Natural Resistance of Threshold Implementations to Statistical (Ineffective) Fault Attacks

Viet Sang Nguyen, Vincent Grosso, and Pierre-Louis Cayrel

Laboratoire Hubert Curien, Université Jean Monnet, Saint-Étienne 42100, France
viet.sang.nguyen@univ-st-etienne.fr
vincent.grosso@univ-st-etienne.fr
pierre.louis.cayrel@univ-st-etienne.fr

Abstract. Fault attacks have been a significant threat against cryptographic implementations. Among many attack techniques, Statistical Fault Attacks (SFA) and Statistical Ineffective Fault Attacks (SIFA) have been shown to be very strong as they exploit only the cipher outputs. Protecting implementations against SFA and SIFA requires countermeasures such as spatial, temporal, or informational redundancy mechanisms. These countermeasures always come along with a high cost of latency or area. Moreover, this cost is even higher for SIFA protection as it additionally requires a special technique to propagate the fault to the output.

In this work, we propose a design for threshold implementations (TI) that resists SFA and SIFA attacks in a relatively relaxed adversary model. Our approach significantly reduces the cost required by existing fault countermeasures. For SFA, we show that TI *alone* is sufficient to provide the resistance without the need for expensive redundancy mechanisms. For SIFA, we show that TI is sufficiently efficient to provide the resistance when combined with a basic redundancy. Furthermore, we prove that TI conforming to our design is composable, enabling the construction of a full cipher that is resistant to both SFA and SIFA. Finally, we validate our approach with fault simulations on a proof-of-concept PRESENT implementation.

Keywords: Threshold Implementations · Fault Attacks · SFA · SIFA.

1 Introduction

Cryptographic algorithms are typically proven to be secure in a theoretical black-box model, where the adversary has access to a set of inputs and outputs. However, in the real world, these algorithms are widely deployed in embedded devices, such as smart cards or IoT devices, where physical attacks are a significant threat. These attacks can be categorized into two groups: (1) passive attacks, which exploit observable leakages, and (2) active attacks, which interfere with the execution of the algorithms.

Side-Channel Analysis (SCA) is a well-known passive attack technique that exploits physical execution characteristics such as timing [Koc96], power consumption [KJJ99], electromagnetic radations [GMO01]. The underlying principle of SCA is the correlation between the observed leakages and an internal variable containing *sensitive data* related to the secret key. Many protection schemes have been proposed to counteract SCA attacks. Among those, *masking* [GP99,CJRR99,ISW03,RBN+15,GMK16] is the most prominent countermeasure. The idea behind of masking is to harness the principle of secret sharing [Sha79] in order to split the sensitive data into several *shares* such that any combination of less than a chosen number of shares is independent of the sensitive data. Threshold Implementations (TI), proposed by Nikova *et al.* [NRR06], is a well-studied masking scheme for hardware computation, that follows the strategy of multi-party computation. In TI, the computation is split into parts, called *component functions* (corresponding to parties), that are *non-complete*, *i.e.*, independent of at least one input share. Nikova *et al.* proved that the first-order TI is secure for digital circuits even in the presence of glitches. Later, Sasdrich *et al.* [SBM18] applied the same concept for software implementations.

Fault Injection Analysis (FIA) is an active attack technique that requires to disrupt the execution of implementations through *e.g.* voltage supply [ZDCT13], clock glitching [DEG+18], electromagnetic field [DDRT12], or focused laser beams [SA03]. The underlying principle of FIA is the propagation of faults from an internal variable related to the secret key to the output. The idea of FIA was first introduced by Boneh *et al.* [BDL97] with an attack on RSA. Later, Biham and Shamir proposed the Differential Fault Analysis (DFA) [BS97], which has become the most studied fault-based attack and opened the door for many works, *e.g.*, [BS03,DLV03,Gir03,MSS06]. The core idea of these attacks is to recover the secret from differential equations constructed from a few pairs of correct and faulty outputs. A requirement for DFA-like attacks is that the adversary has the ability to encrypt the same plaintext twice, once with the presence of the faults and once without. Fuhr *et al.* [FJLT13] then relaxed this requirement with the Statistical Fault Analysis (SFA). This attack only requires access to the faulty ciphertexts and exploits the non-uniform distribution of an intermediate variable caused by the faults. SFA is also shown as a very effective fault attack by Dobraunig *et al.* [DEK+16] with practical applications on nonce-based authenticated encryption schemes. A common method to counteract fault attacks is to detect faults by employing temporal or spatial redundancy mechanisms. The basic principle is to repeat the computation multiple times and compare their outputs. However, redundancy-based countermeasures cannot prevent the clever idea of Statistical Ineffective Fault Analysis (SIFA) proposed by Dobraunig *et al.* [DEK+18], which exploits only the correct outputs. This means that SIFA still works in the implementations returning only correct outputs as the result of applying some error-detection schemes. In other words, error-detection schemes "help" to filter the outputs and select only the correct ones for the powerful SIFA attack.

Protecting implementations against SFA is trivial. Because this attack exploits faulty outputs, common fault detection schemes using temporal or spatial redundancy mechanisms are sufficiently efficient to prevent the threat of SFA. However, this approach always comes along with a cost of latency or area. For example, executing the implementation twice and verifying the two outputs as a temporal redundancy doubles the response time. In fact, recent (costly) schemes such as StaTI [DOT24], ParTI [SMG16], M&M [DAN+18], CAPA [RDB+18], whose concept is to combine redundancy and masking to thwart SCA and FIA, also provide the protection against SFA.

In contrast, protecting implementations against SIFA is not trivial. Exploiting solely correct outputs, this attack can overcome most of the countermeasures based on redundancy combined with masking. To tackle this problem, Daemen *et al.* [DDE+20] proposed the idea of ensuring the propagation of a fault to the output using reversible operations. The fault is then detected at the output by a redundancy method. However, as noted by the authors [DDE+20], the implementation cost can be a bottleneck of this countermeasure due to the redundancy for fault detection. Aiming to improve the efficiency of this approach, Dhooghe *et al.* [DOT24] have recently proposed a countermeasure based on TI and information redundancy, called StaTI. This countermeasure ensures the fault propagation on TI, then detects the fault at the output thanks to a linear decoding technique. Another approach relying on duplication-based countermeasures has been proposed by Baksi *et al.* [BKK+20]. However, their protection is limited to detecting stuck-at faults.

Contributions. This paper investigates the resistance of TI to SFA and SIFA. We start by analyzing the root cause that led to the success of applying SFA and SIFA attacks on TI by Dobraunig *et al.* [DEG+18]. In particular, we show that their fault model allows a fault attacker to break the non-completeness by injecting a fault into multiple component functions. Understanding this root cause, we propose a design for TI that resists SFA and SIFA attacks in a multiple-faults adversary model, regardless of the types of the faults (*e.g.*, bit flips, stuck-at-0 faults, stuck-at-1 faults, random faults). More specifically, we show that TI with extra caution implementation can achieve several levels of security:

- TI *alone* can provide resistance to SFA without the need for costly redundancy mechanisms as many schemes in the literature.
- TI, combined with a basic countermeasure such as spatial or temporal redundancy, is sufficiently efficient to provide resistance to SIFA. This approach is simpler than the existing costly countermeasures specialized for SIFA protection, such as [DDE+20].

Furthermore, we show that TIs adhering to our design are composable, enabling us to build a complete cipher that is resistant to both SFA and SIFA. We provide a proof-of-concept implementation of the PRESENT cipher and demonstrate its resistance through fault simulation.

As a result of using first-order TI, any implementations based on our design are inherently resistant to the first-order SCA, besides provides security against

SFA and SIFA. Nevertheless, we only focus on the resistance to SFA and SIFA in this paper.

Outline. Section 2 provides the background knowledge about TI, SFA and SIFA. Section 3 analyzes the root cause of SFA and SIFA attacks on TI. Section 4 presents the details of our design and the proofs of SFA and SIFA resistance. Section 5 discusses the composition of TIs that conform to our design. Section 6 provides the proof-of-concept PRESENT implementation and the fault simulation.

2 Preliminaries

In this section, we first introduce notation used in the the paper, then present the notions of TI, SFA and SIFA.

2.1 Notation

In this paper, we consider the TI of an n-bit bijection f defined from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$, where $\mathbb{F}_2^n$ denotes a finite field of characteristic 2. In this context, a *bit* is an element of $\mathbb{F}_2$. Lower-case characters (*e.g.*, x, a) are used to refer to elements of the finite field $\mathbb{F}_2^n$, while upper-case characters (*e.g.*, X, A) are used to refer to stochastic variables. A bold character (*e.g.*, $\mathbf{X}$, $\mathbf{a}$) is used to refer to a vector of variables or values. The probability that the variable X takes the value x is denoted by $\Pr[X = x]$. The bitwise addition and multiplication of the two values x and y, designated as the XOR and the AND operations, are represented by the symbols $x \oplus y$ and xy. The Boolean logical *and*, *or* and *negation* are represented by the symbols $\vee$, $\wedge$ and $\neg$, respectively.

To implement the function $A = f(X)$ from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$ in TI, it is first necessary to use Boolean masking to split the variable $X \in \mathbb{F}_2^n$ into s shares. These shares are denoted by $X_1, \ldots, X_s \in \mathbb{F}_2^n$, such that the XOR sum of these shares is equal to the variable itself:

$$X = X_1 \oplus \ldots \oplus X_s.$$

We call X an *unshared variable*. The process of decoding the shares $X_1, \ldots, X_s$ to obtain the unshared variable X is referred to as *unmasking*. In this paper, we use a subscript j is used to denote the j-th share X_j, with $j \in [1, s]$. Furthermore, a superscript i is employed to denote the i-th bit X^i, for $i \in [1, n]$, of the n-bit variable X. Thus, X_j^i refers to the j-th share of the i-th bit of X. The vector containing all s shares of X is denoted by $\mathbf{X} = (X_j)_{j \in [1,s]}$. We also denote by $\mathbf{X}_{\bar{j}}$ the vector of all the shares, but the j-th share, of all n bits, $\mathbf{X}_{\bar{j}} = \mathbf{X} \setminus \{X_j\}$.

2.2 Threshold implementations (TI)

The TI of the bijection $A = f(X)$, is represented by the *shared function*, denoted $\mathbf{A} = \mathbf{f}(\mathbf{X})$. This shared function is a vector of *component functions* $\mathbf{f} =$

$(f_1, \ldots, f_s)$. The shared functions take as input the *input shares* $\mathbf{X} = (X_j)_{j\in[1,s]}$ and produce as output the *output shares* $\mathbf{A} = (A_j)_{j\in[1,s]}$. The shared function $\mathbf{f}$ is constructed in such a way that (for the first-order TI) every component function f_j is independent of at least one share (usually the j-th share). Figure 1 depicts the architecture of a first-order TI with 3 shares. As introduced by Nikova *et al.* [NRR06], the construction of a TI must satisfy the following properties: correctness, non-completeness, and uniformity.

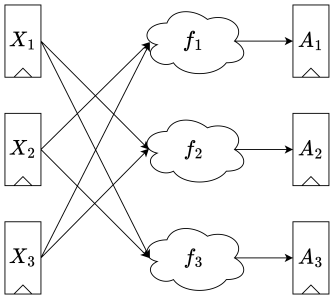


Fig. 1: Example of a first-order threshold implementation with $s = 3$ shares. The component function f_j is independent of the j-th shares of all input bits $\mathbf{X}_j$.

The first property, correctness, ensures that the unmasking of the output shares will yield the intended unshared output.

Property 1 (Correctness [NRR06]). For any $x \in \mathbb{F}_2^n$ and any vector of shares $\mathbf{x} = (x_j)_{j\in[1,s]}$ such that $x = \sum_i x_i$, the output shares $\mathbf{a} = (a_j)_{j\in[1,s]}$ of the threshold implementation $\mathbf{a} = \mathbf{f}(\mathbf{x})$ must satisfy $a = \sum_i a_i$, where $a = f(x)$.

The second property, (first-order) non-completeness, ensures that each component function is independent of at least one of the input shares.

Property 2 (First-order non-completeness [NRR06]). A threshold implementation $\mathbf{f}$ achieves non-completeness if each component function f_j is independent of at least one input share, resulting in each output share being independent of at least one input share.

The third property, uniformity, includes the definition of uniform sharing (Property 3) and the definition of uniform sharing of a function (Property 4). These definitions require the notation for the set of all valid sharings of x, denoted by $\mathrm{Sh}(x)$. The uniform sharing property ensures a sharing vector of x is uniformly distributed over $\mathrm{Sh}(x)$.

Property 3 (Uniform sharing [NRR06]). A sharing $\mathbf{X}$ is uniform if and only if

$$\Pr[\mathbf{X} = \mathbf{x} | \mathbf{x} \in \mathrm{Sh}(x)] = \frac{1}{|\mathrm{Sh}(x)|}.$$

Let us suppose that the input sharing of a shared function is uniformly distributed (that means Property 3 is satisfied). It then follows that the output sharing of this function is uniform. Thanks to this property, we can compose two shared functions without refreshing the shares.

Property 4 (Uniformity or Balancedness [NRR06]). The shared function $\mathbf{f}$ is uniform if and only if $\forall x, a \in \mathbb{F}_2^n$ with $a = f(x), \forall \mathbf{a} \in \mathrm{Sh}(a)$:

$$|\{\mathbf{x} \in \mathrm{Sh}(x) | \mathbf{f}(\mathbf{x}) = \mathbf{a}\}| = 1.$$

In other words, if f is a bijection then $\mathbf{f}$ is also a bijection.

2.3 Statistical Fault Attacks (SFA)

The concept of Statistical Fault Attacks (SFA) was initially proposed by Fuhr *et al.* [FJLT13] with a demonstration of the Advanced Encryption Standard (AES). The key concept of SFA is to induce a non-uniform distribution of a byte before the MixColumns operation in the 9th round. The distribution of the selected byte is modified by faults as depicted in Figure 9 (in Section C). Consequently, the adversary only requires to have access to the faulty ciphertexts in order to recover the key. In the case that a set of faulty ciphertexts is available, the adversary needs to predict 4 bytes of the last round key and then perform a backward computation to predict the value of the faulted position. It should be noted that guessing the 9th round key is not necessary, since it does not affect the non-uniformity distribution of the selected byte (the 9th round key can be regarded as an addition with an unknown constant, therefore it does not impact the distribution). If a sufficient number of ciphertexts are provided, the distance between the distribution at the faulted byte corresponding to the correct key guess and the uniform distribution will be the highest. Consequently, it is possible to distinguish between a correct and an incorrect key guess.

The approach of SFA is noteworthy for its divergence from the methodology of Differential Fault Attacks (DFA), which necessitates the encryption of identical plaintext on two separate occasions. However, the main disadvantage of SFA is its reliance on faulty ciphertexts. This means that cryptographic implementations with countermeasures that prevent the release of faulty ciphertexts for DFA can also be resistant to SFA. Therefore, the potential danger posed by SFA to these protected implementations is not considerable.

2.4 Statistical Ineffective Fault Attacks (SIFA)

Since its publication by Dobraunig *et al.* [DEK+18] the concept of Statistical Ineffective Fault Attacks (SIFA) has emerged as a particularly potent form of fault attack. In contrast to SFA, SIFA exploits only correct ciphertexts resulting from the failure of the faults to be effective, that is when the faults are successfully injected but do not render the ciphertexts faulty. This concept was derived from the interesting observation made by the authors that the ineffective faults

result in a non-uniform distribution of an intermediate variable. Table 1 provides an illustrative example of the distribution table for a stuck-at-0 fault on 2-bit values. The diagonal of red values represents the non-uniform distribution of ineffective faults ($x = x'$).

		x'			
		00	01	10	11
x	00	1	0	0	0
	01	1	0	0	0
	10	1	0	0	0
	11	1	0	0	0

Table 1: Fault distribution table for 2-bit stuck-at-0 fault model [DEK+18]. The probability that any value x becomes $x' = 00$ by a stuck-at-0 fault is 1 (first column). The probability that any value x becomes $x' \neq 00$ by a stuck-at-0 fault is 0 (other columns).

As it relies solely on the correct ciphertexts, SIFA remains applicable in scenarios where redundancy-based countermeasures are employed. In the following analysis, we consider an AES with a classical temporal redundancy, *e.g.*, the computation is run twice and the two outputs are compared to prevent the release of faulty ciphertexts. Let us consider the scenario in which an attacker injects stuck-at-0 faults, into a byte before MixColumns in the 9th round (similar to Figure 9) of one of the two redundant computations. The countermeasure will filter and release solely correct ciphertexts. Given this filtered set of correct ciphertexts, the attacker performs a key recovery as in SFA, *i.e.*, guessing 4 bytes of the last round key and computing backward to the faulted position. The correct key guess will correspond to the furthest distance from the distribution of the faulted byte to the uniform distribution.

Dobraunig *et al.* [DEG+18] subsequently applied SIFA to protected implementations using masking and error detection. In particular, the authors demonstrated that when a single share during the masked computation of an S-box is faulted, a non-uniform distribution of an unshared intermediate variable can be exploited by SIFA. Furthermore, threshold implementations, which will be thoroughly investigated in the following sections, are also vulnerable to SIFA, as shown in [DEG+18].

3 SFA and SIFA on threshold implementations

This section presents a detailed analysis of the root cause of SFA and SIFA attacks by Dobraunig *et al.* [DEG+18] on the threshold implementation (TI) of the 5-bit Keccak S-box with a single fault. We begin with a brief introduction about the TI of the Keccak S-box. We then analyze the effect of the single fault and calculate the probability of each unshared input and output value

in the distribution resulting from the fault. This detailed calculation helps to understand the primary reason why SFA and SIFA are applicable and inspires the design of the implementation resisting SFA and SIFA in the following section.

3.1 Single fault on TI of Keccak S-box

Keccak is a versatile cryptographic function with variable-length input and arbitrary-length output based on the sponge construction [Kec15]. The core of the Keccak S-box $\mathcal{S}$ is a quadratic mapping:

$$A^i \leftarrow X^i \oplus (X^{i+1} \oplus 1)X^{i+2}.$$

For the 5-bit S-box that we consider hereafter, $i \in [1, 5]$, where X^i denotes the i-th input bit and A^i denotes the i-th output bit, X^1 and A^1 are the least significant bits. The indices $i + 1$ and $i + 2$ are implicitly cyclic in [1,5]. The lookup table of $\mathcal{S}$ is given in Table 2.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
$\mathcal{S}(x)$	0	9	12	b	5	c	16	f	a	3	18	1	d	4	1e	7	14	15	6	17	11	10	2	13	1a	1b	8	19	1d	1c	e	1f

Table 2: Lookup table of 5-bit Keccak S-box.

In [BDN+13], Bilgin *et al.* propose a first-order uniform 4-share threshold implementation of the Keccak S-box. Let X_1^i, X_2^i, X_3^i and X_4^i be 4 shares of X^i. The 4 shares $A_1^i, A_2^i, A_3^i, A_4^i$ of each output bit A^i are computed as follows.

For $i = 1, 2, 3, 5$:

$$\begin{aligned}
A_1^i &= X_2^i \oplus X_2^{i+2} \oplus ((X_2^{i+1} \oplus X_3^{i+1} \oplus X_4^{i+1})(X_2^{i+2} \oplus X_3^{i+2} \oplus X_4^{i+2})) \\
A_2^i &= X_3^i \oplus X_3^{i+2} \oplus (X_1^{i+1}(X_3^{i+2} \oplus X_4^{i+2}) \oplus X_1^{i+2}(X_3^{i+1} \oplus X_4^{i+1}) \oplus X_1^{i+1}X_1^{i+2}) \\
A_3^i &= X_4^i \oplus X_4^{i+2} \oplus (X_1^{i+1}X_2^{i+2} \oplus X_1^{i+2}X_2^{i+1}) \\
A_4^i &= X_1^i \oplus X_1^{i+2}
\end{aligned} \tag{1}$$

For $i = 4$:

$$\begin{aligned}
A_1^4 &= X_2^4 \oplus X_2^1 \oplus X_3^1 \oplus X_4^1 \oplus ((X_2^5 \oplus X_3^5 \oplus X_4^5)(X_2^1 \oplus X_3^1 \oplus X_4^1)) \\
A_2^4 &= X_3^4 \oplus X_1^1 \oplus (X_1^5(X_3^1 \oplus X_4^1) \oplus X_1^1(X_3^5 \oplus X_4^5) \oplus X_1^5X_1^1) \\
A_3^4 &= X_4^4 \oplus (X_1^5X_2^1 \oplus X_1^1X_2^5) \\
A_4^4 &= X_1^4
\end{aligned} \tag{2}$$

Dobraunig *et al.* [DEG+18] showed that it is sufficient to inject a fault into the value of a single input share during the execution of the above implementation is sufficient to mount SIFA and SFA. More specifically, a stuck-at-0 fault is injected

into the share X_1^1 before the computation of the four shares of the output bit A^5 are computed, while the shares of the other output bits are computed correctly. We highlight in red the influence of the faulted variable X_1^1 as described in Equation 3.

$$\begin{aligned} A_1^5 &= X_2^5 \oplus X_2^2 \oplus ((X_2^2 \oplus X_3^1 \oplus X_4^1)(X_2^2 \oplus X_3^2 \oplus X_4^2)) \\ A_2^5 &= X_3^5 \oplus X_3^2 \oplus (X_1^1(X_3^2 \oplus X_4^2) \oplus X_1^2(X_3^1 \oplus X_4^1) \oplus X_1^1 X_1^2) \\ A_3^5 &= X_4^5 \oplus X_4^2 \oplus (X_1^1 X_2^2 \oplus X_1^2 X_2^1) \\ A_4^5 &= X_1^5 \oplus X_1^2 \end{aligned} \tag{3}$$

As a consequence, the following AND operations are affected: $X_1^1(X_3^2 \oplus X_4^2)$ and $X_1^1 X_1^2$ in the computation of A_2^5, and $X_1^1 X_2^2$ in the computation of A_3^5. Since the first and the second operations are both in the computation of A_2^5, we obtain $X_1^1(X_1^2 \oplus X_3^2 \oplus X_4^2)$ by factorizing them. The fault is called *effective* when it induces a bit flip on the unshared output bit $A^5 = A_1^5 \oplus A_2^5 \oplus A_3^5 \oplus A_4^5$. This occurs if and only if either A_2^5 or A_3^5 is flipped by the fault. Note that if both shares A_2^5 and A_3^5 are flipped, there is no effect on the unshared output bit A^5 because these two flips will cancel each other out in the unmasking of A^5. Hence, we can combine $X_1^1(X_1^2 \oplus X_3^2 \oplus X_4^2)$ (in A_2^5) and $X_1^1 X_2^2$ (in A_3^5) into a single term expressing the influence of the fault:

$$X_1^1(X_1^2 \oplus X_2^2 \oplus X_3^2 \oplus X_4^2) = X_1^1 X^2. \tag{4}$$

Equation 4 implies that all the shares of the input bit X^2 are affected by the fault. This breaks the non-completeness property of threshold implementations and thus leads to the non-uniform distribution of the unshared S-box input and output (Figure 3 for SIFA and Figure 2 for SFA). We now analyze Equation 4 in more detail to see how the fault affects the output bit A^5. Let us first formally define the effectiveness of the fault.

Definition 1 (Effectiveness of fault(s)). *Fault(s) induced into the computation of a shared function* $\mathbf{A} = \mathbf{f}(\mathbf{X})$ *are said to be effective if the unshared output* A*, where* A *is unmasked from* $\mathbf{A}$*, is changed because of the fault(s). Otherwise, the fault(s) are said to be ineffective.*

We consider the following possibilities:

(a) $X_1^1 = 0$: the stuck-at-0 fault does not change the value of X_1^1. No matter what the value of X^2 is, the result of the AND operation $X_1^1 X^2$ is always 0. Thus, the fault is ineffective.
(b) $X_1^1 = 1$ and $X^2 = 0$: despite the value of X_1^1 change from 1 to 0, the stuck-at-0 fault does not change the result of the AND operation $X_1^1 X^2$ since $X^2 = 0$. Thus, the fault is ineffective.
(c) $X_1^1 = 1$ and $X^2 = 1$: the stuck-at-0 fault changes the value of X_1^1 from 1 to 0, and thus changes the result of the AND operation $X_1^1 X^2$ from 1 to 0. As a result, the unshared output bit A^5 is flipped. Thus, the fault is effective.

Notice that the fault is effective only in the case (c). Let ξ denote the Boolean expression representing the effectiveness of the fault, such that $\xi = 1$ when the fault is effective and $\xi = 0$ otherwise. Equation 5 is the Boolean expression for the fault that we are considering. To show that the induced stuck-at-0 fault results in a non-uniform distribution for the unshared input and output, we will calculate the probability of occurrence for each input and output value. This calculation also helps to ease the proofs for our design in Section 4.

$$\xi = X_1^1 \wedge X^2 \tag{5}$$

Application of SFA. Figure 2 shows the distribution of the unshared input and output values induced by the fault. In SFA, both correct outputs and faulty outputs are taken into account, *i.e.*, there is no filter to eliminate faulty outputs. Since the fault occurs in an intermediate computation, it does not affect the uniformity of the input. This explains the uniform distribution of the unshared input I in Figure 2.

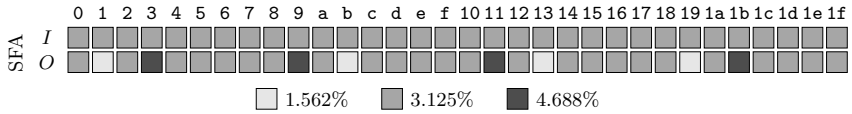


Fig. 2: Distribution of input I and output O for SFA on 4-share Keccak S-box.

We now focus on the distribution of the unshared output O. Recall that the fault is effective when $\xi = X^2 \wedge X_1^1 = 1$, or equivalently, $X^2 = 1$ and $X_1^1 = 1$. We split the input values into two sets, $\mathcal{I}$ and $\mathcal{I}'$, including the values with the second bit $X^2 = 0$ and $X^2 = 1$, respectively.

$$\mathcal{I} = \{\texttt{0,1,4,5,8,9,c,d,10,11,14,15,18,19,1c,1d}\}$$
$$\mathcal{I}' = \{\texttt{2,3,6,7,a,b,e,f,12,13,16,17,1a,1b,1e,1f}\}$$

The two output sets corresponding to these input sets are:

$$\mathcal{O} = \{\texttt{0,9,5,c,a,3,d,4,14,15,11,10,1a,1b,1d,1c}\},$$
$$\mathcal{O}' = \{\texttt{12,b,16,f,18,1,1e,7,6,17,2,13,8,19,e,1f}\}.$$

We continue to split $\mathcal{O}$ and $\mathcal{O}'$ into $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2$ and $\mathcal{O}' = \mathcal{O}'_1 \cup \mathcal{O}'_2$, where:

$$\mathcal{O}_1 = \{\texttt{0,4,5,a,c,d,10,14,15,1a,1c,1d}\}, \quad \mathcal{O}_2 = \{\texttt{3,9,11,1b}\},$$
$$\mathcal{O}'_1 = \{\texttt{2,12,6,16,7,17,8,18,f,1f,e,1e}\}, \quad \mathcal{O}'_2 = \{\texttt{1,b,13,19}\}.$$

Recall that in case the fault is effective, the most significant output bit A^5 will be flipped. Hence, the fault value can be denoted by $\epsilon = \texttt{10}$ and the change of the output from A to A' can be denoted by $A' = A \oplus \epsilon$.

– For $a \in \mathcal{O}'_1$ ($X^1 = 1$), if the fault is effective ($X^1_1 = 1$), we have $a' = a \oplus \epsilon \in \mathcal{O}'_1$. The value a changes to a' and the value a' changes back to a. Overall, the probability does not change

$$\Pr[A = a, a \in \mathcal{O}'_1] = 2^{-5} = 0.03125.$$

– For $a \in \mathcal{O}'_2$ ($X^1 = 1$), if the fault is effective ($X^1_1 = 1$), then $a' = a \oplus \epsilon \in \mathcal{O}_2$ (a becomes a value in $\mathcal{O}_2$). This value a does not change when the fault is ineffective, *i.e.*, when $\xi = X^2 \wedge X^1_1 = 0$, or equivalently, $X^1_1 = 0$ since $X^1 = 1$ for $a \in \mathcal{O}'_2$. Thus, by the independence between A and X^1_1:

$$\begin{aligned}\Pr[A = a, a \in \mathcal{O}'_2] &= \Pr[A = a \wedge \xi = 0] \\ &= \Pr[A = a \wedge X^1_1 = 0] \\ &= 2^{-5} \times 2^{-1} = 0.015625.\end{aligned}$$

– For $a \in \mathcal{O}_2$ ($X^1 = 0$), the fault is ineffective ($\xi = X^1 \wedge X^1_1 = 0$). Note that some values in $\mathcal{O}'_2$ become the values in $\mathcal{O}_2$ because of the fault. Thus:

$$\begin{aligned}\Pr[A = a, a \in \mathcal{O}_2] &= 2^{-5} + (2^{-5} - \Pr[A = a, a \in \mathcal{O}'_2]) \\ &= 0.03125 + (0.03125 - 0.015625) \\ &= 0.046875.\end{aligned}$$

– For $a \in \mathcal{O}_1$ ($X^1 = 0$), the fault is ineffective ($\xi = X^1 \wedge X^1_1 = 0$). The values in this set are not influenced by the fault. Thus:

$$\Pr[A = a, a \in \mathcal{O}_1] = 2^{-5} = 0.03125.$$

We have obtained the distribution as illustrated in Figure 2.

Application of SIFA. Figure 3 shows the distribution of the unshared input and output values induced by the fault when it is *ineffective* (the case (c), $\xi = 1$), *i.e.*, incorrect outputs are detected and eliminated by a redundancy. Intuitively, we consider a truth table of 2^{20} entries for all combinations of 20 shares for 5 input bits. One fourth entries (or 2^{18}) where $X^2 = 1$ and $X^1_1 = 1$ are eliminated from this table. Note that $X^2 = 1$ means that the input of the S-box is in the set {`2,3,6,7,a,b,e,f,12,13,16,17,1a,1b,1e,1f`} (illustrated by light gray cells for I in Figure 3). This explains why the probability of each dark gray cell is greater than the probability of each light gray cell for the input I.

For the output O in Figure 3, the probability of $A = a$ is the same as probability of $X = x$ if $a = \mathcal{S}(x)$ because the S-box is a permutation. For example, $\Pr[A = \texttt{12}] = \Pr[X = \texttt{2}]$ since $\mathcal{S}(\texttt{2}) = \texttt{12}$. Thus, knowing the distribution of the input X, we can directly derive the distribution of the output A. We now focus only on calculating $\Pr[X = x]$ for each $x \in [\texttt{0,1f}]$. This probability in case the fault is ineffective ($\xi = 0$) can be calculated as:

$$\Pr[X = x | \xi = 0] = \frac{\Pr[X = x \wedge \xi = 0]}{\Pr[\xi = 0]}. \tag{6}$$

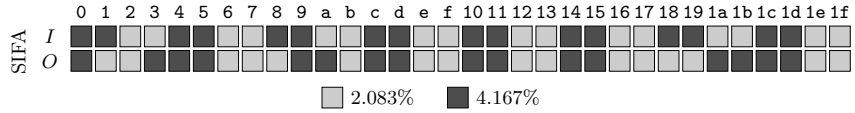


Fig. 3: Distribution of input I and output O for SIFA on 4-share Keccak S-box (taken from [DEG+18]).

First, the probability that the fault is ineffective: $\Pr[\xi = 0] = 1 - \Pr[\xi = 1] = 1 - \Pr[X^2 \wedge X_1^1 = 1] = 1 - \Pr[X^2 = 1] \times \Pr[X_1^1 = 1] = 1 - 0.5 \times 0.5 = 0.75$ (by the independence between X^2 and X_1^1). Second, to calculate $\Pr[X = x \wedge \xi = 0]$, we consider the following cases:

- $X^2 = 0$: This is equivalent to $x \in \{$`0,1,4,5,...,1c,1d`$\}$ (illustrated by dark gray cells for I in Figure 3). We have $\xi = X^2 \wedge X_1^1 = 0$ (always true), thus $\Pr[X = x \wedge \xi = 0] = \Pr[X = x] = 2^{-5}$.
- $X^2 = 1$: This is equivalent to $x \in \{$`2,3,6,7,...,1e,1f`$\}$ (illustrated by light gray cells for I in Figure 3). We have $\xi = X^2 \wedge X_1^1 = X_1^1$, thus $\Pr[X = x \wedge \xi = 0] = \Pr[X = x \wedge X_1^1 = 0] = 2^{-5} \times 2^{-1} = 2^{-6}$ (by the independence between X and X_1^1).

Finally, Equation 6 can be evaluated and we obtain the distribution as illustrated in Figure 3:

$$\Pr[X = x | \xi = 0] = \begin{cases} 0.04167 & \text{for } x \in \{\texttt{0,1,4,5,...,1c,1d}\}, \\ 0.02083 & \text{for } x \in \{\texttt{2,3,6,7,...,1e,1f}\}. \end{cases}$$

3.2 Root cause

In the analysis on the Keccak S-box, a fault injected in a single share (X_1^1) causes an influence on all the shares ($X_1^2, X_2^2, X_3^2, X_4^2$) of the second input bit X^2. It can be said that the fault causes the non-completeness property of TI to be broken (Property 2). This is the main reason for the non-uniform distributions of the input and output (Figure 2 and Figure 3) that can be exploited by SFA and SIFA.

Moreover, X_1^1 is not the only choice for the fault injection. For instance, injecting a stuck-at-0 fault into X_1^2 before the computation of the four shares of the output bit A^5 (Equation 3) also creates an influence on all the shares of the input bit X^1, and thus leads to non-uniform distributions of the S-box that are vulnerable to SFA and SIFA.

The root cause behind the attack is the arrangement of the computation order that allows a single fault to affect the computation of all the shares of an output bit (A^5 in the example of Keccak). The non-completeness property is thus no longer guaranteed because of the consequence of the fault. This root cause not only holds for the case study of Keccak, but also holds for TI in general.

Let us take the 3-share TI of the 4-bit PRESENT S-box as another example (its details of the TI computations are included in Section B). We arrange the computation order as the root cause, *i.e.*, a single fault can spread its influence to the computation of all the shares of an output bit. More specifically, we assume that a stuck-at-0 fault is induced into the share Y_1^4 before the computation of the 3 shares of the output bit A^4. As expected, the distributions of the unshared input and output are no longer uniform as illustrated in Figure 4, which means that SFA and SIFA are applicable.

$$\begin{aligned} A_1^4 &= Y_2^3 \oplus Y_2^4 Y_3^2 \oplus Y_2^2 Y_3^4 \oplus Y_3^2 Y_3^4 \\ A_2^4 &= Y_3^3 \oplus Y_3^4 Y_1^2 \oplus Y_3^2 Y_1^4 \oplus Y_1^2 Y_1^4 \\ A_3^4 &= Y_1^3 \oplus Y_1^4 Y_2^2 \oplus Y_1^2 Y_2^4 \oplus Y_2^2 Y_2^4 \end{aligned} \tag{7}$$

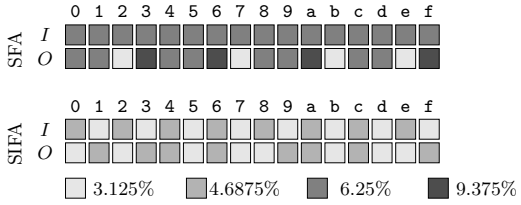


Fig. 4: Distribution of input I and output O for SFA and SIFA on 3-share PRESENT S-box.

4 Resistance of threshold implementations

In Section 3, we have analyzed the root cause of exploitation using SFA and SIFA against unprotected and side-channel protected implementations. In this section, we tackle this root cause and propose a design of threshold implementation that is resistant to these attacks. We then formally prove the resistance of our design against SFA and SIFA in a relatively relaxed adversary model.

4.1 Our design

In Section 3, a single fault on one well-chosen share before the computation of all shares of one unique output bit can break the non-completeness property. Note that the computation of other output shares is not affected. In hardware, this fault injection is possible in one of the following two scenarios: (1) there is a wire containing faulted share before the computation of all shares of an output bit and this wire is not used for the computation of shares of other output bits, or (2) some gates are shared between the component functions so that the fault can affect different shares of an output bit.

We consider the common realization using two register stages (also called *register-to-register function* by Dhooghe *et al.* [DOT24]), one for the input X and one for the output A, as in Figure 1. The first scenario does not make sense since faulting one input share (in the input register) cannot limit the effect to the computation of all shares of only one output bit, *i.e.*, the computation of other output shares is also affected. The second scenario, where some gates are shared between the component functions, is more reasonable. Faulting these gates may affect the computation of all shares of an output bit. In fact, Dhooghe *et al.* [DOT24] prevented this possibility by assuming that no gate is shared between the component functions and that the adversary is only able to fault one gate. However, this assumption does not seem to correspond to the practice when the TI is realized following the approach of Figure 1. In this approach, the computation of TI is done in one cycle. This means that the component functions are executed in parallel. If the adversary performs fault inductions via clock glitching, the faults likely affect gates of different component functions at the same time since their executions use the same clock. This could lead to a violation of the non-completeness property.

Taking a different approach, we aim to provide a realization such that the non-completeness property is maintained even in the presence of multiple faults. Inspired by the time sharing technique of Kumar *et al.* [VDB+24], we propose the idea of temporally separating the processing of component functions in time in order to guarantee the non-completeness property. Figure 5 depicts an example of our design for 3-share TI. In this work, we consider an adversary who is capable of injecting faults, regardless of fault model (*e.g.*, stuck-at-0 fault, stuck-at-1 fault, flipping fault, random fault, etc.), either into gates of a single component function (Subsection 4.2) or into variables in a single register stage (Subsection 4.3). The number of faults will be discussed later. We now formally define the goals to be achieved for our design: the resistance to SFA and SIFA.

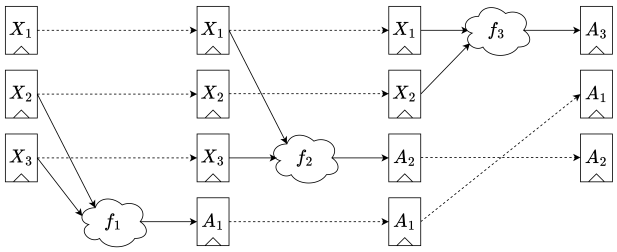


Fig. 5: Our design with 3-shares TI ($s = 3$).

For SFA, both the correct and faulty outputs are taken into account. Thus, there is no need for a countermeasure (*e.g.*, spatial or temporal redundancy) to filter the outputs. We will prove that a threshold implementation *alone* following our design can be resistant to SFA attacks. This means that we allow the

adversary to collect the faulty outputs. This approach is different from other protection schemes such as StaTI [DOT24], ParTI [SMG16], M&M [DAN+18], etc., whose approach is to prevent the release of faulty outputs by redundancy in order to protect the implementation against SFA attacks.

Definition 2 (SFA resistance). *Let* $\mathbf{A} = \mathbf{f}(\mathbf{X})$ *be the threshold implementation of a bijection that is prone to faults. This shared function* $\mathbf{f}$ *is said to be SFA-resistant if both the correct and faulty (*unfiltered*) unshared input* X*, and the correct and faulty (*unfiltered*) unshared output* A*, where* X *and* A *are unmasked from the* unfiltered $\mathbf{X}$ *and* $\mathbf{A}$*, are uniformly distributed.*

For SIFA we only consider correct outputs. Therefore, it needs a filter at the output to detect if the fault is effective or not. We assume that this filter exists in the case of SIFA. This filter could be a basic countermeasure such as temporal or spatial redundancy. Compared to previous schemes for SIFA protection [DDE+20,DOT24], whose idea is to design the implementation in such a way that the fault is ensured to propagate to the output, our approach is simpler. One might think that there is no need to have a filter so that the adversary can only perform SFA (which our design is resistant to). However, a filter is usually necessary in order to prevent other fault attacks, such as DFA.

Definition 3 (SIFA resistance). *Let* $\mathbf{A} = \mathbf{f}(\mathbf{X})$ *be the threshold implementation of a bijection that is prone to faults. This shared function* $\mathbf{f}$ *is said to be SIFA-resistant if both the correct (*filtered*) unshared input* X*, and the correct (*filtered*) unshared output* A*, where* X *and* A *are unmasked from the* filtered $\mathbf{X}$ *and* $\mathbf{A}$*, are uniformly distributed.*

4.2 Unexploitable faults in component function

We consider an adversary who is able to induce *any number of faults* into gates in a single component function. We will prove that our design is resistant to SFA (Theorem 1) and SIFA (Theorem 2).

Theorem 1. *Let* $\mathbf{A} = \mathbf{f}(\mathbf{X})$ *be the threshold implementation of a bijection following our design in Subsection 4.1. Let* $f_j \in \mathbf{f}$ *be a component function whose gates are prone to any number of faults. This shared function* $\mathbf{f}$ *is SFA-resistant.*

We rely on the non-completeness property (Property 2) to prove this theorem.

Proof (Theorem 1). Any faults induced into the gates within a single component function f_j will only affect its outcome A_j, not the shared input $\mathbf{X}$. Therefore, the unshared input X, which is decoded from $\mathbf{X}$, remains uniform. The outcome A_j contains only a single share of every output bit. Due to the nature of secret sharing, the unshared output A is independent of A_j and remains uniform as well. By Definition 2, we conclude that $\mathbf{f}$ is SFA-resistant. □

Theorem 2. *Let* $\mathbf{A} = \mathbf{f}(\mathbf{X})$ *be the threshold implementation of a bijection following our design in Subsection 4.1. Let* $f_j \in \mathbf{f}$ *be a component function whose gates are prone to any number of faults. This shared function* $\mathbf{f}$ *is SIFA-resistant.*

Proving the SIFA resistance is a bit more complicated because we have to eliminate the faulty outputs. Here is the proof sketch. We will first show that there exists a Boolean expression, denoted by ξ, representing the effectiveness of the faults (similar to $\xi = X^2 \wedge X_1^1$ in the analysis for the Keccak S-box in Section 3). Then, we will prove that ξ is independent of the unshared input X. This gives us the conditional probability of the unshared input (when the faults are ineffective): $\Pr[X = x|\xi = 0] = \Pr[X = x] = 2^{-n}$. Then, the probability of the correct unshared output can be easily derived since f is a bijection.

Let $\mathbf{X}_{\bar{j}}$ be the set of all input shares excluding the j-th share of every input bit. The non-completeness property (Property 2) states that each component function f_j is independent of one input share (usually the j-th share). Then, $\mathbf{X}_{\bar{j}}$ is considered as the input shares of f_j.

Lemma 1. *There exists a Boolean expression ξ of the shares involving in the computation of f_j (i.e., the shares in the set $\mathbf{X}_{\bar{j}}$), such that:*

- $\xi = 1$ *if and only if the faults are effective,*
- $\xi = 0$ *if and only if the faults are ineffective.*

Proof (Lemma 1). Recall that the faults are called effective when at least one unshared output bit is flipped (as stated in Definition 1). We consider faults induced into some gates during the computation of the component function f_j, which returns the j-th share of every output bit. If some of these shares are flipped, then their corresponding unshared output bits are flipped, and thus the faults are effective. We draw two truth tables with all possible values of the input shares $\mathbf{X}_{\bar{j}}$ and their corresponding output share A_j, one for the correct computation and one for the computation with the faults activated. By comparing the values of the output shares in the two tables, we determine the case where the faults are effective. The Boolean expression ξ takes as input the shares $\mathbf{X}_{\bar{j}}$, and produces a single output bit. The truth table of ξ includes all possible values of the input shares $\mathbf{X}_{\bar{j}}$ as the two previous tables and their corresponding output bit, which is determined as follows: if the input values correspond to the case where the faults are effective, then $\xi = 1$, otherwise, $\xi = 0$. The Boolean expression of ξ can be constructed from this truth table. □

Corollary 1. *The output of the Boolean expression ξ is independent of unshared input X, which is unmasked from $\mathbf{X}$.*

Corollary 1 can be easily verified as the Boolean expression ξ takes as input the set all input shares excluding the j-th share of every input bit, $\mathbf{X}_{\bar{j}}$. By the nature of secret sharing, ξ is independent of the unshared input X. We are now ready to prove Theorem 2.

Proof (Theorem 2). We will show that the correct (filtered) unshared input and the correct (filtered) unshared output are uniformly distributed (as stated in Definition 3). In the formulas, we will show that $\Pr[A = a|\xi = 0] = 2^{-n}$ for every $a \in [0, 2^n - 1]$, and $\Pr[X = x|\xi = 0] = 2^{-n}$ for every $x \in [0, 2^n - 1]$. By Corollary 1, the unshared input X and the Boolean expression ξ describing the

effectiveness of the faults are independent. We thus have $\Pr[X = x|\xi = 0] = \Pr[X = x] = 2^{-n}$. Since f is a bijection, the uniformity of the unshared input leads to the uniformity of the unshared output, $\Pr[A = a|\xi = 0] = 2^{-n}$. By Definition 3, we conclude that $\mathbf{f}$ is SIFA-resistant. □

4.3 Unexploitable faults in register

We consider an adversary who is capable of inducing a fault into a share X_j or A_j in a register stage. If we consider this fault in the bit level, n faults ($X_j^1, \ldots, X_j^n$, or $A_j^1, \ldots, A_j^n$) are allowed for the adversary. We will prove that our design is resistant to SFA (Theorem 3) and SIFA (Theorem 4).

Theorem 3. *Let $\mathbf{A} = \mathbf{f}(\mathbf{X})$ be the threshold implementation of a bijection following our design in Subsection 4.1. Let X_j (equivalently, $X_j^1, \ldots, X_j^n$ in bit level) or A_j (equivalently, $A_j^1, \ldots, A_j^n$ in bit level) be the share being faulted in a register stage. The shared function $\mathbf{f}$ is SFA-resistant.*

The proof of Theorem 3 relies on the non-completeness property (Property 2) and is quite similar to the proof of Theorem 1. We include the details of this proof in Section A.

Theorem 4. *Let $\mathbf{A} = \mathbf{f}(\mathbf{X})$ be the threshold implementation of a bijection following our design in Subsection 4.1. Let X_j (equivalently, $X_j^1, \ldots, X_j^n$ in bit level) or A_j (equivalently, $A_j^1, \ldots, A_j^n$ in bit level) be the share being faulted in a register stage. This shared function $\mathbf{f}$ is SIFA-resistant.*

To prove Theorem 4, we follow the same strategy as the proof of Theorem 2. The detailed proof is included in Section A.

5 Composition

In this section, we show that two threshold implementations (TI) that are SFA-resistant or SIFA-resistant can be composed. With this composition, we can construct a TI of a full cipher that is resistant to SFA and SIFA attacks.

Let $\mathbf{A} = \mathbf{f}(\mathbf{Y})$ and $\mathbf{Y} = \mathbf{g}(\mathbf{X})$ be the two s-share threshold implementations of two n-bit bijections $A = f(Y)$ and $Y = g(X)$. We denote by $\mathbf{A} = \mathbf{h}(\mathbf{X})$ where $\mathbf{h} = \mathbf{f} \circ \mathbf{g}$ the composition of $\mathbf{f}$ and $\mathbf{g}$ such that the output shares of $\mathbf{g}$ are the input shares of $\mathbf{f}$. Assume that either $\mathbf{g}$ or $\mathbf{f}$ is prone to faults in a component function or in a register stage as presented in Section 4. We will prove that $\mathbf{h}$ is resistant to SFA (Theorem 5) and SIFA (Theorem 6).

Theorem 5. *If $\mathbf{f}$ and $\mathbf{g}$ are SFA-resistant, then the composition $\mathbf{h} = \mathbf{f} \circ \mathbf{g}$ is SFA-resistant.*

To prove this theorem, we will show that both the (unfiltered) unshared input X and the (unfiltered) unshared output A of $\mathbf{h}$ are uniformly distributed.

Proof (Theorem 5). By assumption, $\mathbf{g}$ is SFA-resistant, its unshared input X (also the unshared input of $\mathbf{h}$) and unshared output Y thus are thus uniformly distributed. Recall that the output shares $\mathbf{Y}$ of $\mathbf{g}$ are the input shares of $\mathbf{f}$ and $\mathbf{f}$ is SFA-resistant by assumption. Since f is a bijection, the uniformity of the unshared input Y infers the uniformity of the unshared output A of $\mathbf{f}$ (also the unshared output of $\mathbf{h}$). By Definition 2, we conclude that $\mathbf{h}$ is SFA-resistant. □

For SIFA, we pay special attention to the composition since a filter (redundancy) is required to select the correct outputs. The naive idea is to embed a filter at the output of both $\mathbf{g}$ and $\mathbf{f}$, and assume that these two shared functions are both SIFA-resistant. However, we can improve this idea by leveraging the uniform property (Property 4) of TI. Indeed, a single filter at the end of the composition, *i.e.*, at the output of $\mathbf{f}$, is sufficient. We assume that $\mathbf{g}$ is SFA-resistant (since there is no filter at its output) and $\mathbf{f}$ is SIFA-resistant. Let us suppose that $\mathbf{g}$ is faulted, leading to the flips (errors) of some output shares in $\mathbf{Y}$. These errors will definitely be propagated to the output shares $\mathbf{A}$ of $\mathbf{f}$ because the shared function $\mathbf{f}$ is a bijection on the shares (Property 4). Therefore, a filter (redundancy) at the output of $\mathbf{f}$ can catch the faults induced into $\mathbf{g}$. This also implies that, for a full TI of a cipher following our design, a fault detection at the output of the cipher is sufficient. Compared to the fault propagation scheme using costly reversible operations of Daemen *et al.* [DDE+20], our approach using solely TI is much simpler.

Theorem 6. *If* $\mathbf{g}$ *is SFA-resistant and* $\mathbf{f}$ *is SIFA-resistant, then the composition* $\mathbf{h} = \mathbf{f} \circ \mathbf{g}$ *is SIFA-resistant.*

To prove this theorem, we will show that both the correct (filtered) unshared input X and the correct (filtered) unshared output A of $\mathbf{h}$ are uniformly distributed (as stated in Definition 3).

Proof (Theorem 6). By assumption, $\mathbf{g}$ is SFA-resistant, thus its (unfiltered) unshared output Y is uniformly distributed. Since the output shares $\mathbf{Y}$ of $\mathbf{g}$ are the input shares of $\mathbf{f}$ and $\mathbf{f}$ is SIFA-resistant by assumption, the (filtered) unshared output A is uniformly distributed. This implies that the (filtered) unshared input Y of $\mathbf{f}$ is uniform since f is a bijection. The uniformity of the filtered Y also implies the uniformity of the filtered X since g is a bijection. By Definition 3, we conclude that $\mathbf{h}$ is SIFA-resistant. □

6 Proof-of-concept PRESENT implementation

PRESENT [BKL+07] is an ultra-lightweight block cipher with two versions corresponding to the size of the master key: 80 bits and 128 bits, respectively. The block size of the two PRESENT versions is 64 bits. The cipher consists of 31 rounds, plus an additional whitening key at the end of the computation. Each round consists of three operations: a key addition, an S-box layer applied to 4-bit nibbles and a bit permutation layer.

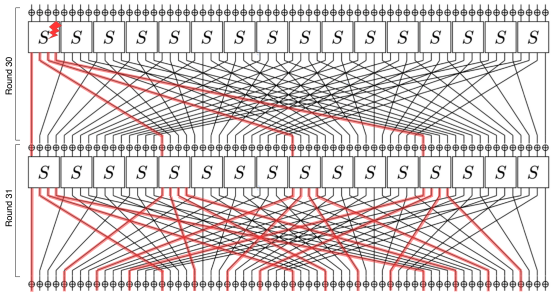
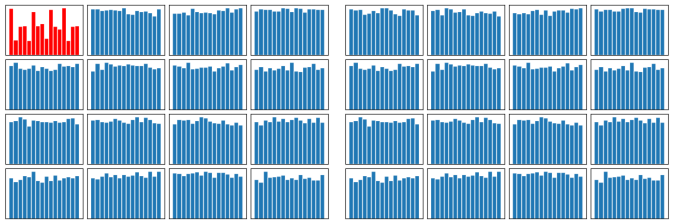
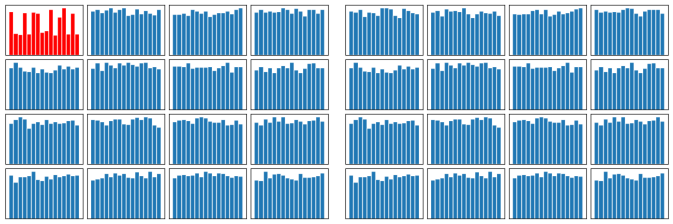


Fig. 6: Influence of fault in the last two rounds of PRESENT.



(a) SFA with correct key guess (b) SFA with wrong key guess

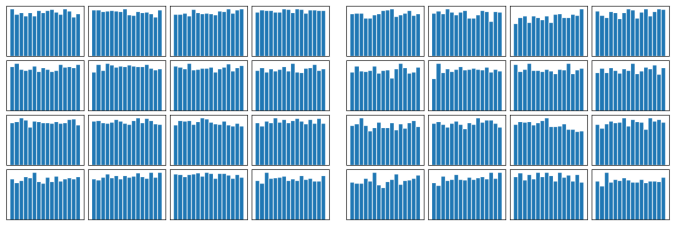


(c) SIFA with correct key guess (d) SIFA with wrong key guess

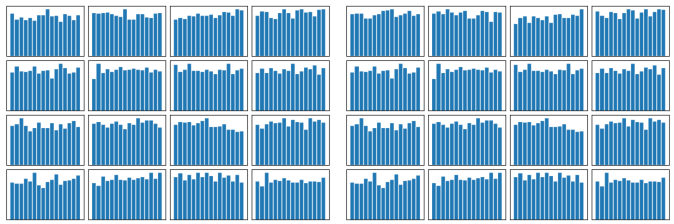
Fig. 7: Distribution of state nibbles after S-box in round 30 of the *vulnerable* implementation after collecting 5000 ciphertexts. The distributions corresponding to SFA use all 5000 ciphertexts. The distributions corresponding to SIFA use 3762 correct ciphertexts out of 5000.

We simulate the concept of our design with a software first-order TI with 3 shares, following the TI in the software of Sasdrich *et al.* [SBM18]. Similar to Poschmann *et al.* [PMK+11], the 4-bit S-box is decomposed into two quadratic functions $f \circ g$ to reduce the number of shares to three. The shared functions $\mathbf{A} = \mathbf{f}(\mathbf{Y})$ and $\mathbf{Y} = \mathbf{g}(\mathbf{X})$, where $\mathbf{f} = \{f_1, f_2, f_3\}$ and $\mathbf{g} = \{g_1, g_2, g_3\}$, can be found in Section B. To facilitate the simulation of faults, we implement the component functions of the S-box instead of using look-up tables as [SBM18]. The non-completeness property is ensured by declaring proper inputs and outputs for the component functions. The time sharing (Section 4) and the composition (Section 5) in our design are ensured by the order of the function calls.

We simulate a stuck-at-0 fault into a variable in the shared computation for the first S-box nibble in the 30th round. The impact of the fault is depicted in Figure 6. To apply SFA or SIFA, the attacker guesses 16 bits of the last round key, then computes backward from the set of collected ciphertexts to the faulted position. If the attack is successful, the attacker should observe a non-uniform distribution for the faulted nibble when the key guess is correct. Note that guessing four bits of the penultimate is not necessary as it does not affect the targeted non-uniform distribution. The simulation is done in two scenarios of the S-box implementation: (1) the approach that is vulnerable to SFA and SIFA as analyzed in Section 3 and (2) the approach of our design Section 4. No filter is required in the case of SFA, while a temporal redundancy (*i.e.*, the cipher is executed twice, and the two outputs are compared to detect faults) is applied before releasing the ciphertext in the case of SIFA.



(a) SFA with correct key guess (b) SFA with wrong key guess



(c) SIFA with correct key guess (d) SIFA with wrong key guess

Fig. 8: Distribution of state nibbles after S-box in round 30 of the *secure* implementation after collecting 5000 ciphertexts. The distributions corresponding to SFA use all 5000 ciphertexts. The distributions corresponding to SIFA use 2477 correct ciphertexts out of 5000.

In the first scenario, the shared S-box computation is arranged such that the stuck-at-0 fault induced into the share Y_1^4 affects the computation of all three shares of the output bit A^4 (see Equation 7). As expected, when the key guess is correct, we can observe in Figure 7 a strongly biased distribution of the first nibble, which can be exploited by SFA and SIFA.

In the second scenario, the shared S-box computation is implemented as our design in Figure 5, *i.e.*, the non-completeness property is guaranteed by the time

sharing. As expected, we cannot distinguish the distribution of the first nibble when the key guess is correct and when the key guess is wrong. For SIFA in particular, this also shows that a basic redundancy combined with TI is sufficient since the faults are naturally propagated to the output thanks to the uniformity property (as discussed in Section 5).

References

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

BDN+13. Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and first-order DPA resistant implementations of keccak. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2013.

BKK+20. Anubhab Baksi, Vinay B. Y. Kumar, Banashri Karmakar, Shivam Bhasin, Dhiman Saha, and Anupam Chattopadhyay. A novel duplication based countermeasure to statistical ineffective fault analysis. In Joseph K. Liu and Hui Cui, editors, *ACISP 20: 25th Australasian Conference on Information Security and Privacy*, volume 12248 of *Lecture Notes in Computer Science*, pages 525–542, Perth, WA, Australia, November 30 – December 2, 2020. Springer, Cham, Switzerland.

BKL+07. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466, Vienna, Austria, September 10–13, 2007. Springer, Berlin, Heidelberg, Germany.

BS97. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Berlin, Heidelberg, Germany.

BS03. Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). In Rebecca Wright, editor, *FC 2003: 7th International Conference on Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181, Guadeloupe, French West Indies, January 27–30, 2003. Springer, Berlin, Heidelberg, Germany.

CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Heidelberg, Germany.

DAN+18. Lauren De Meyer, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. M&M: Masks and macs against physical attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):25–50, 2018.

DDE+20. Joan Daemen, Christoph Dobraunig, Maria Eichlseder, Hannes Gross, Florian Mendel, and Robert Primas. Protecting against statistical ineffective fault attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):508–543, 2020.

DDRT12. Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic transient faults injection on a hardware and a software implementations of AES. In Guido Bertoni and Benedikt Gierlichs, editors, *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*, pages 7–15. IEEE Computer Society, 2012.

DEG+18. Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 315–342, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.

DEK+16. Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Victor Lomné, and Florian Mendel. Statistical fault attacks on nonce-based authenticated encryption schemes. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 369–395, Hanoi, Vietnam, December 4–8, 2016. Springer, Berlin, Heidelberg, Germany.

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.

DLV03. P. Dusart, G. Letourneux, and O. Vivolo. Differential fault analysis on a.e.s. Cryptology ePrint Archive, Report 2003/010, 2003.

DOT24. Siemen Dhooghe, Artemii Ovchinnikov, and Dilara Toprakhisar. StaTI: Protecting against fault attacks using stable threshold implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(1):229–263, 2024.

FJLT13. Thomas Fuhr, Eliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on aes with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 108–118, 2013.

Gir03. Christophe Giraud. DFA on AES. Cryptology ePrint Archive, Report 2003/008, 2003.

GMK16. Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. Cryptology ePrint Archive, Report 2016/486, 2016.

GMO01. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261, Paris, France, May 14–16, 2001. Springer, Berlin, Heidelberg, Germany.

GP99. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172, Worcester, Massachusetts, USA, August 12–13, 1999. Springer, Berlin, Heidelberg, Germany.

ISW03. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Berlin, Heidelberg, Germany.

Kec15. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute of Standards and Technology, NIST FIPS PUB 202, U.S. Department of Commerce, 2015. `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202`.

KJJ99. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Heidelberg, Germany.

Koc96. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Berlin, Heidelberg, Germany.

MSS06. Amir Moradi, Mohammad T. Manzuri Shalmani, and Mahmoud Salmasizadeh. A generalized method of differential fault attack against AES cryptosystem. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 91–100, Yokohama, Japan, October 10–13, 2006. Springer, Berlin, Heidelberg, Germany.

NRR06. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS 06: 8th International Conference on Information and Communication Security*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545, Raleigh, NC, USA, December 4–7, 2006. Springer, Berlin, Heidelberg, Germany.

PMK+11. Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-channel resistant crypto for less than 2,300 GE. *Journal of Cryptology*, 24(2):322–345, April 2011.

RBN+15. Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Berlin, Heidelberg, Germany.

RDB+18. Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel P. Smart. CAPA: The spirit of beaver against physical attacks. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of

Lecture Notes in Computer Science, pages 121–151, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

SA03. Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12, Redwood Shores, CA, USA, August 13–15, 2003. Springer, Berlin, Heidelberg, Germany.

SBM18. Pascal Sasdrich, René Bock, and Amir Moradi. Threshold implementation in software - case study of PRESENT. In Junfeng Fan and Benedikt Gierlichs, editors, *COSADE 2018: 9th International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 10815 of *Lecture Notes in Computer Science*, pages 227–244, Singapore, April 23–24, 2018. Springer, Cham, Switzerland.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

SMG16. Tobias Schneider, Amir Moradi, and Tim Güneysu. ParTI – towards combined hardware countermeasures against side-channel and fault-injection attacks. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 302–332, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Berlin, Heidelberg, Germany.

VDB+24. Dilip Kumar S. V., Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Time sharing - A novel approach to low-latency masking. Cryptology ePrint Archive, Report 2024/925, 2024.

ZDCT13. Loïc Zussa, Jean-Max Dutertre, Jessy Clédière, and Assia Tria. Power supply glitch induced faults on FPGA: an in-depth analysis of the injection mechanism. In *2013 IEEE 19th International On-Line Testing Symposium (IOLTS), Chania, Crete, Greece, July 8-10, 2013*, pages 110–115. IEEE, 2013.

A Detailed proofs

Proof (Theorem 3). If the faulted share is X_j, it is independent of the unshared input X by the nature of secret sharing. The uniformity of the (unfiltered) unshared input X thus remains unchanged. In TI, the share X_j does not involve in at least a component function, typically f_j, which computes the $j-th$ share of every output bit (Property 2, non-completeness). Hence, the (unfiltered) unshared output A is independent of the fault's effect, and thus is uniform. By Definition 2, we conclude that $\mathbf{f}$ is SFA-resistant.

If the faulted share is A_j, it is independent of the unshared output A by the nature of secret sharing. The uniformity of the (unfiltered) unshared output A thus remains unchanged. As A_j is faulted in a register stage, the input shares are not affected. The uniformity of the (unfiltered) unshared input X thus remains unchanged. By Definition 2, we conclude that $\mathbf{f}$ is SFA-resistant. □

Proof (Theorem 4). Following the same proof strategy for Theorem 2, we will show that $\Pr[A = a|\xi = 0] = 2^{-n}$ for every $a \in [0, 2^n - 1]$ and $\Pr[X = x|\xi = 0] = 2^{-n}$ for every $x \in [0, 2^n - 1]$, where $\xi = 0$ represents the ineffective fault.

We first construct the Boolean expression ξ describing the effectiveness of the fault similar to the proof of Lemma 1. We draw two truth tables with all possible values of the input shares $\mathbf{X}$ and their resulted unshared output A, one for the correct computation and one for the computation with the fault activated. By comparing the values of the unshared output A in the two tables, we determine the cases where the fault is effective. The Boolean expression ξ takes as input the shares $(X_j^1, \ldots, X_j^n$ or $A_j^1, \ldots, A_j^n)$ and produces a single output bit. The truth table of ξ includes all possible values of the shares $(X_j^1, \ldots, X_j^n$ or $A_j^1, \ldots, A_j^n)$ and the corresponding output bit which is determined as follows: if the value of the faulted share corresponds to the case where the fault is effective, then $\xi = 1$, otherwise, $\xi = 0$. The Boolean expression of ξ can be constructed from this truth table.

If the faulted share is X_j, we have that ξ is independent of the unshared input X. Thus, $\Pr[X = x|\xi = 0] = \Pr[X = x] = 2^{-n}$. Since f is a bijection, the uniformity of the unshared input leads to the uniformity of the unshared output, $\Pr[A = a|\xi = 0] = 2^{-n}$. By Definition 3, we conclude that $\mathbf{f}$ is SIFA-resistant.

If the faulted share is A_j, we have that ξ is independent of the unshared output A. Thus, $\Pr[A = a|\xi = 0] = \Pr[A = a] = 2^{-n}$. Since f is a bijection, the uniformity of the unshared output leads to the uniformity of the unshared input, $\Pr[X = x|\xi = 0] = 2^{-n}$. By Definition 3, we conclude that $\mathbf{f}$ is SIFA-resistant.

□

B Threshold implementation of PRESENT S-box

$$
\begin{aligned}
Y_1 &= g_1(X_2, X_3)\\
Y_1^1 &= X_2^1 \oplus X_2^2 \oplus X_2^3 \oplus X_2^4 \oplus X_3^2 \oplus X_2^3X_3^2 \oplus X_2^2X_3^3 \oplus X_3^2X_3^3\\
Y_1^2 &= X_2^1\\
Y_1^3 &= 1 \oplus X_2^1 \oplus X_2^4 \oplus X_3^2 \oplus X_2^3X_3^2 \oplus X_2^2X_3^3 \oplus X_3^2X_3^3\\
Y_1^4 &= 1 \oplus X_2^1 \oplus X_2^2 \oplus X_3^2 \oplus X_2^3X_3^2 \oplus X_2^4X_3^2 \oplus X_2^2X_3^3 \oplus X_2^4X_3^3 \oplus X_3^2X_3^3\\
&\quad \oplus X_2^2X_3^4 \oplus X_2^3X_3^4 \oplus X_3^2X_3^4 \oplus X_3^3X_3^4
\end{aligned}
$$

$$
\begin{aligned}
Y_2 &= g_2(X_3, X_1)\\
Y_2^1 &= X_3^1 \oplus X_3^2 \oplus X_3^3 \oplus X_3^4 \oplus X_1^2 \oplus X_3^3X_1^2 \oplus X_3^2X_1^3 \oplus X_1^2X_1^3\\
Y_2^2 &= X_3^1\\
Y_2^3 &= 1 \oplus X_3^1 \oplus X_3^4 \oplus X_1^2 \oplus X_3^3X_1^2 \oplus X_3^2X_1^3 \oplus X_1^2X_1^3\\
Y_2^4 &= 1 \oplus X_3^1 \oplus X_3^2 \oplus X_1^2 \oplus X_3^3X_1^2 \oplus X_3^4X_1^2 \oplus X_3^2X_1^3 \oplus X_3^4X_1^3 \oplus X_1^2X_1^3\\
&\quad \oplus X_3^2X_1^4 \oplus X_3^3X_1^4 \oplus X_1^2X_1^4 \oplus X_1^3X_1^4
\end{aligned}
$$

$$
\begin{aligned}
Y_3 &= g_3(X_1, X_2)\\
Y_3^1 &= X_1^1 \oplus X_1^2 \oplus X_1^3 \oplus X_1^4 \oplus X_2^2 \oplus X_1^3X_2^2 \oplus X_1^2X_2^3 \oplus X_2^2X_2^3\\
Y_3^2 &= X_1^1\\
Y_3^3 &= 1 \oplus X_1^1 \oplus X_1^4 \oplus X_2^2 \oplus X_1^3X_2^2 \oplus X_1^2X_2^3 \oplus X_2^2X_2^3\\
Y_3^4 &= 1 \oplus X_1^1 \oplus X_1^2 \oplus X_2^2 \oplus X_1^3X_2^2 \oplus X_1^4X_2^2 \oplus X_1^2X_2^3 \oplus X_1^4X_2^3 \oplus X_2^2X_2^3\\
&\quad \oplus X_1^2X_2^4 \oplus X_1^3X_2^4 \oplus X_2^2X_2^4 \oplus X_2^3X_2^4
\end{aligned}
$$

$$
\begin{aligned}
A_1 &= f_1(Y_2, Y_3)\\
A_1^1 &= Y_2^1\\
A_1^2 &= Y_2^3 \oplus Y_2^4 \oplus Y_2^4Y_3^2 \oplus Y_2^2Y_3^4 \oplus Y_3^2Y_3^4\\
A_1^3 &= 1 \oplus Y_2^1 \oplus Y_2^2 \oplus Y_2^3 \oplus Y_2^4Y_3^3 \oplus Y_2^3Y_3^4 \oplus Y_3^3Y_3^4\\
A_1^4 &= Y_2^3 \oplus Y_2^4Y_3^2 \oplus Y_2^2Y_3^4 \oplus Y_3^2Y_3^4
\end{aligned}
$$

$$
\begin{aligned}
A_2 &= f_2(Y_3, Y_1)\\
A_2^1 &= Y_3^1\\
A_2^2 &= Y_3^3 \oplus Y_3^4 \oplus Y_3^4Y_1^2 \oplus Y_3^2Y_1^4 \oplus Y_1^2Y_1^4\\
A_2^3 &= 1 \oplus Y_3^1 \oplus Y_3^2 \oplus Y_3^3 \oplus Y_3^4Y_1^3 \oplus Y_3^3Y_1^4 \oplus Y_1^3Y_1^4\\
A_2^4 &= Y_3^3 \oplus Y_3^4Y_1^2 \oplus Y_3^2Y_1^4 \oplus Y_1^2Y_1^4
\end{aligned}
$$

$$
\begin{aligned}
A_3 &= f_3(Y_1, Y_2)\\
A_3^1 &= Y_1^1\\
A_3^2 &= Y_1^3 \oplus Y_1^4 \oplus Y_1^4Y_2^2 \oplus Y_1^2Y_2^4 \oplus Y_2^2Y_2^4\\
A_3^3 &= 1 \oplus Y_1^1 \oplus Y_1^2 \oplus Y_1^3 \oplus Y_1^4Y_2^3 \oplus Y_1^3Y_2^4 \oplus Y_2^3Y_2^4\\
A_3^4 &= Y_1^3 \oplus Y_1^4Y_2^2 \oplus Y_1^2Y_2^4 \oplus Y_2^2Y_2^4
\end{aligned}
$$

C Additional illustration

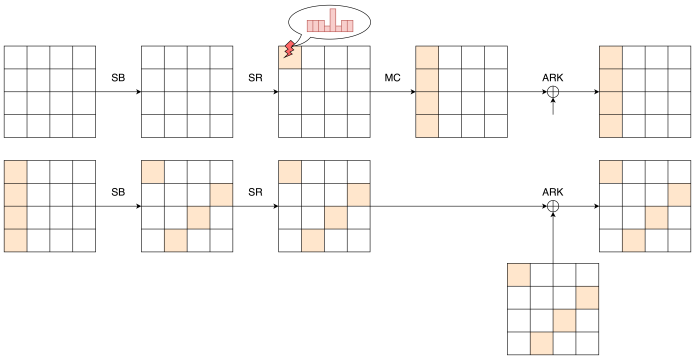


Fig. 9: Illustration of SFA in the last two rounds of AES.