude that SIFA would not happen in this case. The other direction of the proof is straightforward and we do not mention it here. □

There are some subtle points regarding the Theorem. 3 stated above. One may observe that, we have assumed $|\mathcal{X}_n|= |\mathcal{D}|$ and the security proof strongly depends on this assumption. *In other words, we do not consider the random coin tosses defining the* Transform *as the part of the state in the transformed state-space.* One justification behind such assumption is that the transform gets defined at the very beginning of each execution of the cipher. *However, if a resourceful adversary can corrupt both the* Transform *as well as the transformed state, the above-mentioned security claim does not hold.* We note that, such an attack may be fairly simple for the case $d = 1$, as the Transform there is defined by a single bit. Fortunately, for the present case ($d = n$), this threat can be handled reasonably. *The easiest option is to use multiple random bits to transform each bit of the normal (before transformation) state in a way, so that the transformed state-space remains unbiased until all the random bits corresponding to a specific state-bit get corrupted with biased faults.* An obvious realization of this strategy is a secure masking scheme.

5.2.3 Masking as a Practical Transform

Masking is the most commonly used provably secure SCA countermeasure [16], [17]. The main idea of masking stems from secret-sharing [27]. In some sense, masking implements secret-sharing at the level of circuits. Several different flavours of masking have been proposed till date, the most common being the so-called Boolean masking. The main idea here is to share each bit b_i in a state S into m random bits denoted as $\langle b_i^0, b_i^1, \cdots b_i^{m-1} \rangle$. The only constraint on the sharing is that $b_i = b_i^0 + b_i^1 + \cdots b_i^{m-1}$. Furthermore, the shares, when considered on their own or in groups of up to $m - 1$ shares, are statistically independent of the unshared variable b_i. One should note that the shares never supposed to get combined during the entire computation. In order to realize this shared computation, the sub-functions of a cipher are shared into m different component functions with the restriction that the actual outcome of a sub-function f, denoted as $f(x, y)$, remains the same even after the sharing. More precisely, it is required that $f(x, y) = f_0(...) + f_1(...) + \cdots + f_{m-1}(...)$. The security parameter m here represents that up to what statistical order the scheme will be protected against SCA. Most of the secure masking schemes ensure that a security parameter of m provides a $(m - 1)$th order SCA security.

In the present context, any one of the shares of a m-share masking can be considered as our transformed domain ($\mathcal{D}$). The rest of the shares can be considered as the part of the randomness constructing the Transform. In other words, we slightly extend the definition of Transform in Algorithm. 2[4]. More precisely, instead of generating one random bit r_i in Line. 2, we generate $m - 1$ random bits $r_i^0, r_i^1, \cdots, r_i^{m-2}$. The actual transformation operation in the Line. 1 of Algorithm. 3 has to be modified as follows: *Instead of deciding the transform of S_i (unaltered or complement) based on a single bit r_i, we decide with $r_i^0 + r_i^1 + \cdots + r_i^{m-2}$.* In order to further elaborate why masking is a potential candidate for $d = n$-Transform we present the following example.

Example: Consider a bit b_i in S and its corresponding m shares $\langle b_i^0, b_i^1, \cdots b_i^{m-1} \rangle$. By definition of masking, each of the shares b_i^j are uniformly random. Considering a single specific share of all the bits of the complete intermediate state S, it can be observed that this share can assume any value from the entire state-space with equal probability, even if the unshared value remains fixed. As a concrete example, if we consider a 4-bit state S with value $(0, 1, 1, 0)$ and a 4-bit random mask M, then M can assume total 16 possible values with equal probability at different executions. Likewise, $S + M$ can also assume the same number of values in different executions. By this argument, masking becomes one potential candidate satisfying the requirements of Theorem 3 with very high probability.

The security implications of masking-based Transform directly follows from Theorem. 3, if we consider that the fault is limited within a single share of any bit. However, the security claim, in practice, is even stronger. *More specifically, unbiased state distribution can be ensured until all the shares corresponding to a given bit b_i comes under the influence of fault.* This follows from the classical Piling-up lemma [28], in cryptography. To elaborate this further, let us consider a bit b_i and its corresponding m shares $\langle b_i^0, b_i^1, \cdots b_i^{m-1} \rangle$. Any masking scheme requires

$$b_i + b_i^0 + b_i^1 + \cdots + b_i^{m-1} = 0. \tag{10}$$

Now, for a specific valuation of b_i, let $m-1$ shares are simultaneously corrupted with faults making their distributions biased. Without loss of generality we assume that only one share b_i^u remains unaffected by fault. The bias corresponding to each b_i^j is represented as ϵ_i^j. Quite evidently, $\epsilon_i^u = 0$. Now, the Piling-up lemma ensures that

$$\mathbb{P}[b_i + b_i^0 + b_i^1 + \cdots + b_i^{m-1} = 0] = \frac{1}{2} + 2^m \prod_{j=0}^{m} \epsilon_i^j. \tag{11}$$

With $\epsilon_i^u = 0$, the probability becomes $\frac{1}{2}$, which is what we require to ensure SIFA security. Since each bit in masking is transformed independently, this observation extends for the entire state. To summarize, we achieve the following:

Masking can provide multi-bit security against SIFA-1 faults until all m shares corresponding to a specific unmasked bit get influenced with faults.

Although, masking looks fairly good as SIFA countermeasure for state faults there are certain implementation issues to be taken care off.

4. Note that $|S_i|= 1$ for $d = n$

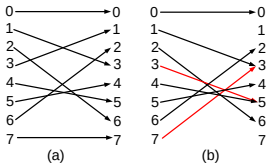

Fig. 2: Illustration of SIFA-2 fault on S-Box mapping. Considering the correct outcomes only, 1 and 7 never appears in output (statistical bias) under the influence of a SIFA-2 fault.

- As already pointed out, the security implication does not hold if all the shares corresponding to a specific bit gets corrupted. In order to ensure this, the masked bits should be placed at sufficiently distant locations. Considering the fact that faults are often localized at a small region within a register, such placement will reduce the chances of all the shares getting corrupted, simultaneously. For software implementations, another reasonable solution is to put different shares in different registers. If an adversary wants to corrupt all the shares simultaneously, she has to affect all the registers together which is fairly challenging even with most sophisticated fault injection setups available today.
- Another potential solution is to increase the number of shares to ensure that the probability of all shares corresponding to a single bit getting corrupted is reasonably low. However, this can only happen at the cost of increased overhead.

The direct application of masking as a SIFA countermeasure is interesting from several context. Most importantly, masking provides a combined protection against both SCA and SIFA. As a matter of fact, protection against SIFA comes without any overhead. It is also important to note that in many hardware and table-based software implementations, SIFA-1 is the most probable fault model. Hence, in practice, for most of these implementations masking will be sufficient to prevent SIFA. However, there also exists implementations, for which SIFA-2 faults are viable. Masking and hence the Transform operation cannot assure security against this fault model. In the next subsection, we elaborate how to remain secure even at the presence of SIFA-2 faults. More specifically, we present the second component Encode in the proposed framework which is meant for this purpose.

5.3 The Encode Operation

Before elaborating the Encode operation, we first provide the basic intuition behind the use of ECC to prevent SIFA-2 faults.

5.3.1 The Basic Intuition

As already pointed out, Transform is not sufficient while the fault is injected at some intermediate sub-operation of a cipher. In particular, nonlinear sub-operations like S-Boxes are typically susceptible to SIFA even if they are masked. The typical attack strategy, in this case, is to inject a fault (possibly unbiased) during some intermediate computation [5] of a sub-operation to create a bias in the actual (unmasked) output of the sub-operation. The core reason behind the creation of such output bias is that SIFA-2 faults typically change the original mapping realized by the target sub-operation. As a simple example, one may consider the simple 3 bit S-Box mapping shown in Fig. 2. The original mapping is bijective as shown in Fig. 2(a). However, under the influence of a typical SIFA-2 fault, the mapping may convert to the one shown in Fig. 2(b), even if masking is present. One may observe that, even for those cases when the S-Box output is correct, the output distribution becomes biased, simply due to that fact that all possible values are not assumed by the output.

In this context, one should note that linear sub-operations are typically less vulnerable against SIFA-2 as they require corruption of all the shares corresponding to a bit, simultaneously to result in a biased output. However, non-linear mappings, due to the existence of AND gates in them, do not require this restriction. In [7], it was shown that corrupting a single share during a masked S-Box computation may lead to a biased S-Box outcome. However, not any arbitrary corruption can lead to such biased output. It is observed that *corrupting at least one input share during the computation of the output shares of a specific output bit leads to the desired bias. Alternatively, one may also consider corrupting an intermediate gate input in the S-Box equations.* In contrast, *corrupting an input share for the computation of every output share (i.e. shares corresponding to all actual output bits) of a masked S-Box will lead to an unbiased output. In this case, the fault is equivalent to a state fault (SIFA-1).*

It is apparent from the above discussion that SIFA-2 is more powerful than SIFA-1. However, the faults in SIFA-2 cannot be arbitrary. Therefore, a countermeasure which may not work for any arbitrary fault, but works for fault instances exploitable in SIFA-2 model, would be reasonable in this context. Following this fact, we propose the use of ECC to throttle SIFA-2. It is well-known that correction capability of any ECC is limited. However, a SIFA-2 adversary is also supposed to be limited by the same fact, that she cannot corrupt any arbitrary number of chosen bits with any practical fault injection setup.

5.3.2 Error Correction Code for Encode

The abstract description of Encode is presented in Algorithm. 4, where the state S is enhanced with the ECC. It is well-known that redundancy in some form is essential to achieve error correction. The presence of redundancy is indicated by the scaling factor γ in Algorithm. 4. One important question here is that which one of Transform or Encode should be applied first. From security point of view, any one of them can be applied before the other. However, for the sake of simplicity, we consider the Transform to be applied before the Encode step.

5.3.3 Duplication Code as ECC

The most important question at the this point is how to realize the function ECC presented abstractly in Algorithm. 4. Several different instantiations of ECC is possible

5. For example, computation of the shares of a single output bit of an S-Box.

in principle. However, in this paper we present a specific instantiation with so-called *Duplication codes*. The main idea of duplication code is to repeat each bit multiple times to form a codeword. Mathematically it can be represented as:

$$\begin{aligned} \mathcal{E}_{dup}(S) &= \mathcal{E}_{dup}(\langle b_0, b_1, \cdots, b_{n-1}\rangle) \\ &= \langle (b_0)^{2t+1}, (b_1)^{2t+1}, \cdots, (b_{n-1})^{2t+1}\rangle \end{aligned} \tag{12}$$

Here t is a security parameter. A $(2t+1)$-bit duplication allows t bit error correction. Referring to Algorithm. 4, the scaling factor $\gamma = (2t+1)$. The fascinating feature of the proposed ECC is its simplicity, which enables easy understanding of the security implications. Moreover, the error correction works in a per-bit manner. The correction circuit can be implemented by simple *majority voting* among the duplicated bits corresponding to a specific state bit.

5.3.4 Secure implementation of Encode

Although duplication and majority voting based error correction is fairly simple, there are several subtle points to be taken care of while instantiating such a scheme. Below we note some of the suggestions required for a secure instantiation of the proposed ECC.

1) The first, and perhaps the most important question is where to put the error correction blocks. One may note that duplication code is functionally similar to calling an unprotected cipher $2t+1$ times and performing majority vote on the ciphertexts. However, there are several subtle differences. In the duplication encoding the redundancy is in-built at the bit level. *In order to take the advantage of this, we propose to instantiate one error correction block per bit, at the end of the S-Box computation at each round.* For example, if we consider PRESENT ($n = 64$) with $t = 1$ duplication, each of the 64-bits will be repeated 3 times. After the `sBoxLayer`, we expect total 64 correction units to be deployed at each round. In case the PRESENT implementation is masked (which is suggested), with a m-th order masking, the total number of units require would be $64 \times m$. One of the clear advantages of putting the correction blocks at each round is that it can prevent some trivial attacks. *Consider an implementation with $t = 1$, for example. An adversary may try to corrupt one of the three redundant branches at the very beginning of the computation with a less precise fault, and then may inject a desired fault in another branch. The first injection converts the correction mechanism to a detection based countermeasure. However, this cannot happen if the correction-block is present at each round. The correction logic can readily correct any such fault even if it is injected at the very beginning of computation.* Placement of correction-block is thus crucial for security.
2) The second crucial question is how the error correction logic should be implemented. One may note that, a resourceful adversary may try to corrupt the correction block to induce a desired fault. In order to throttle this chance we propose to implement the error correction in a redundant manner. In order to illustrate this let us consider the case where $t = 1$. Without loss of generality, let us denote the redundant bits corresponding to a specific state bit b_i as $\langle b_{0,i}, b_{1,i}, b_{2,i}\rangle$. The majority vote circuit here is given as:

$$b_{i,c} = b_{0,i}b_{1,i} + b_{1,i}b_{2,i} + b_{0,i}b_{2,i}. \tag{13}$$

One easy alternative for implementing the correction logic is to generate $b_{i,c}$ and to create two more copies of the same to maintain the encoding. Instead we propose to instantiate the correction circuit 3 times. In essence, we create three bits $\langle b_{c_0,i}, b_{c_1,i}, b_{c_2,i}\rangle$, such that each of them is generated from an independent copy of the correction logic. *Assuming that the adversary can inject a $(t = 1)$-bit precise fault to corrupt the output of the correction block, such an implementation ensures that the fault indeed gets corrected by another layer of correction circuit instantiated in later iterations of the cipher. Moreover, even if such a fault can be induced by corrupting multiple instances of the correction circuit, it can at most result in a biased fault on the state (SIFA-1 fault). Such faults are throttled by masking.*
3) One may wonder that whether masking is still needed if the bit level error correction is present. The answer is positive considering the limited correction capability of the ECC. In other words, the ECC is capable of correcting 1 fault injection per state bit. However, for practical reasons the adversary may not have any restriction on injecting multi-bit faults (say up to 5-6 bits) on the state (i.e. actual bits of the computation), which may result in alteration of 1-2 actual state bits. This may result in a biased fault scenario without the presence of masking. Moreover, masking is essential for SCA protection.

5.3.5 AntiSIFA: Our Final Proposal

The final AntiSIFA proposal suggests the application of any standard masking scheme for protection against SIFA-1. Higher order masking is preferred. For SIFA-2, we suggest bit level redundancy as described with lower values of t (typically $t = 1$ or $t = 2$). The reason behind suggestion is that high value for t would incur prohibitively large overhead. A n-bit cipher state expands to a state of $n \times m \times t$ with AntiSIFA. It is also important to precisely state which faults we prevent against. We summarize them as follows:

- For the SIFA-1, AntiSIFA can protect against any multi-bit faults provided the fault is not corrupting all the shares of a specific bit.
- For the SIFA-2, AntiSIFA can protect against t bit faults even while it is induced on all the redundant bits corresponding to a single bit.

Both of the fault situations for bypassing AntiSIFA are extremely aggressive and difficult to achieve even with modern fault injection setups. In the next section we present a concrete realization of AntoSIFA on PRESENT along with its security validation.

6 Validation on PRESENT

This section begins with a brief description of the PRESENT hardened with AntiSIFA. Next we describe the experiments we performed to validate the security claims.

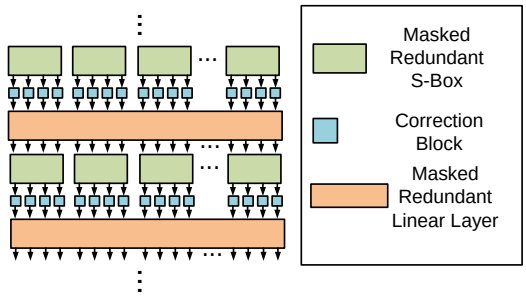

Fig. 3: Illustration of AntiSIFA. Each sub-operation is in masked and encoded domain.

6.1 PRESENT with AntiSIFA

PRESENT is a lightweight cipher having a block size $n = 64$. In order to realize AntiSIFA on PRESENT , we first need to choose a proper masking scheme. Without loss of generality, we choose the scheme proposed by Poschmann et. al. [29]. The proposal describes a 3-share TI implementation of PRESENT . In order to achieve 3-sharing of the cubic S-Box of PRESENT , the S-Box is represented as a composite mapping of two bijective quadratic functions, usually denoted as G and F. Three sharing is feasible for these quadratic functions. The TI implementation of the linear sub-operations (`pLayer` and `addRoundKey`) are straightforward. Comprehensive description of the masking scheme is out of the scope of this paper and we refer to [29] for further details. For the ECC component, we employ redundancy with $t = 1$. The correction blocks are instantiated accordingly, as pointed out in the previous section. A simple illustration of the protected PRESENT is depicted in Fig. 3.

6.2 Security Validation

6.2.1 SIFA-1 Model

We first validate the security against SIFA-1 faults. In order to explain the security claims properly, here we do not incorporate the ECC block. Fig. 4 presents the results for different fault scenarios in SIFA-1 model. The figures depict the variation of SEI scores with the number of correct ciphertexts. The red curves in the plots indicate SEI scores of the correct key and the blue curves represent highest SEI score among the wrong key guesses. For a successful attack we expect the correct key profile to have the higher and well-separated SEI value than that of the wrong keys. For clarity, the SIFA results on the unprotected PRESENT is also depicted (Fig. 4(a)). It is observed that, with a multi bit SIFA fault the attack happens with roughly 100 correct ciphertexts. However, attacks are completely prevented for the masked implementation. We specifically stress on the observation that, even corrupting up to two shares corresponding to a specific bit does not lead to a successful attack (Fig. 4(c)). In fact, to concretely validate this case, we performed a prolonged injection campaign with almost 100000 correct ciphertexts. The results clearly establish the efficacy of masking schemes for throttling SIFA-1 faults.

6.2.2 SIFA-2 Model

The experiments to validate our claims against SIFA-2 have been performed for the complete AntiSIFA. The outcome of this experiment is depicted in Fig. 5. The exploitable SIFA-2 faults were profiled prior to the experiment and used specifically to validate the countermeasure. To clearly establish the utility of the ECC, we also provide the result for SIFA-2 faults on masking implementation (Fig. 5(a)). The ECC block with $t = 1$ prevents SIFA-2 faults, till a single redundant bit corresponding to an actual cipher bit is corrupted with fault (Fig. 5(b)). In summary, AntiSIFA meets all the expectations developed in theory.

In order to further strengthen our claim, we perform experiments on a single S-Box with SIFA-2 faults. The results are depicted in Fig. 6. In the first experiment we consider a masked S-Box without ECC. The injection campaign is performed for all possible mask values. As it can be observed (Fig. 6(a)), the SIFA-2 faults create a bias in the actual output space. However, after the addition of ECC the biased distribution vanishes as only correct data is outputted by the hardened S-Box. As already pointed out, faulting the output of the correction block either results in a state fault or gets correct by the next correction block at the following round. The output distribution of the hardened S-Box is illustrated in Fig. 6(b), which is clearly an uniform distribution.

7 Conclusion

Statistical Ineffective Fault Analysis (SIFA) is a recently proposed class of fault attacks (FA), which typically analyzes the correct ciphertext obtained from a fault injection campaign. Most of the state-of-the-art FA countermeasures fall prey against SIFA. In this paper we propose a general framework to counter different classes of SIFA attacks in a provable manner. One of the key observation of this work is that even masking schemes can provide sufficient protection against SIFA for a little restricted fault model. For the general case we propose the use of error correction codes. Future work in this direction would consider further optimization of the ECC. Evaluating and protecting public key implementations against SIFA is another potential direction of research.

References

[1] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Proc. 17th Annual Cryptology Conference (CRYPTO)*. Santa Barbara, USA: Springer, Aug 1997, pp. 513–525.

[2] D. Boneh, R. DeMillo, and R. Lipton, "On the importance of checking cryptographic protocols for faults," in *Proc. 16th Int. Conf. Theory Applications Cryptographic Techniques (EUROCRYPT)*. Konstanz, Germany: Springer, May 1997, pp. 37–51.

[3] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," in *Proc. 5th IFIP WG 11.2 Int. Workshop, WISTP Inf. Secur. Theory Practice, Secur. Privacy Mobile Devices Wireless Commun.* Crete, Greece: Springer, June 2011, pp. 224–233.

[4] T. Fuhr, E. Jaulmes, V. Lomné, and A. Thillard, "Fault attacks on AES with faulty ciphertexts only," in *Proc. 10th IEEE Workshop Fault Diagnosis Tolerance Cryptogr. (FDTC)*. Santa Barbara, USA: IEEE, Aug 2013, pp. 108–118.

[5] N. F. Ghalaty, B. Yuce, M. Taha, and P. Schaumont, "Differential fault intensity analysis," in *Proc. 11th IEEE Workshop Fault Diagnosis Tolerance Cryptogr. (FDTC)*. Busan, Korea: IEEE, Sept 2014, pp. 49–58.

[6] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas, "SIFA: exploiting ineffective fault inductions on symmetric cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 547–572, 2018.

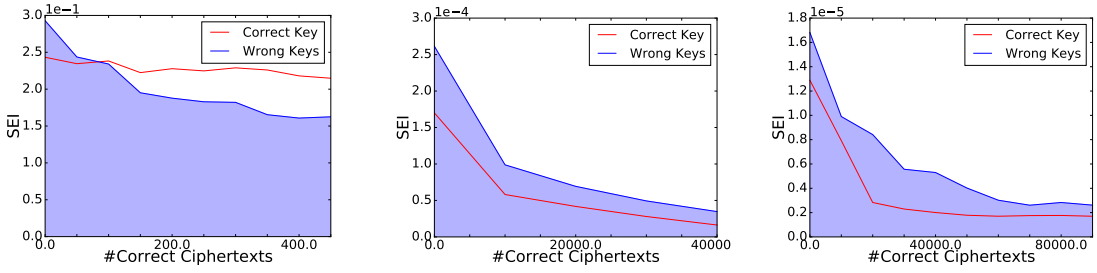

Fig. 4: SIFA-1 results for PRESENT . (a) SIFA on unprotected PRESENT ; (b) TI-PRESENT with multi-bit faults (all faults injected at one specific share of a nibble); and (c) TI-PRESENT with multi-bit faults (faults corrupt bits from two different shares of a nibble); Injections were carried out up to 450000 encryptions.

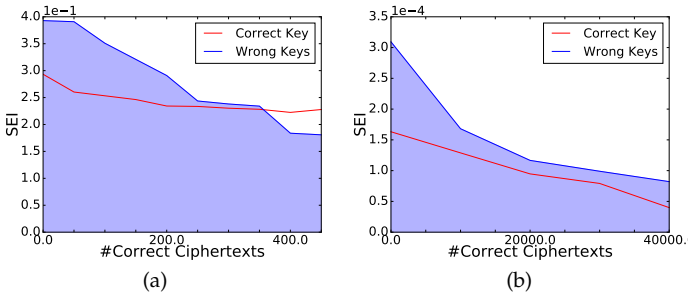

Fig. 5: Attack results on AntiSIFA with SIFA-2 faults: (a) Attack result without ECC; and (b) Attack result with ECC.

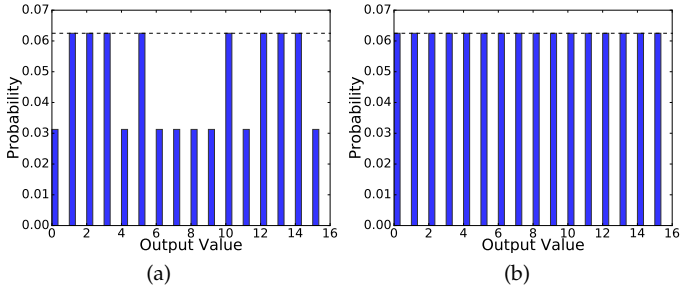

Fig. 6: S-Box output distribution with SIFA-2 faults: (a) Masking only; and (b) Masking + ECC.

[7] C. Dobraunig, M. Eichlseder, H. Gross, S. Mangard, F. Mendel, and R. Primas, "Statistical ineffective fault attacks on masked aes with fault countermeasures," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 315–342.

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

[9] X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri, "Security analysis of concurrent error detection against differential fault analysis," *Journal of Cryptographic Engineering*, vol. 5, no. 3, pp. 153–169, Sep 2015.

[10] H. Tupsamudre, S. Bisht, and D. Mukhopadhyay, "Destroying fault invariant with randomization," in *CHES'14*. Springer, 2014, pp. 93–111.

[11] K. Kulikowski, M. Karpovsky, and A. Taubin, "Robust codes for fault attack resistant cryptographic hardware," in *FDTC*, 2005, pp. 1–12.

[12] B. Gierlichs, J. Schmidt, and M. Tunstall, "Infective computation and dummy rounds: fault protection for block ciphers without check-before-output," in *LatinCrypt'12*. Springer, 2012, pp. 305–321.

[13] C. Clavier, "Secret external encodings do not prevent transient fault analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 181–194.

[14] V. Lomné, T. Roche, and A. Thillard, "On the need of randomness in fault attack countermeasures-application to aes," in *FDTC'12*. IEEE, 2012, pp. 85–94.

[15] S. Patranabis, A. Chakraborty, P. H. Nguyen, and D. Mukhopadhyay, "A biased fault attack on the time redundancy countermeasure for aes," in *Constructive Side-Channel Analysis and Secure Design*. Springer, 2015, pp. 189–203.

[16] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *ICICS*. Springer, 2006, pp. 529–545.

[17] E. Trichina, "Combinational logic design for aes subbyte transformation on masked data." *IACR Cryptology ePrint Archive*, vol. 2003, p. 236, 2003.

[18] C. Dobraunig, S. Mangard, F. Mendel, and R. Primas, "Fault attacks on nonce-based authenticated encryption: Application to Keyak and Ketje," in *International Conference on Selected Areas in Cryptography*. Springer, 2018, pp. 257–277.

[19] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," in *Proc. 9th Int. Workshop Cryptogr. Hardw. Embedded Syst. (CHES)*. Vienna, Austria: Springer, Sept 2007, pp. 450–466.

[20] C. Giraud, "DFA on AES," in *Proc. 4th Int. Conf. Advanced Encryption Standard*. Bonn, Germany: Springer, May 2004, pp. 27–41.

[21] N. Bagheri, R. Ebrahimpour, and N. Ghaedi, "New differential fault analysis on PRESENT," *EURASIP J. Adv. Sig. Proc.*, vol. 2013, no. 1, p. 145, 2013.

[22] N. F. Ghalaty, B. Yuce, and P. Schaumont, "Differential fault intensity analysis on PRESENT and LED block ciphers," in *Proc. 6th Int. Workshop Constructive Side-Channel Analysis Secure Design (COSADE)*. Berlin, Germany: Springer, Apr 2015, pp. 174–188.

[23] S. Saha, D. Mukhopadhyay, and P. Dasgupta, "ExpFault: an automated framework for exploitable fault characterization in block ciphers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 242–276, 2018.

[24] K. Jeong, Y. Lee, J. Sung, and S. Hong, "Improved differential fault analysis on PRESENT-80/128," *International Journal of Computer Mathematics*, vol. 90, no. 12, pp. 2553–2563, 2013.

[25] F. De Santis, O. M. Guillen, E. Sakic, and G. Sigl, "Ciphertext-only fault attacks on PRESENT," in *Proc. 3rd Int. Workshop Lightweight Cryptogr. Security Privacy (LightSec)*. Istanbul, Turkey: Springer, Sept 2014, pp. 85–108.

[26] S. Patranabis, J. Breier, D. Mukhopadhyay, and S. Bhasin, "One plus one is more than two: a practical combination of power and fault analysis attacks on PRESENT and PRESENT-like block ciphers," in *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2017, pp. 25–32.

[27] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[28] M. Matsui, "Linear cryptanalysis method for des cipher," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1993, pp. 386–397.

[29] A. Poschmann and et. al., "Side-channel resistant crypto for less than 2,300 ge," *Journal of Cryptology*, vol. 24, no. 2, pp. 322–345, 2011.