Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures

Christoph Dobraunig[1], Maria Eichlseder[1], Hannes Gross[1], Stefan Mangard[1], Florian Mendel[2] and Robert Primas[1]*

[1] Graz University of Technology, Austria
robert.primas@iaik.tugraz.at
[2] Infineon Technologies AG, Germany

Abstract. Implementation attacks like side-channel and fault attacks are a threat for deployed devices especially if an attacker has physical access to a device. As a consequence, devices like smart cards usually provide countermeasures against implementation attacks, such as masking against side-channel attacks and detection-based countermeasures like temporal redundancy against fault attacks. In this paper, we show how to attack implementations protected with both masking and detection-based fault countermeasures by using statistical ineffective fault attacks using a single fault induction per execution. Our attacks are largely unaffected by the deployed protection order of masking and the level of redundancy of the detection-based countermeasure. Our observations refute the intuition that masking is a viable countermeasure against biased faults, statistical fault attacks, or statistical ineffective fault attacks.

Keywords: Implementation attack · Fault attack · SFA · SIFA

1 Introduction

Fault attacks and passive side-channel attacks, like power [KJJ99] or EM analysis [QS01], are very powerful attacks on implementations of cryptographic algorithms. Therefore, devices like smart cards and IoT devices, which are potentially physically accessible by an attacker, implement corresponding countermeasures. In the case of symmetric cryptography, the typical approach to protect an implementation against these attacks is to use masking and redundancy mechanisms. Masking is the most prominent and widely deployed countermeasure against passive side-channel attacks. There exists a wide range of masking schemes for software and hardware [RP10, RBN+15, GMK17] providing protection up to a given protection order.

For redundancy mechanisms against fault attacks, there has been less research compared to masking. A standard approach to counteract fault attacks is to use temporal or spacial redundancy mechanisms in order to do error detection. The basic idea is to do critical operations multiple times and to release the output only if the outputs of all redundant computations match. There are also works that directly combine masking and redundant encoding techniques for error detection [SMG16]. The standard reasoning when combining masking and error-detection mechanisms is that their effects add up. For example, assume an implementation of AES is protected by a masking scheme of protection order d and all encryption/decryption operations are always computed twice, compared, and the output is only released if the output of both operations match. In this case, the typical assumption is that this implementation is secure against a single fault induction because this is detected

*The list of authors is in alphabetical order (https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf) by the redundant computation and secure against side-channel attacks of up to order d due to the masking scheme.

This reasoning is valid for fault attacks that exploit faulty outputs of a cryptographic algorithm in order to reveal the key. The most prominent attacks of this type are Differential Fault Attacks (DFA) [BS97] and Statistical Fault Attacks (SFA) [FJLT13]. However, some variants of fault attacks are based on a different approach. Ineffective fault attacks (IFA) [Cla07] and Statistical Ineffective Fault Attacks (SIFA) [DEK+18] exploit those outputs of a cipher that are correct although a fault induction has been performed. While IFA requires exact knowledge about the location and effect of a fault, SIFA has much more relaxed requirements, thus allowing to exploit noisy faults whose exact effect is unknown to the attacker. The basic idea of SIFA is to repeatedly execute a cryptographic operation with a fixed key and to apply a fault induction for each execution. The attacker then collects those outputs of the cryptographic operation, where the fault induction has not changed any intermediate value. Given that the implementation is protected by an error-detection scheme, such as redundant computations, this corresponds exactly to the valid outputs of the system. In fact, the error detection that is implemented against DFA provides exactly the filtering of the outputs that is needed to apply SIFA or IFA.

So far, implementations combing masking and error-detection schemes were typically thought to be secure against attacks exploiting single ineffective faults due to the masking, as discussed for example by Clavier for IFA [Cla07]. It was typically assumed that all shares representing an intermediate value would need to be faulted to exploit ineffective faults and it was an open question whether this can be done efficiently in practice. In this work, we show that SIFA attacks are much more powerful than expected so far. Our central contribution is to show that SIFA is not only independent of the degree of redundancy, but also essentially independent of the degree of masking for typical masked implementations. SIFA attacks can be performed efficiently in practice on masked implementations with error detection by using a simple glitching device.

We demonstrate that faulting a single share is sufficient to induce a bias in an unshared intermediate value, which can then be exploited with a statistical analysis based on SIFA. More specifically, we induce a biased fault in one share during the computation of an S-box. Unlike classical fault attacks, attackers cannot directly use this fault as a distinguisher for the key recovery attack: they cannot recover this intermediate value from observing the ciphertext and guessing parts of the key, but can only recover the unshared output of the S-box (Figure 1). We analyze the impact of the local fault on the unshared output for several different S-boxes (including the AES S-box), fault distributions, masking schemes, and protection orders, as well as other fault countermeasures. We conclude that in all analyzed cases, a simple fault setup with a single fault attempt per encryption is sufficient to recover part of the key, given a suitable number of faulted encryptions. This number depends on the precision of the fault and the deployed countermeasures; for example, 1000 encryptions with a cheap clock glitching setup are sufficient for an 8-bit AES software implementation protected with 10th-order masking and arbitrary temporal redundancy.

2 Background on Statistical Faults

In this section, we cover the necessary background of Statistical Fault Attacks (SFA) and Statistical Ineffective Fault Attacks (SIFA). While we show results for both attacks in our practical evaluation, the more practically relevant attack is SIFA, since it cannot be prevented easily using standard fault countermeasures.

2.1 Statistical Fault Attacks

Statistical Fault Attacks (SFA), originally proposed by Fuhr et al. [FJLT13], present a way to recover an AES key with, compared to other fault attacks, quite relaxed assumptions

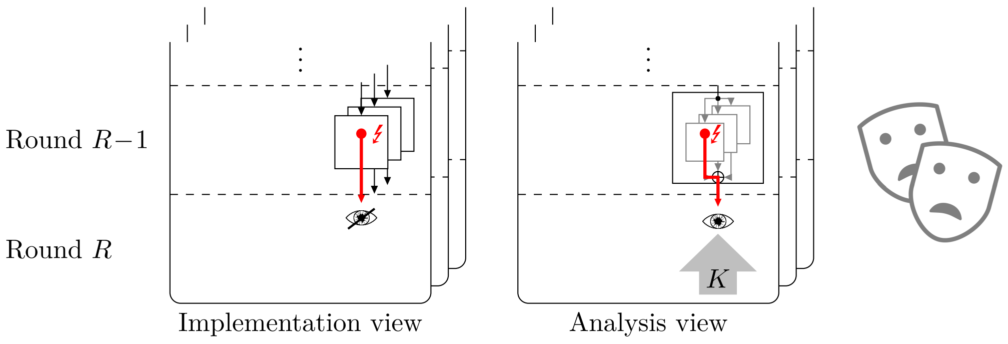


Figure 1: Biased fault attacks on masked, redundant implementations: High-level view.

on the fault induction. In fact, SFA solely requires the attacker to let one AES state byte in round 9 follow any (possibly unknown) non-uniform distribution. This can be achieved in practice, e.g., by using several types of stuck-at faults or instruction skips.

Given a set of faulty ciphertexts, SFA works by partially decrypting every ciphertext back to the faulted state byte S_9 in round 9. This requires guessing 32 bits of the last round key K_{10}, as well as the calculation of the inverse ShiftRows, SubBytes, and MixColumns operation:

$$S_9 = \mathsf{MC}^{-1} \circ \mathsf{SB}^{-1} \circ \mathsf{SR}^{-1}(C \oplus K_{10}).$$

Guessing the penultimate round key (K_9) is not needed, since it does not influence the non-uniformity of S_9. For each key guess the distance of the distribution of S_9 to a uniform distribution can now be measured, e.g., by using the χ^2-statistic (CHI) or the closely related Squared Euclidean Imbalance (SEI). For a sufficient number of evaluated ciphertexts, the key guess corresponding to the highest CHI or SEI statistic is most likely correct. The necessary number of ciphertexts depends on the strength of the bias, i.e., the distance of the biased distribution p from the uniform distribution, which is quantified by the capacity $C(p)$. If the bias is small, then the necessary number of faulted ciphertexts is inverse proportional to $C(p)$; we refer to [DEK+18] for a more detailed discussion.

One obvious downside of SFA, from a practical perspective, is that it relies on exploiting faulty ciphertexts. This is problematic since most cryptographic implementations that operate with critical data can be expected to have various countermeasures against implementation attacks in place. A fault countermeasure like temporal redundancy would already prevent SFA using single fault inductions. While an adoption of SFA to some fault countermeasures is possible, a significantly more powerful attacker would be required. Hence the threat of SFA to such protected implementations is limited.

2.2 Statistical Ineffective Fault Attacks

Statistical Ineffective Fault Attacks (SIFA) [DEK+18] allow an attacker to circumvent many popular fault countermeasures. As the name suggests, and in contrast to SFA, SIFA solely relies on exploiting faulted encryptions where the induced faults are ineffective and the obtained (filtered) ciphertexts are hence always correct. The basic observation of SIFA is that biased ineffective faults lead to a non-uniform distribution of intermediate values, that can be exploited in a key-recovery attack. The data complexity of the attack then depends on the strength of the bias in the targeted intermediate variable and the necessary number of faulted encryptions to obtain sufficiently many ineffective samples.

SIFA is applicable just as easily for more than two redundant computations, since only one computation needs to be faulted. In fact, the authors from [DEK+18] show the applicability of SIFA not only against redundancy countermeasures, but also against infective countermeasures. In the latter case, more faulted encryptions are necessary, but the presented attacks are still efficient.

Consider an AES with simple temporal redundancy and an attacker that is able to force a certain byte in round 9 to follow a (to the attacker unknown) non-uniform distribution. If the attacker is faulting only one of the redundant computations, the attacker will eventually observe correct ciphertexts where the induced fault was ineffective. This filtered set of correct ciphertexts will, when partially decrypted using a correct key guess, show the same bias in the faulted byte as the originally forced non-uniform distribution after fault induction. All that is left to do now for the attacker is to perform the same key recovery as in SFA. The basic intuition here is that certain values of the faulted byte lead to ineffective faults and hence correct ciphertexts more often than others. As a consequence, when decrypting this filtered set of correct ciphertexts using a correct key guess, certain values in the faulted byte will show up more frequently than others. Still, knowledge about the exact effect of the fault induction (and the resulting non-uniformity) is not required since the SEI or CHI metric measures the distance of any distribution to a uniform distribution.

This fact will be especially important when SIFA is performed against masked implementations (cf. Section 3). Here it is almost impossible to predict the actual effect of a fault induction if the attacked implementation is unknown. At the same time it is still comparably easy to cause any joint non-uniform distribution over all shares of an intermediate variable as discussed in the upcoming sections. However, as we will see, it is beneficial to change the usual view that faults just manipulate values. Instead, as introduced in Subsection 3.4, it is beneficial to use the view that faults change the actual function that is computed in order to gasp the underlying principles exploited in our attack.

3 Faults on Masking

In this section, we study the influence of single faults on masked AND gates and subsequently on masked S-boxes. To do so, we first briefly recapitulate the fundamentals of masking. Then, we discuss how faults influence the distribution of unmasked values by taking masked AND gates as an example. After that, we evaluate how single faults influence masked implementations of several S-boxes. Finally, we have a closer look at the internal activities in a faulted masked S-box that allow an SFA and SIFA and argue that it should be almost always possible to influence the distribution of input and output values of masked S-boxes, so that an attacker can exploit this behavior using techniques introduced for SFA and SIFA.

For easier understanding why single faults on a single share can cause a bias in unshared values, we consider very simple fault models such as stuck-at faults in the following exposition. However, it is important to note that the attack approach generalizes efficiently to noisy, unpredictable and imprecise faults. For a discussion of how the attack complexity scales under the influence of noise, we refer to Section 5 and the SIFA analysis [DEK+18].

3.1 Concept of Masking

The goal of masking is to randomize the representation of security-sensitive data in each execution to counteract side-channel analysis by making the produced side-channel leakage, such as power consumption or electromagnetic emanation, independent of the underlying data. The most popular masking approaches are Boolean masking schemes, which are formed over finite field arithmetic in GF(2^n).

Algorithm 1 : Masked GF(2^n) multiplication according to Ishai et al. [ISW03] (ISW)

```
Input: $x_0, \dots x_d$, $y_0, \dots y_d \in \mathrm{GF}(2^n)$
Output: $q_0, \dots q_d \in \mathrm{GF}(2^n)$
1: for $i = 0$ to $d$ do
2:   for $j = i + 1$ to $d$ do
3:     $r_{i,j} \overset{?}{\leftarrow} \mathrm{GF}(2^n)$
4:     $t_{i,j} \leftarrow r_{i,j}$
5:     $t_{j,i} \leftarrow r_{i,j} \oplus x_i y_j \oplus x_j y_i$
6: for $i = 0$ to $d$ do
7:   $q_i \leftarrow x_i y_i$
8:   for $j = 0$ to $d$ do
9:     if $i \neq j$ then
10:      $q_i \leftarrow q_i \oplus t_{i,j}$
```

In Boolean masking, a sensitive variable x is split into a number of so-called shares (denoted x_i) which, when considered on their own or in conjunction of up to d shares, are statistically independent of the unshared variable x. This degree of independence is usually referred to as the protection order d and requires to split each variable with sensitive information into at least $d + 1$ shares. The shares are uniformly random in each execution, but at any time, it is ensured that the sum over all shares again results in the unshared variable x:

$$x = x_0 \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_d \,.$$

In a similar manner, functions over shared variables are split into component functions $\mathrm{f}_i(\dots)$ such that again a correct and secure sharing of the original function is established:

$$\mathrm{f}(x, y) = \mathrm{f}_0(\dots) \oplus \mathrm{f}_1(\dots) \oplus \mathrm{f}_2(\dots) \oplus \cdots \oplus \mathrm{f}_d(\dots) \,.$$

Throughout the entire implementation, a proper separation of shares and of the output of the component functions needs to be ensured in order to not violate the d^{th}-order independence, which is commonly expressed in the probing model of Ishai et al. [ISW03]. In the probing model, an attacker is modeled with the ability to probe up to d intermediate results of the masked implementation. An implementation is said to be secure if the probing attacker cannot gain any statistical advantage in guessing any secret variable by combining the probed results in an arbitrary manner. While this share separation can be easily ensured for functions which are linear over GF(2^n) – for example, the masked calculation of $x \oplus y$ can be performed share-wise $(x_i \oplus y_i)$ –, the secure implementation of nonlinear functions usually requires the introduction of fresh randomness.

As an example for a shared implementation of a nonlinear function, we consider the generic masked multiplication algorithm by Ishai et al. [ISW03] (Algorithm 1). In order to securely calculate $q = xy$, each of the $d + 1$ shares of x is multiplied with each of the shares of y, resulting in $(d + 1)^2$ multiplication terms. Subsequently, the multiplication terms are summed up together with fresh random variables denoted $r_{i,j}$, and distributed over the output shares q_i.

A first-order masked GF(2) multiplication, which corresponds to the calculation of an AND gate, is given in Equation 1:

$$\begin{aligned} q_0 &= x_0 y_0 \oplus r_{0,1} \\ q_1 &= x_1 y_1 \oplus (r_{0,1} \oplus x_0 y_1 \oplus x_1 y_0) \end{aligned} \tag{1}$$

A uniform distribution of each of the shares of q is ensured by the random r shares. In general, the joint distribution of any d shares of q in the masked multiplication algorithm is uniform, or in other words, any d shares are independently and identically (uniformly) distributed. It thus appears as if in order to insert a bias in the underlying unshared value, an attacker would need to insert a biased fault in either each share of x or y, or to insert a bias in each of the component functions in the calculation of q. However, in the following, we show that this intuition is not true.

3.2 Faulting Masked AND Gates

We first note that the calculation of the AND function itself has a probability of 25% for q to be 1. An attacker therefore successfully biases the masked AND function if the probability of q to be 1 is more or less likely than 25%. As an example, we consider an attacker who can skip any AND calculation in Equation 1, for instance the first AND calculating x_0y_0 in q_0. The shared function then effectively calculates q $(= q_0 \oplus q_1)$ to be $x_1y_1 \oplus x_0y_1 \oplus x_1y_0$ which has a probability of 37.5% to be 1. If the attacker would instead introduce a fault that skips the addition of the uniformly random bit $r_{0,1}$ in q_0, then the distribution of q is again biased since the probability of observing a 1 changes from 25% to 50%.

We observe the same biases when looking at single faults for other masked AND gates, like the one used in the CMS scheme of Reparaz et al. [RBN+15] or in the Domain-Oriented Masking scheme by Gross et al. [GMK16, GMK17]. This same bias behavior results from the fact that these masked ANDs calculate the same terms x_iy_j. The masked ANDs only differ in the arrangement of x_iy_j in q_0 and q_1, and the amount of used fresh randomness. Since q is equal to $q_0 \oplus q_1$, the arrangement of the terms has no influence on the bias behavior of q, and a fault of an addition of a single random r bit has the same impact on all masked ANDs.

Another prominent protection mechanism falling in the category of masking schemes are threshold implementations [NRR06]. Threshold implementations use an increased number of shares to achieve first-order side-channel resistance without requiring fresh randomness. In order to explore the impact on threshold implementations, we look at a four-share realization of a first-order masked AND gate by Nikova et al. [NRR06]:

$$\begin{aligned}
q_0 &= (x_2 \oplus x_3)(y_1 \oplus y_2) \oplus y_1 \oplus y_2 \oplus y_3 \oplus x_1 \oplus x_2 \oplus x_3 \\
q_1 &= (x_0 \oplus x_2)(y_0 \oplus y_3) \oplus y_0 \oplus y_2 \oplus y_3 \oplus x_0 \oplus x_2 \oplus x_3 \\
q_2 &= (x_1 \oplus x_3)(y_0 \oplus y_3) \oplus y_1 \oplus x_1 \\
q_3 &= (x_0 \oplus x_1)(y_1 \oplus y_2) \oplus y_0 \oplus x_0
\end{aligned} \tag{2}$$

For this shared AND gate, we perform two experiments. In the first experiment, we have a look at the distribution of the output $q = q_0 \oplus q_1 \oplus q_2 \oplus q_3$ assuming an instruction skip. In the second, we fix one input share x_0 to zero and look what happens.

For the instruction skip, we assume that in q_0 one instruction is skipped and so $q_0 = (x_2)(y_1 \oplus y_2) \oplus y_1 \oplus y_2 \oplus y_3 \oplus x_1 \oplus x_2 \oplus x_3$ is calculated, the other shares are processed correctly. In this case, we observe that for all 256 possible values of the shared input, the unshared output is 160 times (62.5%) 0 and 96 (37.5%) times 1, a clear deviation of the value an unfaulted AND should have.

Next, we fix x_0 to zero and perform the computations according to (2) for all 256 possible values the shared input can take. If we now look at q, we see that 192 times a 0 (75%) appears and 64 times a 1 (25%), which corresponds to the distribution of a correct AND gate. However, if we only consider correct computations of $q = x \cdot y$, we observe that only 192 out of 256 computations are performed correctly. For those correct computations q is 160 times a 1 (83.3%) and 32 times a 0 (16.6%). This "filtered" distribution is the one an attacker would observe and exploit in the case of a SIFA [DEK+18].

In the next section, we will discuss the consequences of our observations with respect to S-boxes.

3.3 Faulting Masked S-boxes

In this section, we discuss how single faults influence the behavior of S-boxes. It is worth mentioning that our selection of masked S-boxes is arbitrary and does not imply that those S-boxes are weaker or more susceptible to fault attacks than others. We have selected those S-boxes because they have a simple and compact description. We will start with a compact 4-bit S-box called Sbox13 [UDCI+11] shown in Figure 2. We have implemented a masked implementation of this S-box in software by using a four-shared threshold implementation of AND (2), OR and XOR. We target exclusively the AND labeled with q, x, and y in Figure 2.

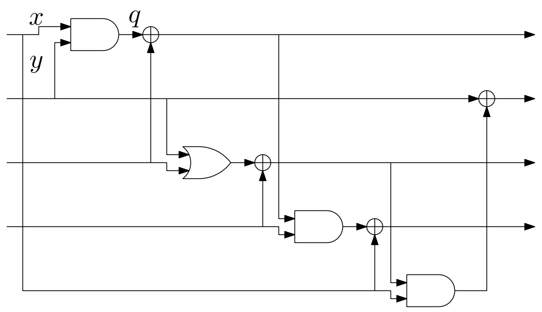


Figure 2: Schematic of the 4×4 S-box: Sbox13 [UDCI+11].

For the first experiment, we assume an instruction skip that alters the execution of only the first AND of the S-box, changing the calculation of one share q_0 to $q_0 = (x_2)(y_1 \oplus y_2) \oplus y_1 \oplus y_2 \oplus y_3 \oplus x_1 \oplus x_2 \oplus x_3$. In Figure 3, we record the distribution of each unshared input value and each unshared output value for each of the $2^{4 \cdot 4} = 65536$ shared input combinations, which can be exploited by an SFA [FJLT13], as well as the "filtered" distribution for ineffectively faulted S-box transitions, which can be exploited by a SIFA [DEK+18]. This "filtered" distribution stems from the subset of transitions, for which the induction of the fault has no influence on the output of the S-box.

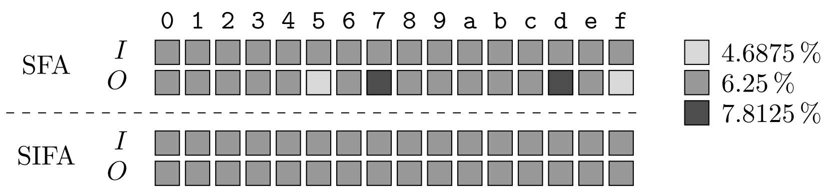


Figure 3: Distribution of input I and output O for faulted Sbox13 with instruction skip.

As we can see in Figure 3, in the unfiltered case, we see a clearly non-uniform distribution, which can be possibly exploited by an SFA. However, we observe a uniform distribution in the SIFA case. So does this mean this S-box is secure against SIFA? Let us consider the distributions we obtain when setting one share x_0 at the input of the first AND permanently to 0. The corresponding distributions we get in this experiment are shown in Figure 4.

As we can see in Figure 4, the situation for this fault changes, so that now, the ineffective faults can be exploited, whereas the distribution without filtering cannot be exploited. Until now, we have exploited the sequential sharing of instructions, especially that we can change the distribution of an AND gate. So one might wonder what happens if an S-box is directly shared, so that the output shares are uniformly distributed.

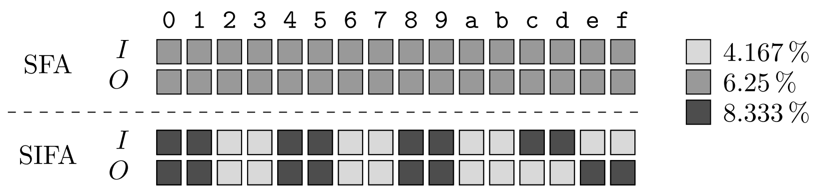


Figure 4: Distribution of I, O for Sbox13 setting input share x_0 of the first AND to zero.

To explore the case of directly shared S-boxes, let us have a look at the uniform 4-share threshold implementation of the Keccak S-box proposed by Bilgin et al. [BDN+13]. Here, A_i, B_i, C_i, and D_i represent the 4 shares of bit i. For the bits $i = 0, 1, 2, 4$, the sharing is calculated by:

$$\begin{aligned}
A'_i &\leftarrow B_i \oplus B_{i+2} \oplus ((B_{i+1} \oplus C_{i+1} \oplus D_{i+1})(B_{i+2} \oplus C_{i+2} \oplus D_{i+2}))\\
B'_i &\leftarrow C_i \oplus C_{i+2} \oplus (A_{i+1}(C_{i+2} \oplus D_{i+2}) \oplus A_{i+2}(C_{i+1} \oplus D_{i+1}) \oplus A_{i+1}A_{i+2})\\
C'_i &\leftarrow D_i \oplus D_{i+2} \oplus (A_{i+1}B_{i+2} \oplus A_{i+2}B_{i+1})\\
D'_i &\leftarrow A_i \oplus A_{i+2}
\end{aligned}$$

Bit 3 is calculated by:

$$\begin{aligned}
A'_3 &\leftarrow B_3 \oplus B_0 \oplus C_0 \oplus D_0 \oplus ((B_4 \oplus C_4 \oplus D_4)(B_0 \oplus C_0 \oplus D_0))\\
B'_3 &\leftarrow C_3 \oplus A_0 \oplus (A_4(C_0 \oplus D_0) \oplus A_0(C_4 \oplus D_4) \oplus A_0A_4)\\
C'_3 &\leftarrow D_3 \oplus (A_4B_0 \oplus A_0B_4)\\
D'_3 &\leftarrow A_3
\end{aligned}$$

Now, in our simple experiment, let us consider that bits 0 to 3 are calculated correctly and an attacker changes the value of one input share A_0 always to 0 before the calculation of the 4 shares for output bit $i = 4$. Then an attacker is able to mount a SIFA as indicated by the distributions of Figure 5. This leads again to an exploitable bias of the distribution of unmasked values at the output of the S-box and the attacker can mount a SIFA attack.

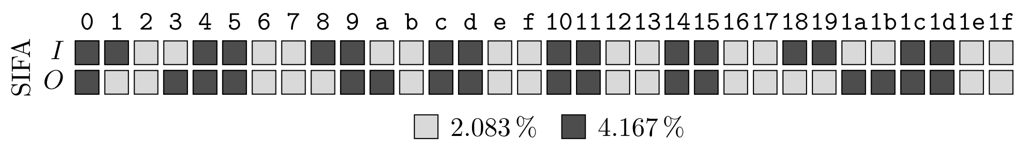


Figure 5: Distribution of I, O for faulted 4-shared Keccak S-box.

The aim of this section was to give easy to follow and reproducible examples to induce a bias in the unshared variable of masked S-boxes by just faulting one share of the S-box. We want to mention that the given ways and locations of introducing the faults are not exhaustive and that there are many more location and various types of faults that make an attack successful. In the next section, we want to give a closer view on the problem of protecting an S-box against these attacks and get more insight into the effect allowing statistical attacks with the help of a 3-bit S-box as an example.

3.4 A Closer Look

In general, fault attacks exploit knowledge about intermediate values of cryptographic primitives, which are gained by disturbing the computation or intermediate values directly by the means of faults. In the case of DFA [BS97], this knowledge is that in certain intermediate bits or bytes a difference is induced, while others remain fault-free. In the case of SFA and SIFA, this is knowledge that the distribution of certain intermediate values is changed from a uniform to a non-uniform distribution. This allows an attacker to guess parts of the round key and calculate backwards to these influenced intermediate values from collected ciphertexts. If a key guess is wrong, an attacker expects to see a distribution of intermediate values, which is closer to uniform compared to the guess of the right key.

Getting such a non-uniform distribution of intermediate values can be achieved in many ways. For ciphers following the SPN structure, where every S-box and the linear layer is a bijective function (permutation), non-uniform distribution of intermediate value can be achieved, for instance, by disturbing the computation of a single S-box, so that this S-box does not act as a permutation anymore. While such a behavior can be expected from an unmasked S-box in the case of a fault induction, it is quite counterintuitive for masked implementations. Thus, we will first discuss the unmasked case to get more insight in which cases SFA and SIFA will work. Then we will have a closer look on masked S-box implementations.

3.4.1 Influencing Unmasked S-boxes

First, we will explore the case of SFA. We consider a bijective S-box, as illustrated in Figure 6. For the sake of simplicity, let us assume that the S-box is implemented in a bit-sliced manner (as a sequence of instructions), or as a Boolean circuit in hardware. Let us further assume that an attacker influences the correct computation by the means of faults. By faulting this computation, it is very likely that the faulted S-box does not behave as a bijection, but rather becomes a general function, which is non-surjective. This leads to a non-uniform distribution of intermediate values, which can be exploited in an SFA. In the case of a SIFA, transitions where a fault induction has an effect and causes a change in the output of the S-box are filtered. This filtering can happen by using the fact that detection-based countermeasures do not deliver an output in the case the fault has an effect, or by comparing the obtained output with an output where no fault was injected. So if we apply a SIFA, we observe a function, where a reduced set of transitions remain compared to the unfaulted S-box.

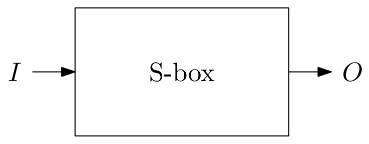
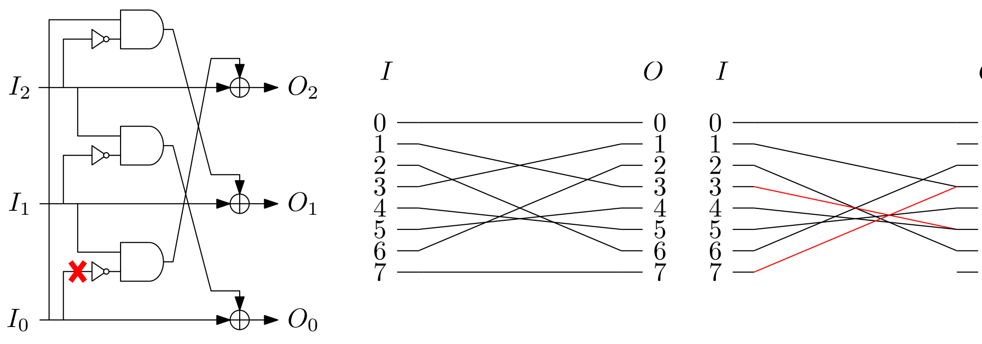


Figure 6: An unmasked S-box.

To get more insight into the behavior of a faulted S-box, we will use the 3×3 S-box χ based on Daemen's χ-layer [DGV94, Dae95] as an illustrative example. Figure 7a shows the 3-bit S-box χ, where the red cross represents a fault that sets the input of the subsequent inversion to zero and hence the input of the AND to 1. Please note that this is only one example of many how to fault an S-box in order to apply an SFA or SIFA.

The fault depicted in Figure 7a changes the behavior of the χ S-box, which is not bijective anymore. Figure 7c shows the transition graph for the faulted S-box. The fault just changes the transitions depicted in red.

In our case, the transitions mapping from $3 \rightarrow 1$ and $7 \rightarrow 7$ in the unfaulted S-box (Figure 7b), now map from $3 \rightarrow 5$ and $7 \rightarrow 3$ (considering I_0 and O_0 being the LSB). This means that the output values 1 and 7 never appear, but 3 and 5 twice, leading to a non-surjective behavior. In the case of a SIFA, the red edges represent fault inductions



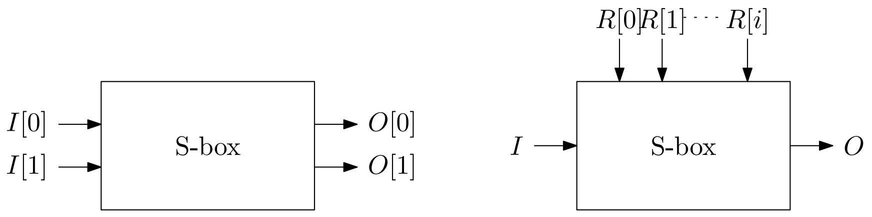
(a) Circuit with single fault (b) Unfaulted mapping (c) Faulted mapping

Figure 7: 3-bit S-box χ with a single fault.

that show an effect on the output. When applying a SIFA, those edges are filtered (e.g by detection-based countermeasure) and just the black edges remain. Hence, when performing a SIFA, the correct transitions $3 \rightarrow 1$ and $7 \rightarrow 7$ do not appear and an attacker can observe an exploitable non-uniform distribution of intermediates.

3.4.2 Influencing Masked S-boxes

Now let us have a look at a shared bijective S-box. For instance, consider the masked S-box shown in Figure 8a that takes two shares as input and returns two output shares. Here, an attacker has also the goal to influence the distribution or transitions of the unshared values $I = I[0] \oplus I[1]$ and $O = O[0] \oplus O[1]$, but does not care about the concrete values the shares take.



(a) Implementation with 2 shares (b) Functional equivalent view

Figure 8: A masked S-box.

For this reason, it is easier to work with the functional equivalent model shown in Figure 8b. In this model, we see a masked S-box as a function, which takes an unshared input I and some randomness $R[i]$ and produces an unshared output O. Here, some of the random bits symbolize all values a shared input can take in a real implementation, e.g., $I[0] = I \oplus R[0]$, and $I[1] = R[0]$, while others represent randomness used in the masked implementation. Now, we can see masking as a very special and complicated function, which takes as input I, $R[0]$, $R[1]$,… $R[i]$ and produces an output O, so that the same I always leads to the same O for all possible choices of $R[0]$, $R[1]$,… $R[i]$. It seems very unlikely that a shared S-box behaves in the same manner in the presence of faults.

To apply SFA successfully, we need that not all values for O (iterating over all values of I) appear the exact same number of times when counting for all possible assignments of $R[0]$, $R[1]$, and $R[2]$. This prerequisite is very likely to hold considering an attacker that can tamper with the intermediate calculation performed, even when restricting the attacker to just manipulate one share used in an intermediate calculation. Similarly, to apply SIFA successfully, we need a fault such that among the ineffectively faulted computations, not all values for I or O appear the exact same number of times over all values of R_0, R_1,… R_i. This condition is similarly very likely to happen in practice when introducing just single faults, as we will show with our practical experiments in Section 4.

As an example, consider again the 3-bit Keccak χ S-box, now with the following masked implementation:

$$I'_i[0] \leftarrow (I_{i+1}[0] \oplus 1)I_{i+2}[0] \oplus ((I_{i+1}[0] \oplus 1)I_{i+2}[1] \oplus I_{i+0}[0])$$
$$I'_i[1] \leftarrow I_{i+1}[1]I_{i+2}[0] \oplus (I_{i+1}[1]I_{i+2}[1] \oplus I_{i+0}[1])$$

This masked S-box just serves us as an illustrative example of the effect of faults on an S-box, hence, we do not care about potential positioning of registers or additional randomness at the output for re-sharing. Figure 9 shows the equivalent circuit of the S-box, where again we just set a single value to 0. The result of this fault is that the value of $O_2[0]$ equals $I_2[0]$. Everything else is calculated correctly.

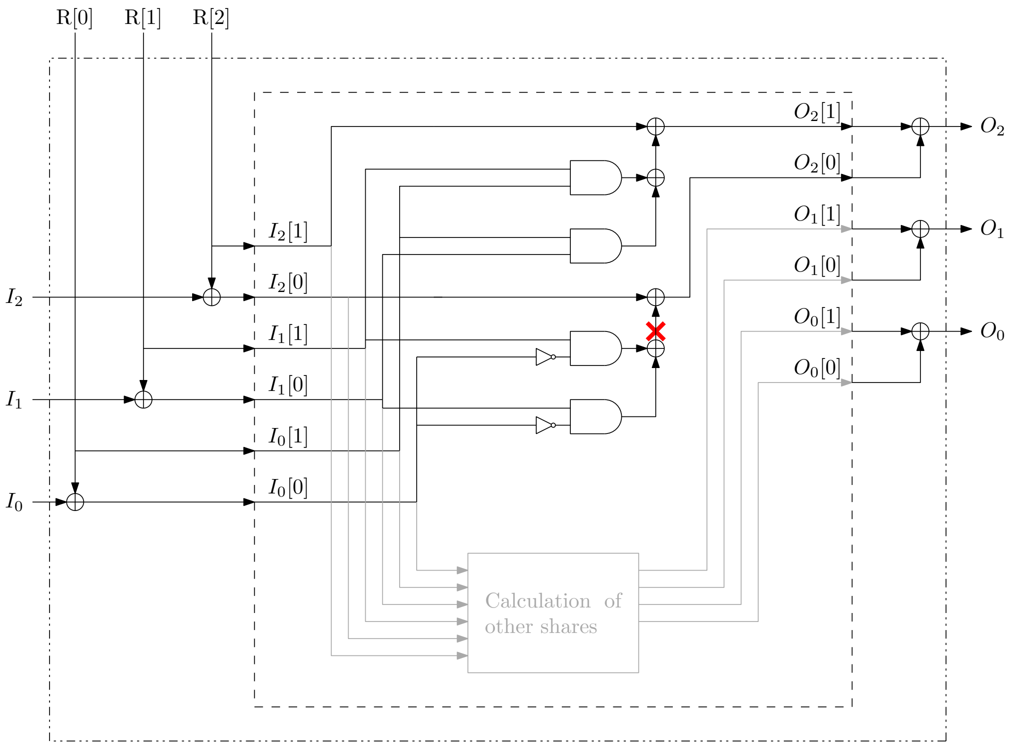


Figure 9: Single fault on masked 3-bit χ S-box.

For our example depicted in Figure 9, we list all possible assignments of I_0, I_1, I_2, $R[0]$, $R[1]$, and $R[2]$ in Table 1. The entries marked in red in Table 1 are entries where the fault pictured in Figure 9 has an effect. Due to the more complex calculations that happen for masked S-boxes, we get more complex relation between masks and actual values of bits. For instance, the transition $2 \rightarrow 6$ is only valid if $R[0] = 1$ and wrong $(2 \rightarrow 2)$ if $R[0] = 0$.

Again, we can represent all possible transitions from inputs I, to the outputs O in a graph shown in Figure 10 (in a similar way as in Figure 7c). However, due to the 2^3 possible ways of masking our input values, each transition from input to output would happen 8 times for an *unfaulted* masked 3-bit S-box χ. In the faulted case, this condition does not hold anymore as shown in Table 1. Hence, we have additional transitions shown in red in Figure 10. These “wrong” transitions also reduce the number of times the “correct” transition happens.

Table 1: Transitions of faulted masked 3-bit S-box χ.

I_2	$R[2]$	I_1	$R[1]$	I_0	$R[0]$	O_2	O_1	O_0	I_2	$R[2]$	I_1	$R[1]$	I_0	$R[0]$	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1	0	1
0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	1	0	0
0	0	0	0	1	1	0	1	1	1	0	0	0	1	1	1	0	0
0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	1
0	0	0	1	0	1	0	0	0	1	0	0	1	0	1	1	0	1
0	0	0	1	1	0	0	1	1	1	0	0	1	1	0	1	0	0
0	0	0	1	1	1	0	1	1	1	0	0	1	1	1	1	0	0
0	0	1	0	0	0	0	1	0	1	0	1	0	0	0	1	1	0
0	0	1	0	0	1	1	1	0	1	0	1	0	0	1	0	1	0
0	0	1	0	1	0	0	0	1	1	0	1	0	1	0	1	1	1
0	0	1	0	1	1	1	0	1	1	0	1	0	1	1	0	1	1
0	0	1	1	0	0	0	1	0	1	0	1	1	0	0	1	1	0
0	0	1	1	0	1	1	1	0	1	0	1	1	0	1	0	1	0
0	0	1	1	1	0	0	0	1	1	0	1	1	1	0	1	1	1
0	0	1	1	1	1	1	0	1	1	0	1	1	1	1	0	1	1
0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1
0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	1	0	1
0	1	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0
0	1	0	0	1	1	0	1	1	1	1	0	0	1	1	1	0	0
0	1	0	1	0	0	0	0	0	1	1	0	1	0	0	1	0	1
0	1	0	1	0	1	0	0	0	1	1	0	1	0	1	1	0	1
0	1	0	1	1	0	0	1	1	1	1	0	1	1	0	1	0	0
0	1	0	1	1	1	0	1	1	1	1	0	1	1	1	1	0	0
0	1	1	0	0	0	0	1	0	1	1	1	0	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1	0	0	1	0	1	0
0	1	1	0	1	0	0	0	1	1	1	1	0	1	0	1	1	1
0	1	1	0	1	1	1	0	1	1	1	1	0	1	1	0	1	1
0	1	1	1	0	0	0	1	0	1	1	1	1	0	0	1	1	0
0	1	1	1	0	1	1	1	0	1	1	1	1	0	1	0	1	0
0	1	1	1	1	0	0	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	1

If we now count the number of transitions in Figure 10 that lead to a certain value O, we see that 12 transitions lead to values 3 and 5, whereas only 4 lead to 1 and 7. This means that an attacker faulting a device can apply an SFA, since the attacker can expect a non-uniform distribution of the value after the S-box for correct key guesses. If we apply a SIFA, the transitions marked in red in Figure 10, get filtered. As an effect, the transitions $2 \rightarrow 6$, $3 \rightarrow 1$, $5 \rightarrow 2$, and $7 \rightarrow 7$ appear with reduced frequency for uniformly distributed $R[0]$, $R[1]$, and $R[2]$. Again, this can be exploited in a key recovery attack.

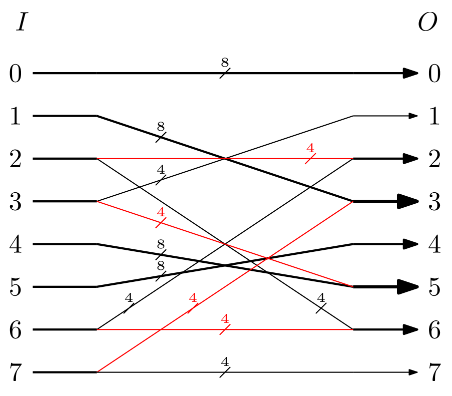


Figure 10: Transition graph of faulted masked 3-bit S-box χ.

4 Attack Evaluation

In this section, we demonstrate the applicability of (ineffective) biased faults for a publicly available higher-order masked AES software implementation. More concretely, we target the provable secure higher-order masked AES from Rivain et al. [RP10] on an ATXmega 128D4. The used implementation is from Coron et al. [Cor17]. Our experiments require multiple faulted encryptions but only one fault induction per encryption. The exact location as well as the actual effect of the induced fault does not need to be known by the attacker. Indeed, inducing a fault anywhere in the shared S-box in round 9 possibly leads to a situation as described in Subsection 3.4. The resulting joint non-uniform (biased) distribution over all shares of an intermediate variable can then be used to distinguish key candidates like in SFA and SIFA. Our practical fault attacks were performed by means of clock glitches.

The higher-order masked AES from Rivain et al. [RP10] consists of a generic dth-order masked S-box that is combined with a linear layer for the $d+1$ shares of the AES state. The target of our fault induction is the shared S-box implementation in round 9. Hence we briefly describe its implementation in the following.

4.1 Generic Higher-Order Masked S-box

The algebraic description of the 8-bit AES S-box consists of determining the multiplicative inverse of a number in $\mathbb{F}_{2^8} = \mathbb{F}_2[x]/(x^8+x^4+x^3+x+1)$, followed by an affine transformation over $\mathbb{F}_2^8$. While masking the affine transformation is trivial since it can be calculated separately for each share, the calculation of the masked multiplicative inverse requires more work. In the design of Rivain et al. the inversion is calculated via the power function $x \to x^{254}$ over $\mathbb{F}_{2^8}$ which is in return calculated via the square-and-multiply algorithm. The squaring operation in $\mathbb{F}_{2^8}$ is a linear function which leaves the masking of the field multiplication as the only non-trivial task. The used algorithm for dth-order masked field multiplication is based on the ISW scheme (cf. Algorithm 1). For a more detailed description we refer to the original paper [RP10].

4.2 Practical Results

In our practical evaluation we perform fault inductions on an ATXmega 128D4 via clock glitching. More precisely, we insert an additional fast clock cycle in between two ordinary clock cycles. The width of the induced clock cycle is chosen such that it is recognized by the microprocessor but too short to allow the correct execution of the current instruction. The target of our fault is the higher-order masked field multiplication operation (`SecMult`) that occurs multiple times (1st-order: 64 times, 10th-order: 2880 times) during the computation of the masked S-box in round 9. We neither require to fault one specific `SecMult` invocation nor to fault one specific instruction within any of the `SecMult` invocations. In fact, any fault that causes a joint non-uniform distribution over all shares (cf. Subsection 3.3) is sufficient for our attack. For this very reason, attacking unknown implementations might actually be easier if higher-order masking is used since the vulnerable masked S-box operations take up a higher percentage of the overall runtime.

The implementation of the higher-order masked AES by Coron et al. [Cor17] has a configurable masking order d. Our experiments were performed up to 10th-order masking, with or without temporal redundancy as an additional fault countermeasure. First, we present the results of attacking the higher-order masked AES without additional fault countermeasures using SFA. Our attack works equally well for any masking order d, hence we only state concrete results for $d = 10$. We then present the results of attacking the same implementation with temporal redundancy as an additional fault countermeasure using SIFA. Please note that our attack is not influenced by the degree of the temporal redundancy, since we only fault one of the redundant AES calculations.

SFA on 10th-order masked AES. For this attack we performed 50 000 faulted encryptions where a single fault was induced somewhere within the masked S-box in round 9. From the obtained faulty ciphertexts we can calculate backwards using partial key guesses (cf. Subsection 2.1) and observe the distribution of the AES state bytes after the S-box in round 9, as illustrated in Figure 11. In this concrete example a significant peak can be observed in the distribution of state byte 0, if the corresponding partial key guess is correct. Already after evaluating about 35 000 faulty ciphertexts the SEI of the observed distributions is highest for the correct partial key guess. Hence about 35 000 faulted encryptions are sufficient to recover 32 key bits. However, this number can be reduced to around 1 000 faulted encryptions if we apply SIFA, as we have to do anyway if the masked AES is additionally protected with temporal redundancy against fault attacks.

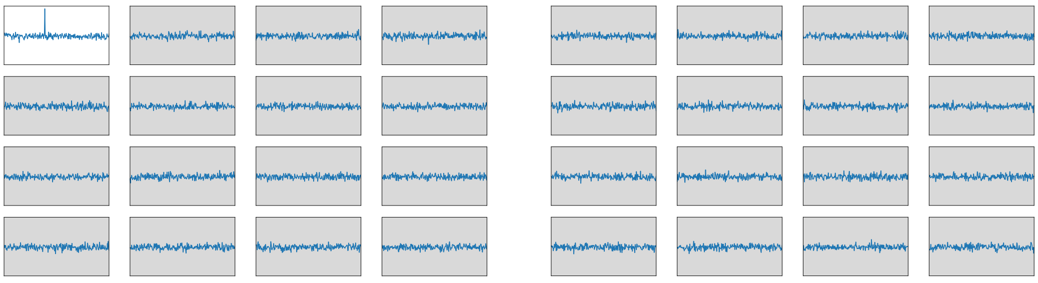
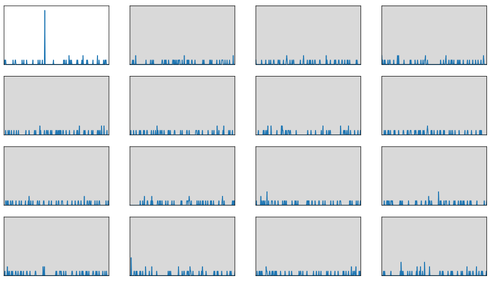

(a) Correct key guess. (b) An incorrect key guess.

Figure 11: **10th-order masked AES.** Distribution of AES state bytes after S-box in round 9. Calculated from 50 000 faulty ciphertexts, only about 35 000 are needed to recover 32 key bits.

SIFA on 10th-order masked AES with temporal redundancy. We then repeated the previous attack but additionally add temporal redundancy as a fault countermeasure to the higher-order masked AES implementation. In temporal redundancy the encryption is performed n-times and the computed ciphertext is only released if all individual computations match up. Such a countermeasure prevents the previous attack that relied on observing faulty ciphertexts and single fault inductions per encryption.

In order to attack such a protected implementation we need to change our attack strategy in that we now exploit statistical ineffective faults, such as in SIFA. Again, we induce single faults during the calculation of the masked S-box in round 9, but only in one out of n redundant encryptions. If the induced fault results in a faulty computation we do not observe any ciphertext because of the redundancy countermeasure. However, we do observe correct ciphertexts stemming from faulted encryptions where the induced fault was ineffective. This filtered set of correct ciphertexts can then again be used to perform key recovery.

In Figure 12, we show the distribution of the AES state bytes in round 9 when calculated from 48 collected correct ciphertexts. Similar to the previous attack, for a correct partial key guess, we can observe a strong biased distribution in one of the state bytes in round 9 and thus can perform key recovery (cf. Subsection 2.1). This time we only exploit correct ciphertexts and thus successfully circumvent the redundancy countermeasure while using the same assumptions as for the previous attack. Interestingly, the required number of faulted encryptions in this scenario is significantly lower compared to before. Already after collecting about 20 correct ciphertexts stemming from about 1 000 faulted encryptions the SEI of the observed distributions is highest for the correct partial key guess. In other words, for this concrete higher-order masked AES implementation, SIFA allows us to perform key recovery 35 times faster than SFA. This might seem counterintuitive but we need to recall that the effect of a fault is different between SFA, where we partially decrypt faulted ciphertexts and SIFA, where we only partially decrypt ciphertexts where the induced fault was ineffective (cf. Subsection 3.3). Hence, depending on the attacked implementation and the actual fault effect, either of the two techniques might work better, however the results for SIFA are more interesting in a practical setting where typically fault countermeasures are implemented.


(a) Correct key guess.

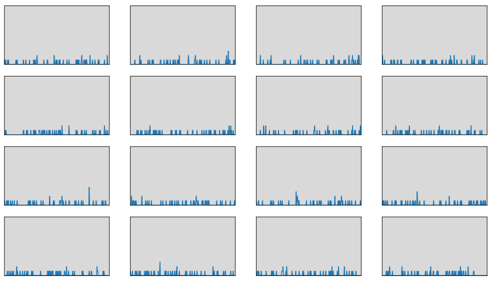

(b) An incorrect key guess.

Figure 12: **10th-order masked AES with temporal redundancy.** Distribution of AES state bytes after S-box in round 9. Calculated from 48 correct ciphertexts stemming from about 2 000 faulted encryptions, only about 20 correct ciphertexts (1 000 faulted encryptions) are needed to recover 32 key bits.

5 Discussion

5.1 On the Nature and Number of Faults

In Section 3, we explore the behavior of masked building blocks using deterministic stuck-at-0 faults, or instruction skips. The reason for this is to make the processes leading to a bias easier to digest for a reader. However, it is easy to see that making the fault probabilistic, e.g., going to a more realistic setup, where an instruction skip does not work all the time, or that a bit is only set with a certain probability to 0, just effects the bias an attacker observes and hence, the amount of ciphertext the attacker has to collect, but the attack still works (see [DEK+18] for more details).

Furthermore, clock glitches and setting values to zero are not an exhaustive list of effects that a fault could have in order to make the attack work. All in all, the only requirement we have on the fault is that it leads to a biased distribution of the real (unmasked) value at some place in the primitive. Hence, even for random faults and bitflips one could sometimes observe a biased distribution in the primitive in practice. In general, a fault can have a more complex nature than only the cases discussed in Section 3. For instance, in software implementations of masked ciphers, a large amount of instructions are LOAD instructions from memory since all shares might not fit in the registers. We have observed in practical experiments that faulting a LOAD can also lead to biased S-box distributions, but having a quite complex effect depending on previous calculations. However, the big benefit of SIFA is the fact that an attacker does not have to know or model the effect of a fault. For the attack to work, there just has to be a bias.

This fact also comes into play when dealing with the location of a fault. The examples for the location (e.g., skipped instruction) of the fault given in Section 3 just show one out of many different locations, where a fault targeting the S-box leads to a biased distribution that can be exploited in SFA or SIFA. How many such locations exists crucially depends on how the S-box is implemented. In a similar manner as for the effect of a fault, an attacker does not have to know, or to aim for just one specific instruction of location to fault. The only requirement for the attack to work in practice is that the faulted location leads to a bias. All these points make the attack to be executed in practice quite easily, even with a cheap setup using clock-glitches as demonstrated in Section 4.

The last point we want to discuss regarding faults is the number of faults per execution. We have opted for a single fault per execution, since inserting multiple faults per execution is usually considered to be harder. This is probably related to the prominence of fault attack techniques, where attacks requiring multiple faults per execution have high requirements of the exact location and effects of the induced faults. However, at least in the case of an SFA, we could not care less about the number of faults, since there are no strong requirements regarding effect and location. In fact, injecting multiple consecutive faults usually just leads to a lower number of necessary ciphertexts as the bias increases. For SIFA, the situation is slightly different; however, multiple faults injected in the computation of a single S-box have the potential to reduce the required number of ciphertexts, while being not necessarily harder to conduct in practice.

5.2 Countermeasures

In the following, we discuss the effectivity and practicability of well-known countermeasures against our attacks. Since most fault countermeasures prevent the SFA variants of our attack using a single fault per execution, we focus on the SIFA variants and show that it is not easy to prevent these attacks. In fact, some countermeasures (e.g., detection) even facilitate certain aspects of our SIFA variants in practice.

5.2.1 Self-Destruct

The most radical approach of destroying the device as soon as a fault is detected is a valid countermeasure against any fault attack. However, this technique has several downsides and limitations, including false positives and additional effort to reliably destroy a circuit.

A lot of cryptographic devices deployed in the field like smart cards and RFID tags have to function and operate under rather tough conditions. They typically have to deal with abrupt loss of power, for instance if a smart card is withdrawn from the terminal while working. Furthermore, they have to handle power spikes from electrostatic discharges, or electromagnetic fields. Hence, deciding between an active fault attack and interference due to normal usage is not a trivial task and would potentially lead to detection of a huge amount of false positives that render such an approach useless for a wide range of applications.

5.2.2 Detection and Correction

We classify countermeasures as detection-based if they use some kind of redundancy to detect a fault and do not return a ciphertext in this case. As demonstrated in [DEK+18], such a countermeasure does not effect SIFA at all, regardless of the degree of redundancy.

A different approach to make use of redundancy is to correct the effect of a fault, for instance using error-correcting codes or simple majority voting. However, correction-based countermeasures usually can be reduced to the detection-based case using additional faults. How hard this is and which requirements this might have on the precision of the fault crucially depends on the implementation of the countermeasure. As an example let us assume a simple majority voting between the result of 3 block cipher calls. To do this, the 3 block cipher calls take the same inputs and hence, perform redundant computations. An attacker can now use an additional fault to just ensure that the computations performed on one redundant block cipher call are always incorrect. This usually does not require a precise fault. This reduces the majority voting of 3 block cipher calls to a construction, which essentially behaves as a detection-based countermeasure (or infection-like countermeasure if the majority voting happens at bit-level of the ciphertext). Then, an attacker can proceed with the same attacks as usual, using a second fault.

5.2.3 Infection

In [DEK+18], the application of SIFA on an infective countermeasure [TBM14] has been demonstrated. The employed dummy rounds in this countermeasure increase the needed number of faulted encryptions until the key can be recovered. However, when aiming to prevent SIFA, dummy rounds that do not infect the state in the case of a fault should provide even more protection. Hence, we explore this countermeasure next.

5.2.4 Hiding

The goal of hiding countermeasures is to reduce the attacker's knowledge of what is currently computed, and thus effectively decrease his precision when placing the fault. Examples include adding dummy rounds randomly between the relevant rounds, or shuffling the order of execution, for example the order in which the 16 AES S-boxes per round are executed. In the following, we analyze the case of dummy AES rounds in more detail, and show that the noise introduced this way quadratically increases the necessary number of faulty encryptions for the analysis.

Dummy Rounds. We consider a protected AES implementation and make the following assumptions for our model:

- The attacker needs to fault round 9 out of 10 (identical) AES rounds.
- The protected implementation executes 10 real AES rounds and $(k-1)\cdot 10$ ineffective dummy rounds in a uniformly random ordering, labeled $1,\dots,R$ with $R = 10k$.
- The attacker targets round $R-t$. Three outcomes are possible:
    1. Hit: It is the real round 9 with probability σ, resulting in a distribution with ineffectivity rate π_{fault} and ineffective distribution p_{fault}.
    2. Miss: It is a dummy round with ineffectivity rate $\pi_{\mathsf{dummy}} = 1$ and uniform ineffective distribution θ
    3. Miss: It is a real round, but not round 9, with ineffectivity rate π_{fault} and uniform ineffective distribution θ. For simplicity, we assume an ineffectivity rate of $\pi_{\mathsf{dummy}} = 1$, so this case can be merged with item 2.

With these assumptions, the success probability that round $R-t$ of R is a hit (signal) is

$$\sigma_{\mathsf{fault}}[R,t] = \mathbb{P}\big[\text{Hit in round } R-t \text{ of } R\big] = \frac{t\cdot\binom{R-t-1}{8}}{\binom{R}{10}} = \frac{90\cdot t}{R\cdot(R-1)}\cdot\prod_{s=2}^{9}\left[1-\frac{t-1}{R-s}\right].$$

This parametrized function is plotted in Figure 13 and attains its maximum near $t = k = \frac{R}{10}$. The resulting function $\sigma_{\mathsf{fault}}[10k,k]$ for the optimized success probability is also plotted in Figure 13 (dashed, with x-axis $t=k$), and can be approximated as

$$\begin{aligned}\sigma_{\mathsf{fault}}[10k,k] &= \mathbb{P}\big[\text{Hit in round } 9k \text{ of } 10k\big] = \frac{90k}{10k\cdot(10k-1)}\cdot\prod_{s=2}^{9}\left[1-\frac{k-1}{10k-s}\right] \\ &= \frac{1}{k}\cdot\prod_{s=1}^{9}\frac{9k-s+1}{10k-s} \xrightarrow{k\to\infty} \frac{1}{k}\cdot\left(\frac{9}{10}\right)^{9} \approx \frac{1}{k}\cdot 0.387 \qquad \text{for large } k.\end{aligned}$$

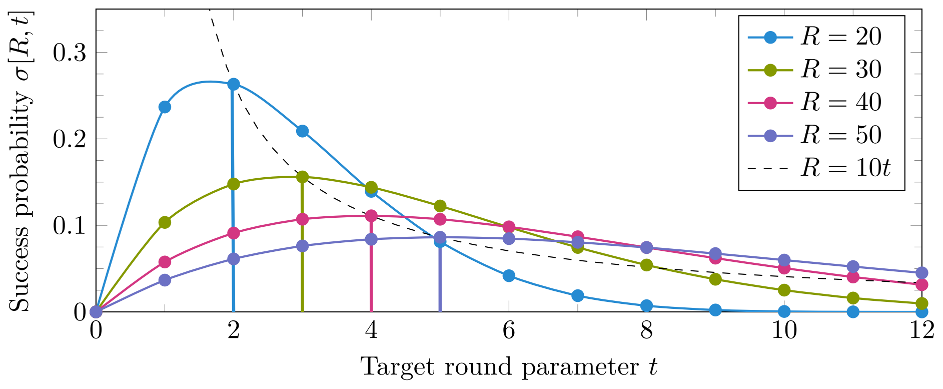


Figure 13: Success probability $\sigma[R,t]$ when targeting round $R-t$, for different $R = 10k$. For the choice $t = k$ (dashed line), $\sigma[10k,k] \approx \frac{0.387}{k}$ for $k \gg 2$.

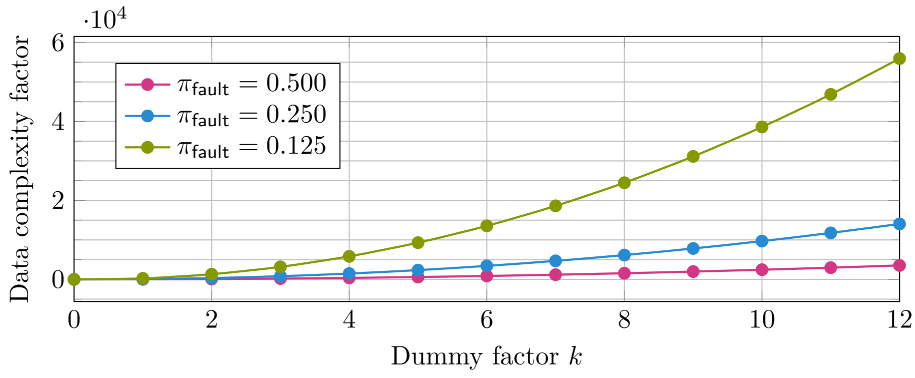


Figure 14: Increasing data complexity with dummy factor k, for different π_{fault}.

A SIFA attacker samples the resulting distribution p_{total} among the ineffective faults, with a total ineffectivity rate of π_{total} and a signal of σ_{total}:

$$\begin{aligned}\pi_{\mathsf{total}} &= \sigma_{\mathsf{fault}} \cdot \pi_{\mathsf{fault}} + (1 - \sigma_{\mathsf{fault}}) \cdot 1 = 1 - (1 - \pi_{\mathsf{fault}}) \cdot \sigma_{\mathsf{fault}} \\ \sigma_{\mathsf{total}} &= \frac{\sigma_{\mathsf{fault}} \cdot \pi_{\mathsf{fault}}}{\pi_{\mathsf{total}}} \\ p_{\mathsf{total}}(x) &= \sigma_{\mathsf{total}} \cdot p_{\mathsf{fault}}(x) + (1 - \sigma_{\mathsf{total}}) \cdot \theta(x)\end{aligned}$$

The necessary sample size to distinguish p_{total} is inverse proportional to the capacity

$$C(p_{\mathsf{total}}) = \sigma_{\mathsf{total}}^2 \cdot C(p_{\mathsf{fault}}) \,,$$

which corresponds to a data complexity proportional to $(\pi_{\mathsf{total}} \cdot \sigma_{\mathsf{total}}^2 \cdot C(p_{\mathsf{fault}}))^{-1}$.

Thus, for a fixed fault setup with $p_{\mathsf{fault}}, \pi_{\mathsf{fault}}$, increasing the dummy factor k increases the data complexity of the attack quadratically (Figure 14):

$$(\pi_{\mathsf{total}} \cdot \sigma_{\mathsf{total}}^2)^{-1} = \frac{1 - (1 - \pi_{\mathsf{fault}}) \cdot \sigma_{\mathsf{fault}}}{\sigma_{\mathsf{fault}}^2 \cdot \pi_{\mathsf{fault}}^2} \approx k^2 \cdot \frac{1}{(0.387 \cdot \pi_{\mathsf{fault}})^2} - k \cdot \frac{1 - \pi_{\mathsf{fault}}}{0.387 \cdot \pi_{\mathsf{fault}}^2} \,.$$

Shuffling. Similar to dummy rounds, shuffling operations reduces the attacker's precision and success probability in hitting the right S-box and thus induces noise in the distribution. However, in the case of SIFA, there is an important difference due to the ineffectivity rate in case of misses, which we assume is the same as in case of hits. For this reason, the data complexity will also grow quadratically in the number of shuffled operations in the relevant scope (e.g., 16 S-boxes), but only linearly instead of quadratically in the inverse ineffectivity rate $\pi_{\mathsf{fault}}^{-1}$.

5.2.5 Reducing the Data Complexity

For our attacks to work, we usually need several faulty encryptions per key to retrieve it. Therefore, methods that restrict the usage of the key and hence, put a limit on the data complexity can be a viable strategy for providing protection against this type of attack. Existing re-keying strategies can be roughly split into two groups, one where the used key is derived via a re-keying functions from a static master key [MSGR10, DEM$^+$17, DKM$^+$15, BPPS17] and the other group being methods where a secret internal state is maintained and constantly updated. In the first group, the problem of protection against the attack is basically shifted to the re-keying function and has to be solved there.

5.3 Choice of the Target and Attack Setup

It is important to note that we have not chosen the S-boxes in Section 3, or AES in Section 4 as targets of our attacks because we have found them to be weaker than others. In fact, we have chosen them because a lot of masked or threshold implementations for them are publicly available.

Furthermore, we have done the practical experiments using clock glitches, because the equipment is cheap and we do not have easy access to other equipment at the moment. Obviously, the attack is not limited to a specific fault injection method. Quite the contrary, we expect other methods of inserting faults like lasers, needle probes, etc to be far superior compared to our cheap setup using clock glitches [DEK$^+$16].

5.4 Further Applications

For the sake of simplicity, we have put the main focus of our paper on the application of statistical fault attacks for (masked) design strategies using bijective S-boxes. Doing so allows us to follow the simpler to digest and for the human reader to gasp narrative that the attack can be applied if the distribution is shifted away from uniform. However, this does not mean that the attack only applies on primitives using bijective S-boxes. As discussed in Subsection 3.4, a fault attack may influence only some transitions in the transition graph, while leaving other intact. In the case of a SIFA, an attacker can observe and exploit the “filtered” graph, where most likely only the intact transitions remain. Note that in the masked case, there is more than one transition from one input to one output value, due to masks. Hence, an attacker can potentially exploit all cases, where this “filtered” transition graph shows a differently distributed occurrence of input and output transitions.

As another narrative restriction, we have restricted our focus on block ciphers. One potential countermeasure one could come up against our attack is the use of a PRF like the AES-PRF [MN17]. Such a PRF prevents an attacker seeing ciphertexts from decrypting backwards under guessing the key. So the attack does not work anymore right? No. In fact a SIFA still remains possible targeting the input of the AES-PRF since here, a known input is most likely to be processed. In general, our presented attacks are almost always applicable whenever some known input is mixed with a secret, which covers most stateless symmetric cryptographic primitives. However, it is an interesting future research topic how good such attacks work.

6 Conclusion

This paper demonstrates that SIFA is a very powerful attack. We show that state-of-the-art countermeasures against implementation attacks: redundancy against faults and masking against side-channels are not as effective against SIFA as expected. In particular, SIFA is still possible using just a single fault per execution, contradicting the common folklore that masking plus a detection-based countermeasure provides sufficient protection against fault attacks. We show the practical feasibility of the attack by using a cheap clock glitch setup attacking a 10th-order masked AES software implementation with arbitrary temporal redundancy on a standard 8-bit microcontroller without specific security features. Even with such a cheap setup, we are able to recover 32 bits of the key after collecting 20 ciphertext where the fault is ineffective, needing approximately a total of 1000 encryptions where a single fault induction is performed.

References

[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 *Smart Card Research and Advanced Applications – CARDIS 2013*, volume 8419 of *LNCS*, pages 187–199. Springer, 2013.

[BPPS17] Francesco Berti, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. On leakage-resilient authenticated encryption with decryption leakages. *IACR Transactions on Symmetric Cryptology*, 2017(3):271–293, 2017.

[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 *LNCS*, pages 513–525. Springer, 1997.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *LNCS*, pages 181–194. Springer, 2007.

[Cor17] Jean-Sébastien Coron. Higher order countermeasures for AES and DES, 2017. https://github.com/coron/htable#higher-order-countermeasures-for-aes-and-des.

[Dae95] Joan Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis.* PhD thesis, KU Leuven, 1995. http://jda.noekeon.org/.

[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*, volume 10031 of *LNCS*, pages 369–395, 2016.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. Exploiting ineffective fault inductions on symmetric cryptography. Cryptology ePrint Archive, Report 2018/71, 2018.

[DEM+17] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. ISAP – towards side-channel secure authenticated encryption. *IACR Transactions on Symmetric Cryptology*, 2017(1):80–105, 2017.

[DGV94] Joan Daemen, René Govaerts, and Joos Vandewalle. An efficient nonlinear shift-invariant transformation. In B. Macq, editor, *Information Theory in the Benelux*, pages 108–115. Werkgemeenschap voor Informatie- en Communicatietheorie, 1994.

[DKM$^+$15] Christoph Dobraunig, François Koeune, Stefan Mangard, Florian Mendel, and François-Xavier Standaert. Towards fresh and hybrid re-keying schemes with beyond birthday security. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications – CARDIS 2015*, volume 9514 of *LNCS*, pages 225–241. Springer, 2015.

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

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

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *Topics in Cryptology – CT-RSA 2017*, volume 10159 of *LNCS*, pages 95–112. Springer, 2017.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.

[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 *LNCS*, pages 388–397. Springer, 1999.

[MN17] Bart Mennink and Samuel Neves. Optimal PRFs from blockcipher designs. *IACR Transactions on Symmetric Cryptology*, 2017(3):228–252, 2017.

[MSGR10] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In Daniel J. Bernstein and Tanja Lange, editors, *Progress in Cryptology – AFRICACRYPT 2010*, volume 6055 of *LNCS*, pages 279–296. Springer, 2010.

[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, *Information and Communications Security – ICICS 2006*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security – E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9215 of *LNCS*, pages 764–783. Springer, 2015.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.

[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*, volume 9815 of *LNCS*, pages 302–332. Springer, 2016.

[TBM14] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Destroying fault invariant with randomization – A countermeasure for AES against differential fault attacks. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *LNCS*, pages 93–111. Springer, 2014.

[UDCI+11] Markus Ullrich, Christophe De Cannière, Sebastiaan Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. Finding optimal bitsliced implementations of 4×4-bit S-boxes. In *ECRYPT Symmetric Key Encryption Workshop – SKEW 2011*, pages 16–17, 2011.