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, Graz, Austria
robert.primas@iaik.tugraz.at
[2] Infineon Technologies AG, Neubiberg, Germany

Abstract. Implementation attacks like side-channel and fault attacks are a threat to deployed devices especially if an attacker has physical access. As a consequence, devices like smart cards and IoT devices usually provide countermeasures against implementation attacks, such as masking against side-channel attacks and detection-based countermeasures like temporal or spacial 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. These observations show that the combination of masking plus error detection alone may not provide sufficient protection against implementation attacks.

Keywords: Implementation attack · Fault attack · SFA · SIFA

1 Introduction

Fault attacks and passive side-channel attacks, like power [17] or EM analysis [21], are very powerful attacks on implementations of cryptographic algorithms. Therefore, devices like smart cards and IoT devices implement corresponding countermeasures, especially if they are potentially physically accessible by an attacker. In the case of symmetric cryptography, the typical approach to protecting 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 [15,22,23] providing protection up to a given protection order.

The list of authors is in alphabetical order (https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf).

T. Peyrin and S. Galbraith (Eds.): ASIACRYPT 2018, LNCS 11273, pp. 315–342, 2018.
https://doi.org/10.1007/978-3-030-03329-3_11

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 to detect errors. The basic idea is to compute critical operations multiple times and to release the output only if all redundant computations match. There are also works that directly combine masking and redundant encoding techniques for error detection [24]. The standard reasoning when combining masking and error-detection mechanisms is that their effects add up. For example, assume an implementation of AES protected by a masking scheme of protection order d and all encryption/decryption operations are always computed d times, compared, and the output is only released if the outputs of all d operations match. In this case, the typical assumption is that this implementation is secured against up to d fault inductions in one execution/encryption, because this is detected by the redundant computations, as well as secured 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 to reveal the key. The most prominent attacks of this type are Differential Fault Attacks (DFA) [2] and Statistical Fault Attacks (SFA) [13]. However, some variants of fault attacks are based on a different approach. Ineffective fault attacks (IFA) [5] and Statistical Ineffective Fault Attacks (SIFA) [10] 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 for different inputs 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 a redundant execution of the cipher, 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 which is needed to apply SIFA or IFA.

Our Contribution. So far, implementations combing masking and error-detection schemes were typically thought to be secure against attacks exploiting single ineffective faults due to masking, as discussed for example by Clavier for IFA [5]. It was typically assumed that all shares representing an intermediate value would need to be faulted for exploiting 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 and hence a suitable choice against implementations with countermeasures against both power analysis and fault attacks. Additionally, we show that SIFA is not restricted to

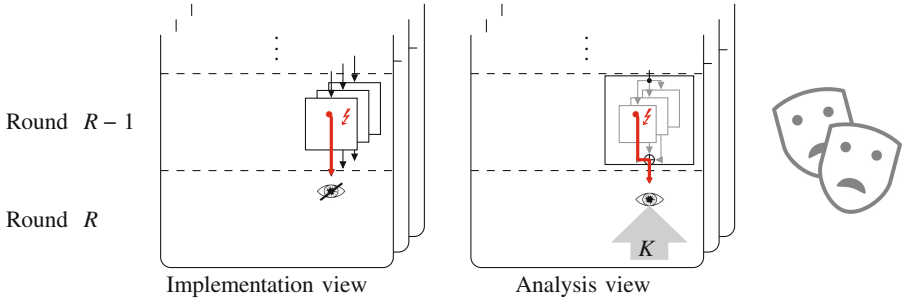


Fig. 1. Biased fault attacks on masked, redundant implementations: high-level view.

biased faults. Instead, any type of fault might be suitable to mount this kind of attack. In order to better explain why, we introduce a change of perspective by separating the fault location and the logical attack location. To back up our claims we provide a broad evaluation based on multiple (masked) S-boxes and two real AES implementations.

More concretely, we demonstrate that faulting a single share during the computation of an S-box is often sufficient to induce a bias in an unshared intermediate value, which can then be exploited with a statistical analysis based on SIFA. 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 (Fig. 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 like dummy rounds. 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 on block cipher level running on a standard 8-bit microcontroller.

2 Background on Statistical Faults

2.1 Statistical Fault Attacks

Statistical Fault Attacks (SFA), originally proposed by Fuhr et al. [13], present a way to recover an AES key with quite relaxed assumptions 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 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:

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

Note that 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 [10] 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 or spacial 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) [10] 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 induced faults can lead to a non-uniform distribution of intermediate values for the cases where the fault has been ineffective, which in turn 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, it has been shown in [10] that SIFA is not only applicable against redundancy countermeasures, but also against infective countermeasures [27]. Although in the latter case more faulted encryptions are necessary, the presented attacks are still efficient.

Consider an AES with simple temporal redundancy and an attacker that is able to fault a certain byte in round 9 such that the faulted value follows some non-uniform distribution, which is not known to the attacker. 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, typically also show some non-uniform, biased distribution in the faulted byte, related to the non-uniform distribution after fault induction. Now, the attacker can perform the same key recovery attack 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 important when SIFA is performed against masked implementations (cf. Sect. 3). Here it is practically 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 a 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, to better understand the underlying principles exploited in our attack, it is more useful to see the faults as changes to the actual function that is computed.

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 some S-boxes. Finally, we take a closer look at the internal activities in a faulted masked S-box that allow an SFA and SIFA and argue that it is probably 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 an easier understanding of 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 Sect. 5 and the SIFA analysis in [10].

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 resulting 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 $\mathrm{GF}(2^n)$.

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. [16]. 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 $\mathrm{GF}(2^n)$ – 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. [16]. In order to securely calculate $q = x \cdot y$, 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 to the output shares q_i (Algorithm 1).

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

Algorithm 1. Masked GF(2^n) multiplication according to Ishai et al. [16] (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: $r_{i,j} \xleftarrow{?} \mathrm{GF}(2^n)$
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: $q_i \leftarrow q_i \oplus t_{i,j}$

3.2 Faulting Masked AND Gates

We first note that the calculation of the AND $q = x \cdot y$ itself has a probability of 25% for q to be 1. An attacker therefore successfully biases the masked AND gate 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 Eq. 1, for instance the first AND calculating $x_0 y_0$ in q_0. The shared function then effectively calculates q ($= q_0 \oplus q_1$) to be $x_1 y_1 \oplus x_0 y_1 \oplus x_1 y_0$, which has a probability of 37.5% to be 1. If the attacker instead introduced a fault that skips the addition of the uniformly random bit $r_{0,1}$ in q_0, then the distribution of q would again be 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. [22] or in the Domain-Oriented Masking scheme by Gross et al. [14,15]. This same bias behavior results from the fact that these masked ANDs calculate the same terms $x_i y_j$. The masked ANDs only differ in the arrangement of $x_i y_j$ 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 [20]. 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. [20]:

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

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 Eq. 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 0 (83.3%) and 32 times a 1 (16.6%). This "filtered" distribution is the one an attacker can potentially exploit in the case of SIFA [10]. 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 SFA and SIFA 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 [28] shown in Fig. 2. We have implemented a masked implementation of this S-box in software by using a four-shared threshold implementation of AND (see Eq. 2), OR and XOR. We target exclusively the AND labeled with q, x, and y in Fig. 2.

For the first experiment, we assume an instruction skip that alters the execution of 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 Fig. 3a, we record the

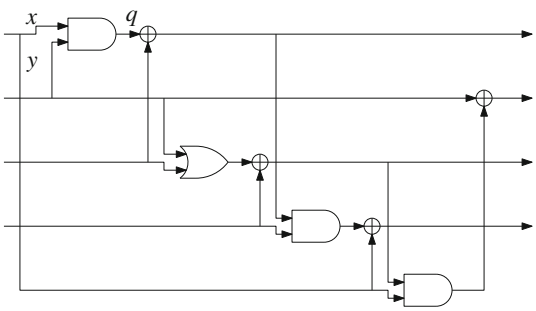
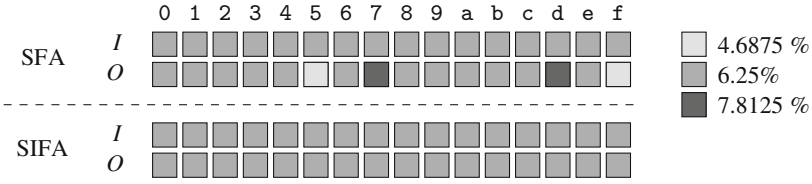
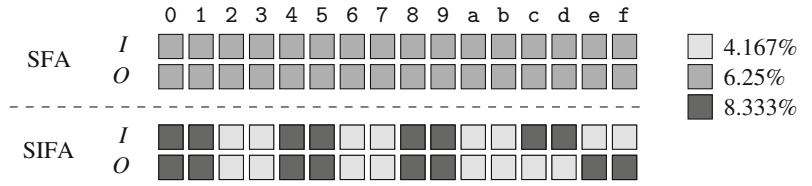


Fig. 2. Schematic of the 4×4 S-box: Sbox13 [28].



(a) Fault example 1: Skip first XOR instruction in share q_0 in (2).



(b) Fault example 2: Set input share x_0 of the first AND to zero in (2).

Fig. 3. Distribution of input I and output O for faulted 4×4 Sbox13.

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 [13], as well as the "filtered" distribution for ineffectively faulted S-box transitions, which can be exploited by SIFA [10]. 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.

As we can see in Fig. 3a, 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 3b.

As we can see in Fig. 3b, 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.

To explore the case of directly shared S-boxes, let us take a closer look at the uniform 4-share threshold implementation of the Keccak S-box proposed by Bilgin et al. [3]. Here, $A[i]$, $B[i]$, $C[i]$, and $D[i]$ represent the 4 shares of bit i with $i = 0, \ldots, 4$. While the bit $i = 3$ is calculated by:

$$A'[3] \leftarrow B[3] \oplus B[0] \oplus C[0] \oplus D[0] \oplus ((B[4] \oplus C[4] \oplus D[4])(B[0] \oplus C[0] \oplus D[0]))$$
$$B'[3] \leftarrow C[3] \oplus A[0] \oplus (A[4](C[0] \oplus D[0]) \oplus A[0](C[4] \oplus D[4]) \oplus A[0]A[4])$$

$$C'[3] \leftarrow D[3] \oplus (A[4]B[0] \oplus A[0]B[4])$$
$$D'[3] \leftarrow A[3]$$

the other bits $i = 0, 1, 2, 4$ are calculated by:

$$\begin{aligned}
A'[i] \leftarrow & B[i] \oplus B[i+2] \\
& \oplus \ ((B[i+1] \oplus C[i+1] \oplus D[i+1])(B[i+2] \oplus C[i+2] \oplus D[i+2])) \\
B'[i] \leftarrow & C_i \oplus C[i+2] \\
& \oplus \ (A[i+1](C[i+2] \oplus D[i+2]) \oplus A[i+2](C[i+1] \oplus D[i+1]) \oplus A[i+1]A[i+2]) \\
C'[i] \leftarrow & D[i] \oplus D[i+2] \oplus (A[i+1]B[i+2] \oplus A[i+2]B[i+1]) \\
D'[i] \leftarrow & A[i] \oplus A[i+2]
\end{aligned}$$

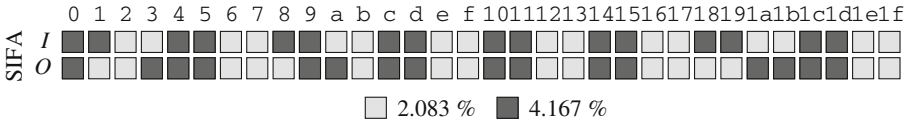


Fig. 4. Distribution of I, O for faulted 4-shared Keccak S-box.

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 SIFA as indicated by the distributions of Fig. 4. 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 SIFA.

The aim of this section was to give reproducible, easy-to-follow examples of inducing 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 locations and various types of faults that make an attack successful. In the next section, we 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 [2], 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 knowledge is 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, this seems quite counterintuitive for masked implementations at first glance. Thus, we will first discuss the unmasked case to get more insight in which cases SFA and SIFA will work. Then we will take a closer look on masked S-box implementations.

Influencing Unmasked S-Boxes. First, we will explore the case of SFA. We consider a bijective S-box, as illustrated in Fig. 5. 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 with SFA. In the case of 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 in SIFA, we observe a function where a reduced set of transitions remains compared to the unfaulted S-box.

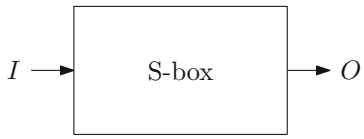
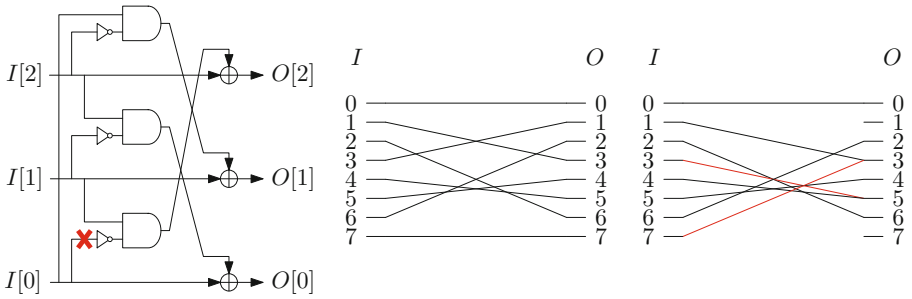


Fig. 5. An unmasked S-box.

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 [7,8] as an illustrative example. Figure 6a 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 gate to 1. Please note that this is only one example of many how to fault an S-box to apply an SFA or SIFA.



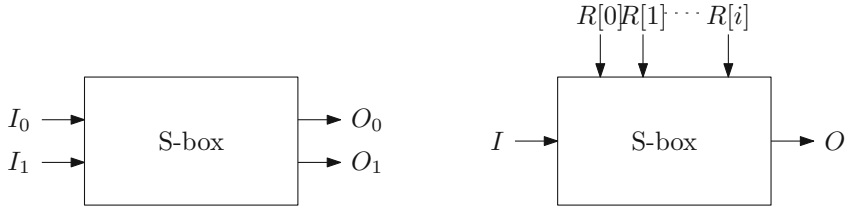
(a) Circuit with single fault (b) Unfaulted mapping (c) Faulted mapping

Fig. 6. 3-bit S-box χ with a single fault. (Color figure online)

The fault depicted in Fig. 6a changes the behavior of the χ S-box, which is not bijective anymore. Figure 6c 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 (Fig. 6b), 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 appear twice, leading to a non-surjective behavior. In the case of SIFA, the red edges represent fault inductions that show an effect on the output. When applying SIFA, those edges are filtered (e.g. by detection-based countermeasure) and just the black edges remain. Hence, when performing 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.

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 Fig. 7a that takes two shares as input and returns two output shares. Here, the goal for an attacker is also 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 of the shares.



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

Fig. 7. A masked S-box.

For this reason, it is easier to work with the functional equivalent model shown in Fig. 7b. 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 the inputs I, $R[0]$, $R[1], \ldots R[i]$ and produces an output O, so that the same I always leads to the same O for all possible choices of $R[0]$, $R[1], \ldots R[i]$. 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 $R[0]$, $R[1], \ldots R[i]$. This condition is similarly very likely to happen in practice when introducing just single faults, as we will show with our practical experiments in Sect. 4.

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

$$O_0[i] \leftarrow (I_0[i+1] \oplus 1) I_0[i+2] \oplus ((I_0[i+1] \oplus 1) I_1[i+2] \oplus I_0[i])$$
$$O_1[i] \leftarrow I_1[i+1] I_0[i+2] \oplus (I_1[i+1] I_1[i+2] \oplus I_1[i])$$

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 8 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 $O_0[2]$ equals $I_0[2]$. Everything else is calculated correctly.

For our example depicted in Fig. 8, 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 depicted in Fig. 8 has an effect. Due to the more complex calculations that happen for masked S-boxes, we get a 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 Fig. 9 (in a similar way as in Fig. 6c). However, due to the 2^3 possible ways of masking our input values, each transition from input to output will 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 Fig. 9. These "wrong" transitions also reduce the number of times the "correct" transition happens.

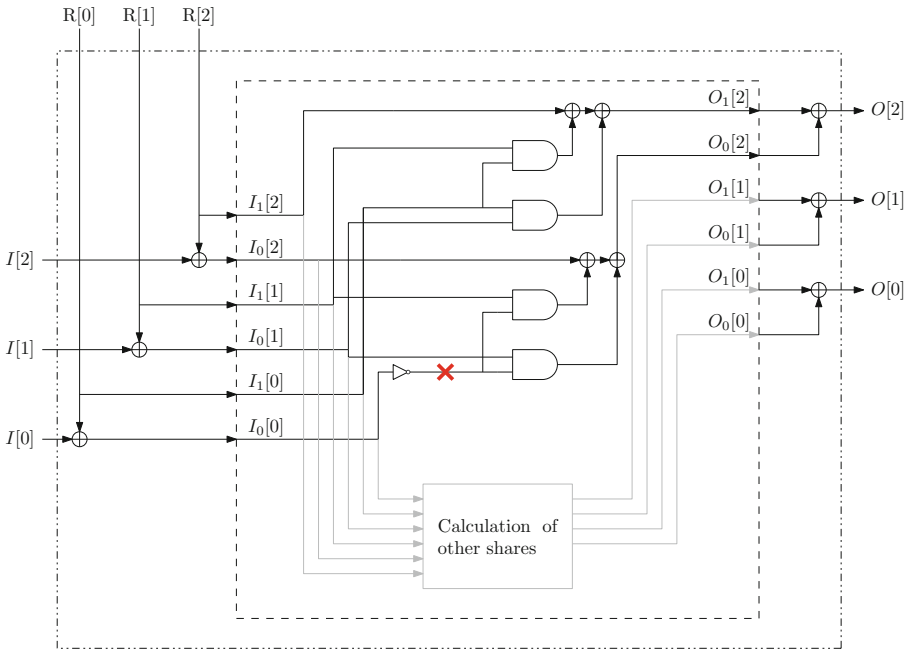


Fig. 8. Single fault on masked 3-bit χ S-box.

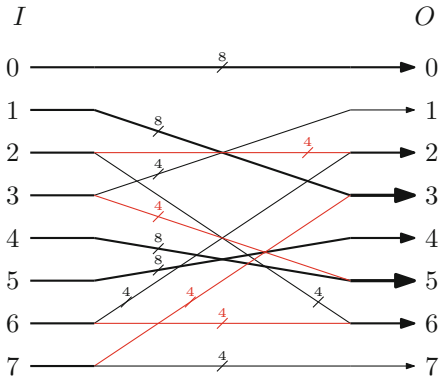


Fig. 9. Transition graph of faulted masked 3-bit S-box χ. (Color figure online)

If we now count the number of transitions in Fig. 9 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 SIFA, the transitions marked in red in Fig. 9, will be filtered. As an effect, the transitions $2 \rightarrow 6$, $3 \rightarrow 1$, $5 \rightarrow 2$, and

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

$7 \rightarrow 7$ appear with reduced frequency for uniformly distributed $R[0]$, $R[1]$, and $R[2]$. Again, this can be exploited in a key recovery attack.

4 Attack Evaluation

In this section, we demonstrate the applicability of Statistically Ineffective Fault Analysis (SIFA) for two very different publicly available masked AES implementations. First, we perform a practical attack evaluation for the provable secure, higher-order masked AES implementation from Rivain et al. [23] on a standard 8-bit microcontroller (ATXmega 128D4). We then present a comprehensive evaluation of simulated faults for the 32-bit, bitsliced, first-order masked AES implementation of Schwabe and Stoffelen [25]. Since both implementations do not originally have additional fault countermeasures in place, we added temporal redundancy, meaning that the block cipher is executed multiple times and the ciphertexts are compared. The number of redundant computations was set to two, since more redundancy does not affect the effectiveness of SIFA.

Our experiments require multiple faulted encryptions, but only one fault induction per encryption. The precise 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 likely leads to a situation that is similar as described in Sect. 3.4. The resulting joint non-uniform (biased) distribution over all shares of an intermediate variable can then be used to distinguish correct and wrong key candidates (cf. Sect. 2).

4.1 Practical Attack on AES from Rivain et al.

The higher-order masked AES from Rivain et al. [23] 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. First, we briefly describe the implementation of the masked S-box. Then we present the attack setup that was used for fault induction. The results of this practical evaluation are stated at the end of this section.

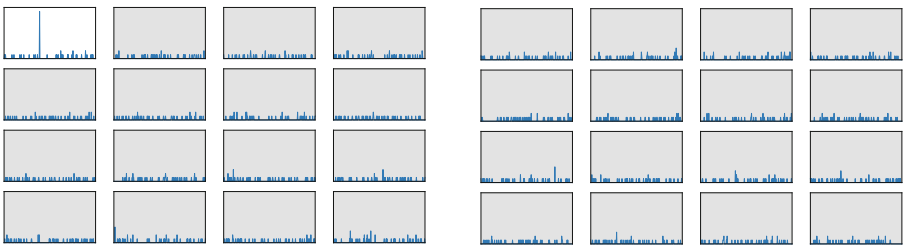
Generic Higher-Order Masked S-Box. The algebraic description of the 8-bit AES S-box consists of determining the multiplicative inverse of a number in $\mathbb{F}_{2^8} = \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$, followed by an affine transformation over $\mathbb{F}_2^8$. While masking the affine transformation 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 $x \rightarrow x^{254}$ 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 [23].

Attack Setup. In the 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 during the execution of one of the redundant encryptions. The width of the induced clock cycle is chosen such that it is recognized by the microprocessor but too short to allow a correct execution of the current instruction. The target of our fault is one of the higher-order masked field multiplication operations (`SecMult`) that occur 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. Sect. 3.3) is sufficient for our attack. For this reason, finding a suitable fault location might actually be easier for higher-order masked implementations since the runtime of masked S-boxes grows faster than the runtime of the masked linear layer for increasing masking orders.

The implementation of the higher-order masked AES by Coron et al. [6] has a configurable masking order d. Our experiments were performed up to 10th-order masking, with temporal redundancy as an additional fault countermeasure. In order to attack such a protected implementation we exploit statistically ineffective faults. The restriction to ineffective faults is required, since we want to circumvent the redundancy countermeasure and the subsequent statistical fault analysis is required, since the effect of a fault is almost impossible to predict, in particular if masked and/or unknown implementations are attacked.

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. Since our attacks work comparably well for any masking order d we only state concrete results for $d = 10$ in this section.

Results. In Fig. 10, we show the distribution of the AES state bytes in round 9 from the 48 collected correct ciphertexts. 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. Sect. 2.1). Note that in the attack we only exploit correct ciphertexts and thus successfully circumvent the redundancy countermeasure. Already 20 (of the 48) collected correct ciphertexts stemming from about 1000 faulted encryptions the SEI of the observed distributions is the highest for the correct partial key guess. The time required for collecting the correct ciphertexts was about 3 min, and key recovery was performed in about 2 min with negligible memory requirements. Repetitions of the experiment lead to very similar results.



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

Fig. 10. 10th-order masked AES with temporal redundancy. One clock glitch was performed during the calculation of the masked Sbox in round 9. Distribution of AES state bytes after S-box in round 9 after collecting a sufficient amount of correct ciphertexts. 32 key bits can be recovered.

4.2 Simulated Attacks on AES from Schwabe and Stoffelen

In this section, we present a comprehensive analysis of simulated faults for an assembler-optimized, masked AES implementation for the 32-bit Cortex-M4 platform with high practical relevance of Schwabe and Stoffelen [25]. This implementation can encrypt two 128-bit inputs per block cipher call in CTR mode and is fully unrolled. For our purposes, we add temporal redundancy but only encrypt one 128-bit input per block cipher call in ECB mode to have the same scenario as in the previous section. We briefly describe the masked S-box implementation of Schwabe and Stoffelen and then we discuss the results of our evaluation that is split into three parts:

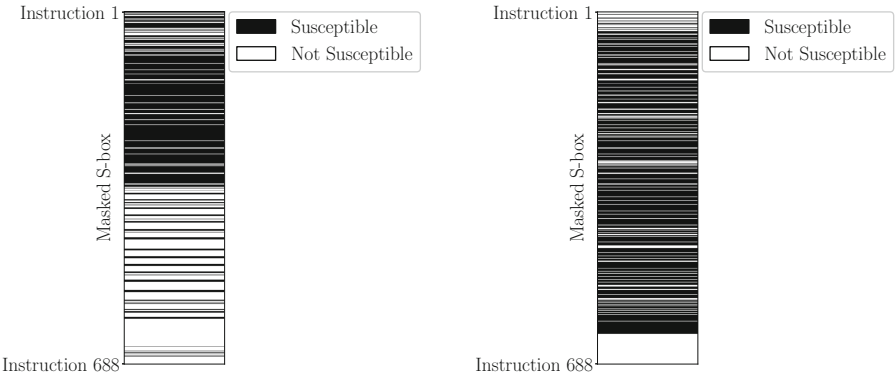
First, we analyse how many instructions in the masked S-box in round 9 are "*susceptible*" to faults considering two common fault models, i.e., would allow an attacker to mount SIFA. Then we pick one susceptible instruction and discuss the required effort of mounting SIFA considering 7 common fault models. For each fault model we give the required amount of faulted encryptions and the amount of recoverable key bits. Finally, we present a figure that illustrates the effect of faulting the masked Sbox on the unmasked AES state bytes (that are observed during key recovery).

Optimized, Bitsliced, 1st-Order Masked S-Box. One shared S-box computation consists of 688 instructions and is executed in parallel for the entire AES state. The implementation is based on the efficient bitsliced, S-box implementation from Boyar et al. [4]. Masking via Trichina gates [26] as well as efficient platform-specific scheduling was added by Schwabe and Stoffelen [25].

Results – Susceptible Instructions. In this section, we demonstrate that SIFA is neither restricted to a specific fault model nor requiring precise information of the attacked implementation. We have performed experiments using two common fault models (single bitflip and byte-stuck-at-0) and simulated fault inductions that cause an erroneous value in the result of the targeted instruction of an S-box computation in round 9. This simulation is performed separately for each of the 688 instructions in the masked S-box. For each instruction and both types of simulated faults we performed 2000 encryptions and collected correct ciphertexts from unaffected encryptions (if there were any). Then we performed key recovery, i.e., for each targeted instruction, both types of simulated faults, and the corresponding set of collected ciphertexts we guessed 32 bits of the last round key, calculated back to round 9 and checked if some bytes follow a non-uniform distribution (using the SEI distinguisher). To reduce the runtime of the evaluation we took one shortcut by always fixing 16-bit of the 32-bit partial key guess to the correct value. While this significantly reduced the runtime of our exhaustive analysis, it does not affect the results.

The results of our analysis are presented in Fig. 11. Figure 11a shows for each of the 688 instructions within the masked S-box whether or not single bitflips in this specific instruction allow an attacker to mount SIFA, i.e., recover bits

of the key. Note that black lines represent susceptible instructions while white lines represent the other instructions. An instruction is not susceptible, e.g., if a bitflip is never ineffective, always ineffective, or does not lead to a non-uniform distribution that is distinguishable from uniform after observing 2 000 faulted encryptions. In total, 359 out of 688 (52%) of the instructions are susceptible to single bitflips. In Fig. 11b, we show the same analysis using byte-stuck-at-0 faults instead. Here, 483 (70%) of the instructions are susceptible. If we oppose these results with the fact that the masked linear layer in round 9 only consists of 290 instructions, it is fairly safe to say that finding a suitable fault location should be easy in practice.



(a) Bitflips: 359 (52%) of the S-box instructions are susceptible.

(b) Byte-stuck-at-0: 483 (70%) of the S-box instructions are susceptible.

Fig. 11. Instructions in the masked S-box in round 9 that are susceptible to faults and allow to mount SIFA.

Results – Attack Performance for Various Fault Models. After determining that large parts of the masked S-box are susceptible to single fault inductions in the previous section, we now discuss the effort of key recovery when targeting one of these susceptible instructions. This time we consider 7 different fault models and the results are presented in Table 2. For each of the 7 fault models we give the number of faulted encryptions, the number of resulting ineffective fault inductions (i.e., correct ciphertexts), and the number of key bits that can be recovered from those correct ciphertexts. Each experiment was repeated 3 to 5 times, and the averaged values are presented. The relative position of the affected bytes/bits within the targeted 32-bit register is not important for the analysis.

From the results we can see that faults with fine granularity (only affecting single bits) allow an attacker to recover key bits faster (= using less faulted encryptions). However, only 32 key bits can be recovered in these scenarios since only a small portion of the AES state is affected by those fault inductions. On the other side, faults that affect whole bytes/registers require more faulted

Table 2. 32-bit Cortex-M4: Attack evaluation when targeting one of the instructions in the masked S-box in round 9. Each experiment was repeated 3 to 5 times, the resulting numbers were averaged.

Fault effect	# Ineffective faults	# Faulted encryptions	# Recoverable key bits
Flip one bit	194	386	32
Set one bit to zero	214	428	32
Randomize one bit	574	763	32
Flip one byte	192	2 940	128
Set one byte to zero	192	3 129	128
Randomize one byte	602	1 808	128
Instruction skip	400	45 527	128

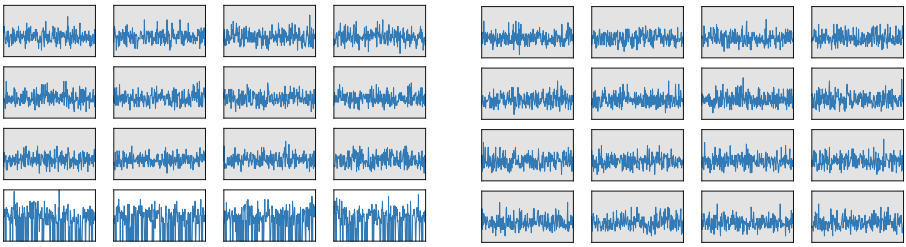
encryptions but cause a non-uniformity in bigger portions of the AES state. Consequently, an attacker can recover more key bits.

Results – Non-uniformity of AES State Bytes. Finally, we present more arguments why the statistical fault analysis (SFA) portion of SIFA is crucial to mount such manifold attacks as presented in this work. If we take a look at Fig. 12, we can see the distribution of AES state bytes in round 9 after inducing one byte-stuck-at-0 fault during multiple encryptions and performing key recovery using unaffected, correct ciphertexts. Even though our simulated faults are noise-free, i.e., they have the same effect on each faulted instruction, there is no meaningful way (other than having precise knowledge of the attacked implementation and the induced fault) for the attacker to predict the resulting non-uniformity in the state bytes which would allow for faster key recovery than in SIFA (or SFA). In fact, the observed distributions have some relations with the pen-and-paper examples given in Sect. 3.3 and can be expected to vary significantly depending on the attacked implementation, the fault location, and the actual fault effect. This motivates our choice of simply using a metric of non-uniformity to distinguish the key candidates.

5 Discussion

5.1 On the Nature and Number of Faults

In Sect. 3, we explored 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 understand. However, it is easy to see that making the fault probabilistic, e.g., assuming a more realistic setup, where an instruction skip does not work all the time, or that a bit is only set to 0 with a certain probability, just affects the bias an attacker observes and hence, the amount of ciphertext the attacker has to collect, but the attack still works (see [10] for more details).



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

Fig. 12. 1st-order masked, bitsliced AES with temporal redundancy. Byte-stuck-at-0 fault model. Distribution of AES state bytes after S-box in round 9 after collecting a sufficient amount of correct ciphertexts. The whole 128-bit key can be recovered.

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. For instance, in Sect. 4.2 we show that attacks are possible even for random faults and bitflips. 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 a suitable place in the primitive among the filtered encryptions. In general, a fault can have a more complex nature than only the cases discussed in Sect. 3 or 4.2. For instance, in software implementations of masked ciphers, a large number of instructions are LOAD instructions from memory since all shares might not fit in the registers. We have observed in experiments that skipping 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.

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 Sect. 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. In Sect. 4.2, we evaluated the number of instructions that can be faulted and in turn, can be exploited in an attack for one particular implementation. However, how many such locations exist 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 or 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 rather cheap setup using clock glitches as demonstrated in Sect. 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 SFA, we are not interested in 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 might to reduce the required number of faulted encryptions, 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 SIFA in practice.

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 a few 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.

One way to compensate some false positives is to destruct a device only once a certain amount of faults was detected. Such a fault counter could be considered an effective countermeasure, yet is still not used by a large portion of embedded devices, since it is challenging to implement appropriately tamper-resistant, especially in hardware.

Correction. 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 will always be

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 before, using a second targeted fault.

Infection. In [10], the application of SIFA on an infective countermeasure [27] 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.

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 $1,\ldots,R$ 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 π_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 π_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

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

This parametrized function is plotted in Fig. 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 Fig. 13 (dashed, with x-axis $t = k$), and can be approximated as

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

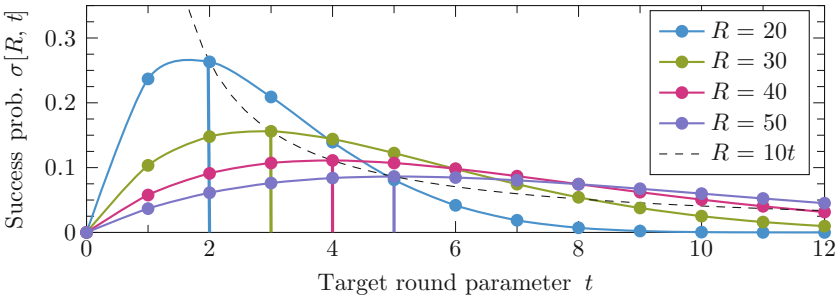


Fig. 13. Success probability $\sigma[R, t]$ when targeting round $R - t$, for different $R = 10k$. For the choice $t = k$ (dashed line), $\sigma[10k, k] \approx \frac{0.387}{k}$ for $k \gg 2$.

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}:

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

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

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

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

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 (Fig. 14):

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

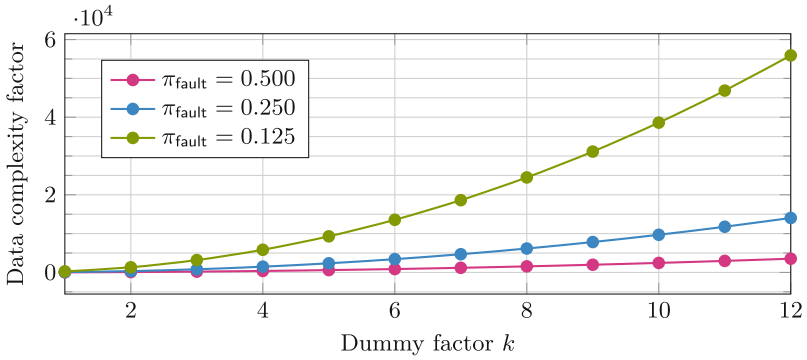


Fig. 14. Increasing data complexity with dummy factor k, for different π_{fault}.

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 $\pi_{\mathsf{fault}}^{-1}$.

Limiting the Data Complexity. For our attacks to work, we usually need several faulted 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 function from a static master key [1,11,12,18] 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 Sect. 3 or AES in Sect. 4 as targets of our attacks because we have found them to be weaker than others. In fact, we chose them, because many masked implementations for them are publicly available.

Furthermore, we performed the practical experiments using clock glitches because the equipment is cheap and we do not have easy access to other, more sophisticated equipment at the moment. Obviously, the attack is not limited to a specific fault injection method; quite on the contrary, we expect other methods of inserting faults, such as Lasers, needle probes, etc., to be far superior compared to our cheap setup using clock glitches [9].

5.4 Further Applications

For the sake of simplicity, we have put the main focus of this paper on the application of statistical fault attacks for (masked) design strategies using bijective S-boxes, where we want to distinguish a uniform from a non-uniform distribution. However, this does not mean that the attack in only applicable on primitives using bijective S-boxes. As discussed in Sect. 3.4, a fault attack may influence only some transitions in the transition graph, while leaving others intact. In the case of 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 with against our attack is the use of a PRF like the AES-PRF [19] instead of a block cipher. Such a PRF prevents an attacker from observing ciphertexts and decrypting backwards under guessing the key. However, SIFA remains possible by targeting the input of the AES-PRF since here, a known input like a nonce is usually processed. In general, the 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 to evaluate how well such attacks will 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 presented a comprehensive analysis of simulated faults for an assembler-optimized, masked AES implementation for the 32-bit Cortex-M4 platform that might be of high practical relevance. We showed that most of the instructions of the masked S-box implementation are "susceptible" to faults and can be exploited in SIFA using any of the common fault models.

Moreover, the practical feasibility of the attack was shown by attacking a 10th-order masked AES software implementation with arbitrary temporal redundancy on block cipher level on a standard 8-bit microcontroller without specific security features using a cheap clock glitch setup. Even with such a cheap setup, we are able to recover 32 bits of the key after collecting 20 ciphertexts where the fault is ineffective, needing approximately a total of 1000 encryptions where a single fault induction is performed.

Acknowledgments. This project has received funding in part from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402) and by the Austrian Research Promotion Agency (FFG) via the project ESPRESSO, which is funded by the province of Styria and the Business Promotion Agencies of Styria and Carinthia.

References

1. Berti, F., Pereira, O., Peters, T., Standaert, F.X.: On leakage-resilient authenticated encryption with decryption leakages. IACR Trans. Symmetric Cryptol. **2017**(3), 271–293 (2017)
2. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052259
3. Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., Van Assche, G.: Efficient and first-order DPA resistant implementations of Keccak. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 187–199. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_13
4. Boyar, J., Peralta, R.: A depth-16 circuit for the AES S-box. IACR Cryptology ePrint Archive, Report 2011/332 (2011). https://eprint.iacr.org/2011/332
5. Clavier, C.: Secret external encodings do not prevent transient fault analysis. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 181–194. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_13
6. Coron, J.S.: Higher order countermeasures for AES and DES (2017). https://github.com/coron/htable#higher-order-countermeasures-for-aes-and-des
7. Daemen, J.: Cipher and hash function design, strategies based on linear and differential cryptanalysis. Ph.D. thesis, KU Leuven (1995). http://jda.noekeon.org/
8. Daemen, J., Govaerts, R., Vandewalle, J.: An efficient nonlinear shift-invariant transformation. In: Macq, B. (ed.) Information Theory in the Benelux. pp. 108–115. Werkgemeenschap voor Informatie- en Communicatietheorie (1994)
9. Dobraunig, C., Eichlseder, M., Korak, T., Lomné, V., Mendel, F.: Statistical fault attacks on nonce-based authenticated encryption schemes. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 369–395. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_14
10. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 547–572 (2018)
11. Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F., Unterluggauer, T.: ISAP - towards side-channel secure authenticated encryption. IACR Trans. Symmetric Cryptol. **2017**(1), 80–105 (2017)
12. Dobraunig, C., Koeune, F., Mangard, S., Mendel, F., Standaert, F.-X.: Towards fresh and hybrid re-keying schemes with beyond birthday security. In: Homma, N., Medwed, M. (eds.) CARDIS 2015. LNCS, vol. 9514, pp. 225–241. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31271-2_14
13. Fuhr, T., Jaulmes, É., Lomné, V., Thillard, A.: Fault attacks on AES with faulty ciphertexts only. In: Fischer, W., Schmidt, J.M. (eds.) FDTC 2013, pp. 108–118. IEEE Computer Society, Los Alamitos (2013)
14. Groß, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. IACR Cryptology ePrint Archive, Report 2016/486 (2016). https://eprint.iacr.org/2016/486

15. Gross, H., Mangard, S., Korak, T.: An efficient side-channel protected AES implementation with arbitrary protection order. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 95–112. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_6
16. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
17. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
18. Medwed, M., Standaert, F.-X., Großschädl, J., Regazzoni, F.: Fresh re-keying: security against side-channel and fault attacks for low-cost devices. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 279–296. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12678-9_17
19. Mennink, B., Neves, S.: Optimal PRFs from blockcipher designs. IACR Trans. Symmetric Cryptol. **2017**(3), 228–252 (2017)
20. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 529–545. Springer, Heidelberg (2006). https://doi.org/10.1007/11935308_38
21. Quisquater, J.-J., Samyde, D.: ElectroMagnetic Analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45418-7_17
22. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_37
23. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_28
24. Schneider, T., Moradi, A., Güneysu, T.: ParTI – towards combined hardware countermeasures against side-channel and fault-injection attacks. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 302–332. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_11
25. Schwabe, P., Stoffelen, K.: All the aes you need on Cortex-M3 and M4. In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 180–194. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_10
26. Trichina, E.: Combinational logic design for AES SubByte transformation on masked data. IACR Cryptology ePrint Archive, Report 2003/236 (2003). https://eprint.iacr.org/2003/236
27. Tupsamudre, H., Bisht, S., Mukhopadhyay, D.: Destroying fault invariant with randomization. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 93–111. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_6
28. Ullrich, M., De Cannière, C., Indesteege, S., Küçük, Ö., Mouha, N., Preneel, B.: Finding optimal bitsliced implementations of 4×4-bit S-boxes. In: ECRYPT Symmetric Key Encryption Workshop - SKEW 2011, pp. 16–17 (2011)