SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography

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

[1] Graz University of Technology, Austria
first.last@iaik.tugraz.at
[2] Infineon Technologies AG, Germany
first.last@infineon.com

Abstract. Since the seminal work of Boneh et al., the threat of fault attacks has been widely known and techniques for fault attacks and countermeasures have been studied extensively. The vast majority of the literature on fault attacks focuses on the ability of fault attacks to change an intermediate value to a faulty one, such as differential fault analysis (DFA), collision fault analysis, statistical fault attack (SFA), fault sensitivity analysis, or differential fault intensity analysis (DFIA). The other aspect of faults—that faults can be induced and do not change a value—has been researched far less. In case of symmetric ciphers, ineffective fault attacks (IFA) exploit this aspect. However, IFA relies on the ability of an attacker to reliably induce reproducible deterministic faults like stuck-at faults on parts of small values (e.g., one bit or byte), which is often considered to be impracticable.
As a consequence, most countermeasures against fault attacks do not focus on such attacks, but on attacks exploiting changes of intermediate values and usually try to detect such a change (detection-based), or to destroy the exploitable information if a fault happens (infective countermeasures). Such countermeasures implicitly assume that the release of "fault-free" ciphertexts in the presence of a fault-inducing attacker does not reveal any exploitable information. In this work, we show that this assumption is not valid and we present novel fault attacks that work in the presence of detection-based and infective countermeasures. The attacks exploit the fact that intermediate values leading to "fault-free" ciphertexts show a non-uniform distribution, while they should be distributed uniformly. The presented attacks are entirely practical and are demonstrated to work for software implementations of AES and for a hardware co-processor. These practical attacks rely on fault induction by means of clock glitches and hence, are achieved using only low-cost equipment. This is feasible because our attack is very robust under noisy fault induction attempts and does not require the attacker to model or profile the exact fault effect. We target two types of countermeasures as examples: simple time redundancy with comparison and several infective countermeasures. However, our attacks can be applied to a wider range of countermeasures and are not restricted to these two countermeasures.

Keywords: fault attack · infective countermeasure · fault detection · countermeasure · statistical ineffective fault attack · SIFA

1 Introduction

Shortly after the seminal work of Boneh et al. [BDL97] showed fault attacks on RSA, it became clear that also symmetric schemes are susceptible to this type of active implemen-

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

IACR Transactions on Cryptographic Hardware and Embedded Systems ISSN 2569-2925, Vol. 2018, No. 3, pp. 547–572
DOI:10.13154/tches.v2018.i3.547-572

tation attacks. Starting with the differential fault analysis (DFA) of DES by Biham and Shamir [BS97], a rich field of research emerged that focuses on techniques to recover the secret key from faulty ciphertexts. Meanwhile, there exists a wide range of publications on how to apply DFA attacks to different cryptographic algorithms and in particular to AES [AMT13]. In addition, novel attack techniques have been introduced such as fault sensitivity analysis (FSA) [LSG+10], differential fault intensity analysis (DFIA) [GYTS14] and statistical fault attacks (SFA) [FJLT13].

In order to prevent an attacker from learning the secret key from faulty outputs, extensive research on countermeasures has been conducted. There are essentially two main categories of countermeasures. The first category covers sensor-based countermeasures that aim at detecting the physical process of the fault induction. For example, protected implementations may include light, voltage and temperature sensors in order to detect fault inductions by lasers [SBHS15], voltage glitches [BECN+06], or temperature variations [HS13]. Countermeasures of this kind have a long tradition in smart card industry. However, as there are more and more ways to induce faults, the focus is more and more on countermeasures that aim at managing the effect of a fault induction. This is the second category of countermeasures and also the main focus of the academic research.

The effect of a fault induction on a cryptographic algorithm can be modelled as the change of an intermediate variable x to a faulty intermediate variable x'. Such a change of a value can occur due to a direct modification of the variable x, but also due to instruction skips or addressing errors. Independent of the exact effect that leads from x to x', there are two approaches on how to prevent that this change is exploited by an attacker. The first approach is to detect the difference $\Delta = x - x'$ by adding redundancy to a design and to suppress an output in case $\Delta \neq 0$. Corresponding redundancy techniques range from simple temporal or spacial duplications to error detection codes. The second approach for managing $\Delta \neq 0$ are infection-based countermeasures. In this case, a cipher output is always provided, but the goal is to change the ciphertext in a way that the ciphertext becomes useless for an attacker.

While most attack techniques and countermeasures focus on exploiting or preventing information leakage in case $\Delta \neq 0$, the question of whether an attacker can also learn information from ineffective faults has not been explored in depth so far. A fault induction is ineffective in case the fault induction is performed (e.g., a voltage glitch is performed), but it holds that $\Delta = 0$ and the cipher output is consequently not changed due to the fault induction. Information leakage on the secret key can occur in this case, if there is a dependency between the fault induction being ineffective and the data that is processed.

Exploiting ineffective fault inductions typically implies that a larger number of fault inductions needs to be performed than in case of classical fault attacks that exploit $\Delta \neq 0$. While in case of classical fault attacks a few selected fault inductions are usually sufficient to determine the key, the exploitation of ineffective faults requires that fault inductions are performed on many different data inputs in order to find ineffective fault inductions.

To the best of our knowledge, the first attacks that exploit ineffective inductions are ineffective fault attacks (IFA) by Clavier [Cla07]. For IFA, it is assumed that an attacker can reliably force an intermediate value x to a known value (e.g., 0) by a fault induction. Then, the basic idea is to feed random input data into a device and to perform the fault inductions until an ineffective induction is observed. In this case, the intermediate x is known and the secret key can be determined. The big drawback of this attack is that it requires a precise fault induction for a large number of encryptions, which is very difficult to achieve in practice.

Our contribution. In this work, we generalize IFA attacks and introduce statistical ineffective fault attacks (SIFA). As we argue and show with practical evaluations, SIFA is typically not only applicable when SFA or IFA is applicable, but also in a broader range of scenarios - in particular in the presence of countermeasures. Our attack does not rely on a specific fault model. We simply require that there is some dependency between the observation of an ineffective fault induction and the faulted intermediate value x. However, the attacker does not need to know any further details of this dependency. This means simply that the probability for changing an intermediate value x due to a fault induction is not the same for all values x. This bias of the probabilities for ineffective fault inductions is the sole requirement on the fault induction.

Like IFA, SIFA can be applied in settings where it is possible to perform many fault inductions on encryptions with different data inputs and to observe whether the fault induction was ineffective. While IFA typically requires strong fault models like stuck-at faults, the requirements for SIFA on the fault induction are minimal and corresponding faults can be induced easily in practice with a high frequency and without the need for sophisticated laboratory equipment.

To show this, we attack protected implementations that feature countermeasures against fault attacks like SFA or DFA. In particular, we target countermeasures based on detection and infection. In fact, countermeasures that are based on managing a fault effect $\Delta \neq 0$ are ideal targets for SIFA. These countermeasures allow the attacker to collect observations where the fault induction was ineffective. Our empirical study shows that these countermeasures can be easily bypassed in practice and that it is necessary to combine them with additional countermeasures to provide protection against SIFA attacks.

Our concrete attack results are as follows. First, we target a detection-based countermeasure for AES that uses simple time redundancy with subsequent comparison. In order to show the robustness of our attack, this evaluation is performed on 3 different AES implementations, attacking 8-bit and 32-bit-bitsliced software implementations as well as a hardware co-processor. The fault is induced by using a simple clock glitch. In all cases, the number of needed faulty encryptions is comparably low. SFA is not applicable here, since no exploitable faulty output is released. Although IFA is not prevented by simple time redundancy with subsequent comparison, it still relies on precise stuck-at faults in certain bytes, which are hard to achieve in practice, especially in the case of the 32-bit-bitsliced and the hardware co-processor implementations. In contrast, SIFA can exploit any case where ineffective faults lead to a biased distribution, even without knowledge about the distribution of these values.

We then target infective countermeasures, where typically neither SFA nor IFA are applicable. Here, we extend the software AES implementation from the AVRCryptoLib [avr] and evaluate our attack for multiple security parameterisations. Again, simple clock glitches are used to induce the required faults, resulting in attacks that are rather easy to execute in practice and do not require any expensive laboratory equipment.

Related work. SIFA extends and connects several other ideas that have previously been published in the literature. One keypoint of the presented attack is the fact that it exclusively exploits cases where a fault does not change the result of the computation. Therefore, our attack shares a common reference point with safe-error attacks [YJ00] and IFA [Cla07]. In a safe-error attack, the value of an intermediate variable is changed (fault effect $\Delta \neq 0$) and the knowledge whether the faulted value is used or not is exploited. Typically, safe-error attacks are used to attack asymmetric schemes. In contrast, ineffective fault attacks [Cla07] exploit specific cases where $\Delta = 0$ and the fault shows no effect. More concretely, IFA relies on strong and known fault models, like precise stuck-at-0 faults, in order to probe values of intermediate variables.

We extend this idea from stuck-at faults as already used by Biham and Shamir [BS97] to the case that ineffective faults lead to a non-uniform distribution of intermediate values. As a result, we do not probe specific values; rather, we exploit the non-uniform distributions. Hence, we are naturally able to deal with noise (e.g., failure of fault induction, or faults induced at a wrong position), which allows us to demonstrate the attack in practice. Our used methods to exploit non-uniform distributions are related to those in SFA [FJLT13].

Outline. First, we give a short overview and summary of statistical fault attacks and the reviewed countermeasures in section 2. Then, we state the idea and show the working principle of the attack in section 3. Section 4 contains the results of our practical attack and we finally conclude in section 6.

2 Background

In this section, we first give a brief introduction to countermeasures against fault attacks and review the countermeasures we put our focus on more closely. Then we discuss two attacks which are related to our attack: ineffective fault attacks and statistical fault attacks.

2.1 Countermeasures

To protect against fault attacks, countermeasures aim to detect or prevent faults either on the physical layer (e.g., light sensors or supply voltage detectors) or on an algorithmic level. In this work, we solely focus on the second category. The strategy of detection-based countermeasures is to detect that a fault changes an intermediate value, e.g., by performing redundant operations. If a fault is detected, the computation is aborted and no ciphertext is returned. In contrast, infection-based countermeasures always return a ciphertext, but attempt to process the ciphertext in such a way that the output becomes useless for an attacker in case of faults during the computation. Next, we review the detection-based and the infection-based countermeasure that we target in our practical evaluation.

2.1.1 Detection-based Countermeasure

In this work, we consider detection-based countermeasures that detect faults by means of redundant operations. An overview of various techniques that achieve detection of faults with the help of redundant operations is given by Bar-El et al. [BECN+06]. We focus on simple time redundancy with comparison, although the attack is applicable to a wide range of detection-based countermeasures. The idea of this countermeasure (Algorithm 1) is to encrypt each plaintext block twice. Then, the resulting ciphertexts are compared. Only if they match, the ciphertext is released.

Algorithm 1 Simple time redundancy with comparison

Input: key K, plaintext P
Output: ciphertext $C = E_K(P)$, or $\perp$

$C_1 \leftarrow E_K(P)$
$C_2 \leftarrow E_K(P)$
if $C_1 \neq C_2$ **return** $\perp$
return C_1

Detection-based countermeasures would also allow to include mechanisms that disable a device upon a certain amount of fault inductions. While this approach sounds very appealing at first glance, it is very hard to realize in practice. On the one hand, there is the risk of false positives that might lead to the disabling of a device in regular use cases (e.g., due to supply problems when being powered by an electromagnetic field). On the other hand, there is the need to count faults in such a way that it cannot be easily bypassed by an attacker. For example, a simple increasing of a counter in non-volatile memory can be detected easily by an attacker in the power trace and due to the timing behavior. Hence, an attacker can detect whether a fault induction was effective or not and can remove the power supply during the programming of the memory in order to prevent the increasing of the counter. No sophisticated equipment is needed in this case. For more secure counting mechanisms, dedicated hardware support is required, which is not available in most devices (e.g., IoT devices) that are exposed to fault attacks. However, in applications that allow realizing fault counting in a secure and reliable manner without the risk of too many false positives, it is an effective countermeasures against classic fault attacks as well as IFA.

2.1.2 Infective Countermeasure

In contrast to detection-based countermeasures that aim to detect a fault and then do not release a ciphertext, infective countermeasures always provide a ciphertext, but amplify a possibly induced fault in such a way that a faulty ciphertext becomes useless for an attacker. As an example for infection-based countermeasures, we consider the infective countermeasure presented by Tupsamudre et al. at CHES 2014 [TBM14] as an extension of an infective countermeasure presented by Gierlichs et al. [GST12]. Patranabis et al. [PCM15] give a formal proof for this countermeasure against differential fault analysis using a single fault injection under the assumption that the sequence of executed instructions is neither skipped, nor altered. The only attacks on this countermeasure so far are attacks that either skip or alter instructions [BG16]. The approach is summarized in Algorithm 2.

Algorithm 2 Infective countermeasure by Tupsamudre et al. (taken from [TBM14])

Input: P, k^j for $j \in \{1, \dots, n\}$, (β, k^0), $(n = 11)$ for AES-128
Output: $C = E_K(P)$, or infected state

State $R_0 \leftarrow P$, Redundant state $R_1 \leftarrow P$, Dummy state $R_2 \leftarrow \beta$
$i \leftarrow 1$, $q \leftarrow 1$
$rstr \xleftarrow{\$} \{0,1\}^t$ // $\#1(rstr) = 2n$, $\#0(rstr) = t - 2n$
while $q \leq t$ **do**
  $\lambda \leftarrow rstr[q]$ // $\lambda = 0$ implies a dummy round
  $\kappa \leftarrow (i \wedge \lambda) \oplus 2(\neg\lambda)$
  $\zeta \leftarrow \lambda \cdot \lceil i/2 \rceil$ // ζ is actual round counter, 0 for dummy
  $R_\kappa \leftarrow RoundFunction(R_\kappa, k^\zeta)$
  $\gamma \leftarrow \lambda(\neg(i \wedge 1)) \cdot BLFN(R_0 \oplus R_1)$ // check if i is even
  $\delta \leftarrow (\neg\lambda) \cdot BLFN(R_2 \oplus \beta)$
  $R_0 \leftarrow (\neg(\gamma \vee \delta) \cdot R_0) \oplus ((\gamma \vee \delta) \cdot R_2)$
  $i \leftarrow i + \lambda$
  $q \leftarrow q + 1$
return R_0

We will now give the basic intention behind Algorithm 2. For a more detailed description we refer to the original work of Tupsamudre et al. [TBM14]. Algorithm 2 works on three different states R_0, R_1 and R_2. State R_0 is initialized with the plaintext P and is the state on which the primary AES computation is performed. State R_1 is also initialized with P and serves as working state for the redundant AES computation. In the fault-free case, both states R_0 and R_1 should contain the ciphertext at the end of the computation. The state R_2 is initialized with a random 128-bit value β and serves as working state for the dummy round calculations. The key k^0 is chosen such that $RoundFunction(\beta, k^0) = \beta$.

Before the computation starts, a random string $rstr$ of length t is initialized randomly so that it contains 22 bits "1" and $t - 22$ bits "0". The algorithm iterates over $rstr$ and executes for every "1" an AES round on R_0, or a redundant round on R_1 (22 rounds for 2 times 10 rounds AES plus 2 times the whitening key addition) in an alternating sequence, i.e., if a round on R_0 has been calculated, the next "1" executes a redundant round on R_1 so that after this calculation, the content of R_0 and R_1 should be the same in a fault-free case. For every "0", a dummy round is computed to update R_2. The security level with respect to the number of dummy rounds that are executed depends on the size of t and can be chosen by the developer.

After every executed AES round, the algorithm checks if any of the values in registers R_0, R_1, or R_2 has been modified ($R_0 \neq R_1$ or $R_2 \neq \beta$). If this is the case, state R_0 is, from this point on, always overwritten with the content of R_2, which is then returned as ciphertext. Since the value stored in R_2 is random and has never been mixed with, nor depends in any other way on the value of the secret key, learning this value should be useless for the attacker.

2.2 Statistical Fault Attacks

Statistical fault attacks (SFA) were introduced by Fuhr et al. [FJLT13] as a method to recover the secret key of AES, if an attacker is able to change an intermediate variable to a biased (i.e., not uniformly distributed) value by inducing a fault. They considered three different fault models on byte level:

1. Stuck-at-0
2. Stuck-at-0 with probability 0.5, or logical AND with random uniform value with probability 0.5
3. Logical AND with random uniform value

Fuhr et al. [FJLT13] evaluated various key recovery strategies dependent on the round where the fault is induced. For instance, they showed that if the fault is induced in one byte right before the last MixColumns application, 6 faulty ciphertexts in case of fault model 1, 14 faulty ciphertexts in case of fault model 2, and 80 faulty ciphertexts in case of fault model 3 are needed to recover 4 bytes of the secret key. These attacks require to partially decrypt every ciphertext back to the faulted byte for each key candidate and measure the squared euclidean imbalance (SEI) of this byte. The key candidate that gives the highest SEI is most likely the correct one.

Since SFAs make use of the ability of faults to change an intermediate value (to a biased value), they can be prevented by both countermeasures discussed in subsection 2.1. To bypass both types of countermeasures, an attacker could try to induce identical faults in both redundant computations and thus evade detection. With strongly biased faults, this may be easier to achieve than for random faults, as has been demonstrated for a detection-based countermeasure [PCNM15]. However, the attacker's task gets more and more complicated with increasing redundancy of the countermeasure.

2.3 Ineffective Fault Attacks

The idea of ineffective fault attacks (IFA) by Clavier [Cla07] is that certain faults can be used to probe intermediate values of a cryptographic algorithm. This technique can be used to circumvent countermeasures like simple time redundancy with comparison. Consider an attacker who induces a stuck-at-0 fault in one byte during one execution of AES, while leaving the other one correct. If the attacker nevertheless receives an output (ciphertext), the faulted value must already have been 0 before the fault. If this stuck-at-0 fault is induced in one byte of the last AES round before the last key addition, we can immediately recover one byte of the last round key: All an attacker has to do is to guess one byte of the key, decrypt the corresponding byte of the correct ciphertext back to the intermediate byte that has been faulted, and check if the resulting byte value is 0. If it is 0, the guessed key byte is the right one. This approach is applicable just as easily for more than two redundant computations, since only one computation needs to be faulted.

However, the assumption that an attacker is able to deterministically change the value of an intermediate variable to 0 requires a very strong and powerful attacker. In practice, an attacker is usually less powerful and has to consider, for instance, false positives in case of failed fault inductions that do not show any effect. For simple time redundancy with comparison, one solution for this specific problem would be to repeat the fault induction several times for encryptions of the same plaintext to get results which are more or less noise-free, as suggested by Clavier and Wurcker [CW13]. Using this strategy typically also causes troubles in the case of infective countermeasures. Although the fault might be induced always at the same byte at the same time, an attacker does not know if the affected byte belongs to a dummy round or not.

In the following, we demonstrate that not only stuck-at faults can be exploited in IFA and introduce statistical ineffective fault attacks. On a high level, these attacks can be seen as an intersection of the principles exploited in the case of IFA [Cla07] and SFA [FJLT13]. In section 3, we explain the necessary conditions for our attack to work and demonstrate in section 4 that they are usually fulfilled when attacking real devices with algorithmic countermeasures. In particular, the attacker does not need to assume any specific fault model and can successfully recover the key even with very "noisy" faults with unpredictable, unreliable effects.

3 Statistical Ineffective Fault Attack

In this section, we discuss the ideas behind the extension from ineffective fault attacks [Cla07] (IFA) to statistical ineffective fault attacks (SIFA). First, we review the effects of faults with the help of fault distribution tables to identify the necessary conditions for SIFA to work in subsection 3.1. Then we introduce the working principle of SIFA in subsection 3.2. Finally, we develop some theoretical background of our attacks in subsection 3.3.

3.1 The Effects of Faults

The effects caused by faults during the execution of cryptographic primitives are manifold and depend on the method used to induce the fault (e.g., laser, clock glitches), the architecture and manufacturing technology of the attacked device, and various other parameters (e.g., targeting a register or arithmetic instruction). However, all faults have in common that they change the value of a b-bit intermediate variable from a value x, which it would have for the correct execution, to a value x' in the presence of a fault. Observing the probability of transitions from a certain value $x \to x'$ gives us a fault distribution table (see subsection 3.3 for the exact definition).

With the help of such a fault distribution table, we are able to characterize the effects of a wide range of faults that can happen in practice. For example, this allows us to capture faults where the value of x' is independent of the value x, like stuck-at faults, random faults, and biased faults, but also more complex relations where x' depends in some sense on x, for instance by faulting the instruction that computes x. In Table 1, we show various examples of fault distribution tables for different faults on a 2-bit intermediate variable.

Most fault countermeasures that work on an algorithmic level can only conceal cases where $x \neq x'$, because a fault that results in $x = x'$ is indistinguishable from a normal working condition. As a consequence, an attacker has access to ciphertexts where the attacked (faulted) intermediate variable follows a distribution determined by the diagonal

Table 1: Fault distribution tables for several 2-bit fault models.

(a) Stuck-at-0

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

(b) Random-And

x \ x'	00	01	10	11
00	1	0	0	0
01	$\frac{1}{2}$	$\frac{1}{2}$	0	0
10	$\frac{1}{2}$	0	$\frac{1}{2}$	0
11	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

(c) Bit-flip

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

(d) Random fault

x \ x'	00	01	10	11
00	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
01	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
10	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
11	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

(red values) in Table 1. The attacks presented in the following sections show that a non-uniform distribution in this diagonal can be exploited to recover the key. Therefore, for an implementation protected by such a fault countermeasure to be resistant against our attack, one of the two following conditions has to be fulfilled: Either the probability that an ineffective fault happens is negligible (as in Table 1c), or the distribution in the diagonal of the fault distribution table is uniform (as in Table 1d).

Although in theory, the bit-flip and random fault models of Table 1c and Table 1d are not susceptible to SIFA, our practical experiments in section 4 indicate that countermeasures cannot rely on the hope that only such "perfect" fault models occur in practice. For instance, consider the case where a bit-flip occurs probabilistically with the tendency to flip more often from 1 to 0 than from 0 to 1, as illustrated in Table 2. The resulting distribution has a biased diagonal.

Table 2: Bit-flip from 1 to 0 with 75 % and from 0 to 1 with 50 %.

x \ x'	00	01	10	11
00	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
01	$\frac{3}{8}$	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{1}{8}$
10	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
11	$\frac{9}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{1}{16}$

In the following section, we will explain how such distributions can be exploited. Since the fault distribution tables are typically not known by an attacker (unless the attacker is able to profile the device), our attack works without any knowledge of the fault distribution table. This is demonstrated by the practical attacks of section 4, which we perform without any knowledge of the underlying fault model and fault distribution table.

3.2 Working Principle

We now consider a protected AES implementation as an example to show the working principle of SIFA. The attack can be split into 3 phases. The first phase is the actual fault attack and collection of suitable ciphertexts. In the second phase, parts of the last round key are guessed and the distribution of an intermediate state is evaluated. In the last phase of the attack, the partial key-guesses are ranked according a metric (e.g., the Squared Euclidean Imbalance (SEI)) and the correct key is identified.

Collecting ciphertexts. Assume that we target one byte before the last application of MixColumns. We request the ciphertexts for a number of plaintexts and fault each encryption. If the implementation is protected with a detection-based countermeasure, we only obtain those ciphertexts where the fault was ineffective; in case of an infective countermeasure, we need to filter for ineffective faults ourselves by comparing the obtained ciphertexts with a second, unfaulted encryption (or decryption).

Key guessing. Following the fault model of subsection 3.1, we obtain a set of filtered ciphertexts whose intermediate value in one byte before the last MixColumns is non-uniformly distributed according to the diagonal of the fault distribution table. This information can be exploited to recover 32 bits of the last round key with a key-guessing strategy similar to the approach of SFA [FJLT13]: The attacker guesses 4 bytes of the last round key K_{10} and partially decrypts the last operations for each correct ciphertext to obtain a partial state S_9:

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

Then, the attacker can evaluate the distribution of the byte in (1) where the fault has been induced, for example by computing the Squared Euclidean Imbalance (SEI) of the byte for each key candidate. In case of evaluating the SEI, no information of the penultimate round key has to be guessed, because the constant key addition changes only the values of the byte, but has no influence on the non-uniformity of the distribution.

Determining the correct key. In the previous phase, for each key candidate, the SEI of the targeted byte has been calculated. We assume that the right key leads to the distribution with the highest SEI if a sufficient number of ciphertexts is evaluated. A closer insight in the number of needed ciphertexts is given in subsection 3.3.

We want to point out that this attack allows to exploit any ineffective fault that causes a non-uniform distribution of an intermediate value, even if the distribution is not known by an attacker (as demonstrated in section 4). In addition, SIFA is robust against noise introduced by failed fault induction attempts, or a fault induction in dummy rounds in the case of the infective countermeasure. In the next section, we provide a statistical model for our attacks and justify the use of the SEI.

3.3 Statistical Model

In this section, we provide a more detailed statistical model of the attack. Our aim is to investigate the effect of various parameters, such as the fault distribution and the configuration of the countermeasure, on the necessary number of faulted ciphertexts to perform the attack with a certain success probability. We compare two scenarios: The practical scenario where the fault distribution is unknown to the attacker (CHI/SEI statistic), but also the theoretical scenario where the attacker happens to know the distribution (LLR statistic). The emphasis of our analysis is on the hardest case: An unknown fault distribution, close to uniform, with additional noise induced by countermeasures.

We consider the b-bit intermediate variable which contains the result of the operation targeted by the fault, and consider its distribution during the attack in more detail. From the attacker's point of view, the value of this variable on a particular input (in absence of faults) is a random variable X which depends on the input and key. Additionally, the random variable X' denotes the value of this variable on the same input, but where the attacker additionally attempted to fault the operation. We also refer to X and X' as "before" and "after" the fault, although this is not strictly accurate. Both X and X' take values $x \in \mathcal{X} = \{0, \dots, 2^b - 1\}$. The action of the fault can be characterized by the transition probabilities

$$p_x(x') := \mathbb{P}[X' = x' | X = x].$$

In practice, this fault distribution table $\mathrm{FDT} = (p_x(x'))_{x,x'}$ is usually not known, or can only be roughly estimated. To perform the proposed attack, the attacker does not need to know the FDT. However, the success and efficiency of the attack depends on some of the table's properties. In the following, we will analyze the attack complexity and its dependency on the two relevant metrics: The fault's ineffectivity rate $\pi_=$, and the capacity $C(p)$ of the target distribution p.

3.3.1 Direct sampling: Detection countermeasure

We first consider attacks on detection-based countermeasures. We can only take advantage of samples where $X = X'$, i.e., the fault is ineffective. We assume that X is uniformly distributed, that is, $\mathbb{P}[X = x] = 2^{-b}$. Then, the probabilities $\pi_=$ of an ineffective fault (ineffectivity rate) and $\pi_{\neq}$ of an effective fault are

$$\pi_= = \mathbb{P}[X' = X] = \sum_{x' \in \mathcal{X}} \frac{p_{x'}(x')}{2^b}, \qquad \pi_{\neq} = 1 - \pi_=.$$

We target the conditional distribution $p_=(x')$ of X' in case of ineffective faults, i.e., the diagonal of the fault distribution table (see subsection 3.1):

$$p_=(x') := \mathbb{P}[X' = x' | X' = X] = \frac{p_{x'}(x')}{2^b \cdot \pi_=}.$$

The attacker neither knows this distribution, nor can she directly observe X'. However, based on the observed cipher output and a key hypothesis for the κ-bit last-round key material as in subsection 3.2, she obtains a hypothesis $\hat{X}'$ for the value of X', and can analyze the distribution $\hat{p}$ of $\hat{X}'$ for a fixed key guess across multiple samples. For an incorrect key guess, we assume a distribution very close to uniform[1]. For the correct key guess, we sample the unknown distribution $p(x') = p_=(x')$. If $p_=(x')$ differs significantly from uniform, we can distinguish these two cases, and identify the samples from $p_=(x')$ produced by the correct key k_0 among the collection of samples from the nearly uniform distributions $\theta_i(x') \approx \theta(x') = 2^{-b}$ produced by the wrong keys k_i, $1 \leq i < 2^\kappa$.

To identify the correct key k_0 and its distribution $p = p_=$, we associate a score statistic $S(\hat{p})$ with each key candidate and the corresponding distribution $\hat{p}$, and rank the key candidates according to this statistic. This approach is closely related to statistical cryptanalysis, such as differential and linear cryptanalysis, and has been theoretically analyzed in those contexts. Under the assumption that $S(\hat{p})$ is independently normally distributed for samples from either p or θ,

$$S(\hat{p}) \sim \begin{cases} \mathcal{N}(\mu_\mathrm{R}, \sigma_\mathrm{R}^2) & \text{if } \hat{p} \text{ was produced by } p\,, \\ \mathcal{N}(\mu_\mathrm{W}, \sigma_\mathrm{W}^2) & \text{if } \hat{p} \text{ was produced by } \theta\,, \end{cases} \tag{2}$$

Selçuk [Sel08] analyzed the success probability of ranking the correct key k_0 among the top $2^{\kappa-a}$ of 2^κ key candidates based on N samples, where a is the advantage. Then, the difference Δ_a between the score of k_0 and the score of the wrong key with rank $2^{\kappa-a}$ (quantile $\alpha = 1 - 2^{-a}$) is normally distributed with parameters

$$\Delta_a \sim \mathcal{N}(\mu_\Delta, \sigma_\Delta^2), \qquad \begin{aligned} &\mu_\Delta = \mu_\mathrm{R} - \mu_\mathrm{W} - \sigma_\mathrm{W}\, \Phi_{0,1}^{-1}(\alpha)\,, \\ &\sigma_\Delta^2 \approx \sigma_\mathrm{R}^2 \qquad \text{for sufficiently large } 2^\kappa \text{ [BGN12]}, \end{aligned}$$

and thus the success probability depending on N and a can be estimated as [Sel08]

$$\mathbb{P}[\Delta_a > 0] \approx \Phi_{0,1}\left(\frac{\mu_\mathrm{R} - \mu_\mathrm{W} - \sigma_\mathrm{W}\, \Phi_{0,1}^{-1}(\alpha)}{\sigma_\mathrm{R}}\right). \tag{3}$$

[1] In practice, this is not necessarily the case, in particular for partially correct key guesses. For example, for a byte-stuck-at fault and a key guess that is only incorrect in one byte, the capacity is expected to drop from 255 to about 1, instead of 0.

To obtain useful complexity estimates from (3), we need a suitably distributed statistic $S(\hat{p})$ and its parameters according to (2). We first consider the (unusual) case[2] that we know the real distribution $p = p_{=}$. Then, the Neyman-Pearson lemma [NP33, CT06] states that the optimal statistic S is the log-likelihood ratio

$$S(\hat{p}) = \mathrm{LLR}(\hat{p}) = \mathrm{LLR}(\hat{p}, p, \theta) := N \sum_{x \in \mathcal{X}} \hat{p}(x) \log_2 \frac{p(x)}{\theta(x)} .$$

For large N, $\mathrm{LLR}(\hat{p})$ tends towards a normal distribution as required in (2) [CT06, BJV04]. The success probability in (3) then depends on the Kullback-Leibler divergence $D(p\|\theta)$:

$$D(p\|\theta) := \sum_{\substack{x \in \mathcal{X} \\ p(x) \neq 0}} p(x) \log_2 \frac{p(x)}{\theta(x)} , \quad D_\Delta(p\|\theta) := \sum_{\substack{x \in \mathcal{X} \\ p(x) \neq 0}} p(x) \left[\log_2 \frac{p(x)}{\theta(x)}\right]^2 - D(p\|\theta)^2 .$$

If p is very close to uniform θ, these can be approximated using the capacity $C(p, \theta)$ [BDQ04]:

$$C(p, \theta) := \sum_{x \in \mathcal{X}} \frac{(p(x) - \theta(x))^2}{\theta(x)} \approx 2\, D(p\|\theta) \approx D_\Delta(p\|\theta) \qquad \text{(only if } p \text{ is close to } \theta.)$$

The resulting estimate for the necessary number of samples N_{LLR} to achieve a success probability $P = \mathbb{P}[\Delta_a > 0]$ can be derived as [BJV04, BGN12]:

$$N_{\mathrm{LLR}} \approx \left[\frac{\Phi_{0,1}^{-1}(P)\sqrt{D_\Delta(p\|\theta)} + \Phi_{0,1}^{-1}(\alpha)\sqrt{D_\Delta(\theta\|p)}}{D(p\|\theta) + D(\theta\|p)}\right]^2 \approx \frac{2[\Phi_{0,1}^{-1}(P) + \Phi_{0,1}^{-1}(\alpha)]^2}{C(p, \theta)} .$$

When applied to the AES fault analysis scenario, knowing $p = p_{=}$ means both knowing the exact fault distribution of the ineffective faults and guessing the corresponding 8 key bits in the penultimate round, with a correspondingly increased advantage a. In the context of differential cryptanalysis, it has been demonstrated [BGN12] that even small errors in the estimate of p can significantly increase the necessary number of samples. Since such exact models of $p_{=}$ are usually not available for practical fault attacks, we can consider less optimal, but more robust statistics.

The classical test statistic for an unknown distribution p is Pearson's χ^2:

$$S(\hat{p}) = \mathrm{CHI}(\hat{p}) := \chi^2(\hat{p}, \theta) = N \sum_{x \in \mathcal{X}} \frac{(\hat{p}(x) - \theta(x))^2}{\theta(x)} ,$$

or, for uniform θ, the closely related Squared Euclidean Imbalance (SEI):

$$S(\hat{p}) = \mathrm{SEI}(\hat{p}) := \sum_{x \in \mathcal{X}} (\hat{p}(x) - \theta(x))^2 = (N \cdot 2^b)^{-1} \cdot \mathrm{CHI}(\hat{p}) .$$

The statistic $\mathrm{CHI}(\hat{p})$ is distributed according to the (noncentral) chi-squared distribution with $k = |\mathcal{X}| - 1 = 2^b - 1$ degrees of freedom and noncentrality parameter $\lambda_{\mathrm{R}} = N\, C(p, \theta)$ or $\lambda_{\mathrm{W}} = 0$. For large k and N, this tends towards a normal distribution with parameters [HCN09, DKMO89]

$$\mathrm{CHI}(\hat{p}) \sim \begin{cases} \mathcal{N}(\mu_{\mathrm{R}} = k + NC(p, \theta), & \sigma_{\mathrm{R}}^2 = 2\,[k + 2NC(p, \theta)]) , \\ \mathcal{N}(\mu_{\mathrm{W}} = k, & \sigma_{\mathrm{W}}^2 = 2k) . \end{cases}$$

[2]Note that in this case, we could also target the last round with lower a and N, but more repetitions to obtain the full key.

Based on these parameters, we can solve the quadratic equation in (3) to estimate the necessary number of samples as [BGN12]

$$N_{\mathrm{CHI}} \approx \frac{s+\sqrt{s^2-t}}{C(p,\theta)} \qquad \left(s=\sqrt{2k}\,\Phi_{0,1}^{-1}(\alpha)+2\Phi_{0,1}^{-2}(P), \quad t=2k(\Phi_{0,1}^{-2}(\alpha)-\Phi_{0,1}^{-2}(P))\right)$$
$$= \frac{\sqrt{2k}\,\Phi_{0,1}^{-1}(\alpha)}{C(p,\theta)} \qquad \text{(for success probability } P=0.5.)$$

Summarizing, both statistics lead to an estimated number of samples that is proportional to $1/C(p,\theta)$, where the constant depends on the desired success probability P and advantage a (or quantile $\alpha = 1-2^{-a}$). However, these estimates are only useful if the resulting N is reasonably large, that is, if p is not extremely different from θ.

3.3.2 Noisy sampling: Infective countermeasure

So far, we assumed that for the correct key guess, the attacker makes the correct hypothesis $\hat{X}' = X'$, and thus directly samples the distribution $p_=(x')$. We will now show that the same approach also generalizes naturally to cases where the attacker only obtains noisy measurements.

As an example, consider the infective countermeasure with r dummy rounds. The attacker targets round $R-t$ of the $R = r+11+11$ executed AES rounds, indexed $1,\dots,R$. To identify runs with ineffective faults, she has to compare the faulted ciphertexts C' with previously obtained correct ciphertexts C for the same plaintexts P, and keeps only the samples where $C = C'$. Assuming the same fault model as before, she will keep a fraction of about $\pi_=$ samples. However, she does not know whether the ineffective fault really occurred in the penultimate AES round of the main (or, equivalently, redundant) encryption of P, or elsewhere: in a dummy round or the wrong AES round. The probability σ that the faulted round $R-t$ was a relevant round depends on the attack setup. Figure 1 illustrates the practically observed probability ($\sigma = \sigma_*$, see subsection 4.2), as well as the expected probability if the attacker can equivalently target both main and redundant rounds in the practical attack setup ($\sigma = \sigma_+$) or if she has to choose the target in advance ($\sigma = \sigma_{\max}$), where

$$\sigma = \begin{cases} \sigma_+ = \sigma_2 + \sigma_3, \\ \sigma_{\max} = \max\{\sigma_2, \sigma_3\}, \end{cases} \qquad \sigma_s = \frac{\binom{t}{s}\cdot\binom{R-t-1}{22-s-1}}{\binom{R}{22}}.$$

This probability is assumed to be independent of whether the fault was ineffective or not.

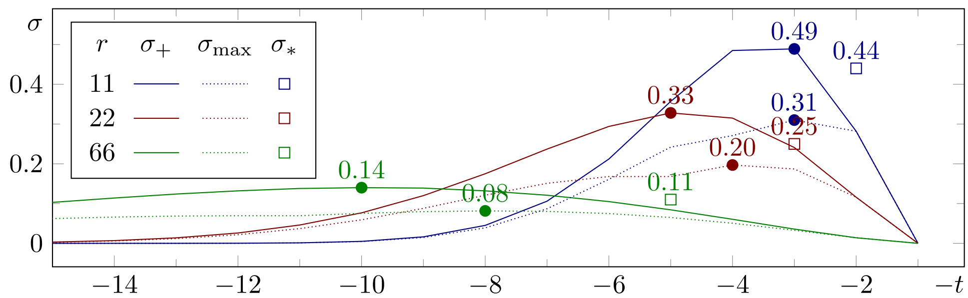


Figure 1: Probability σ of successful sampling for $r \in \{11, 22, 66\}$ dummy rounds.

Depending on whether round t was indeed relevant, the hypothesis $\hat{X}'$ for the correct key now samples one of two distributions: If t was relevant, $\hat{X}' = X'$ and we sample $p_=(x')$;

else, we sample a distribution close to uniform. Thus, we sample a noisy variable X'' with distribution $p_{\approx}(x'')$, where

$$p_{\approx}(x'') = \sigma\, p_{=}(x'') + (1-\sigma)\, 2^{-b} = \sigma\, (p_{=}(x'') - 2^{-b}) + 2^{-b}.$$

The capacity of this distribution is

$$C(p_{\approx}) = \sum_{x \in \mathcal{X}} \frac{(p_{\approx}(x) - 2^{-b})^2}{2^{-b}} = \sum_{x \in \mathcal{X}} \frac{\big(\sigma\, (p_{=}(x'') - 2^{-b})\big)^2}{2^{-b}} = \sigma^2\, C(p_{=}).$$

Thus, the expected data complexity for noisy sampling is σ^{-2} times higher compared to direct sampling.

In summary, the expected number of faults the attacker has to induce to collect enough samples is inverse proportional to $\pi_{=} \cdot \sigma^2 \cdot C(p_{=})$, where the constant depends on the desired success probability P and advantage a.

3.4 Examples and Simulations

To illustrate the statistical model in more detail, we consider a simulation of the attack with a random-and fault, i.e., each set bit of the target byte is flipped from 1 to 0 with probability $\frac{1}{2}$. The ineffectivity rate of this fault is $\pi_{=} = (3/4)^8 \approx 10\,\%$. We attack an AES implementation protected with the infective countermeasure (subsection 2.1) with $r = 22$ dummy rounds and target round $R - t = 44 - 4 = 40$, obtaining a signal of $\sigma = \frac{1111}{3526} \approx 0.315$ among the ineffectively faulted samples. The expected target distribution $p(x)$ for the correct key is illustrated together with the uniform distribution θ in Figure 2 and depends on the Hamming weight $\mathrm{hw}(x)$:

$$p(x) = \sigma \cdot 2^{8-\mathrm{hw}(x)}/3^8 + (1-\sigma) \cdot 2^{-8}.$$

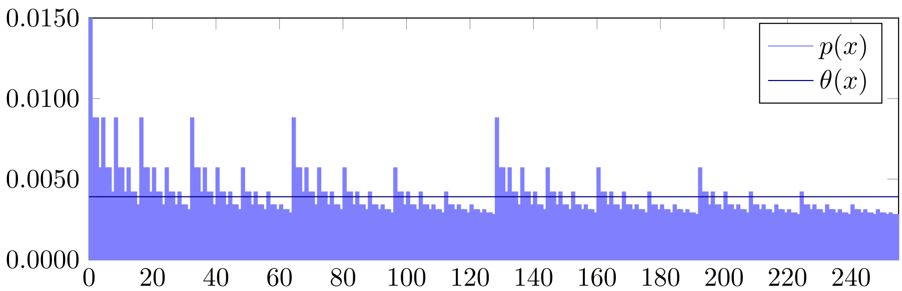


Figure 2: Distributions p and θ for random-and fault and infective countermeasure.

To compare the practically necessary number of samples N (with or without knowledge of $p(x)$) with the predictions of subsection 3.3, we evaluate the statistics $\mathrm{LLR}(\hat{p})$ and $\mathrm{CHI}(\hat{p})$ for the correct last-round key and for 2^{24} wrong key candidates (out of 2^{32}; we set one byte to the correct value).

For the $\mathrm{LLR}(\hat{p})$ statistic, we need to know the exact target distribution after addition of the penultimate round key, so we need to guess a byte K' of the penultimate round key in addition to the 24-bit key guess K. For simplicity, we evaluate each candidate K based on the statistic

$$S(\hat{p}) = \max_{K'} \mathrm{LLR}(\hat{p}, p_{K'}, \theta) = \max_{K'} N \sum_{x \in \mathcal{X}} \hat{p}(x) \cdot \log_2 \frac{p(x \oplus K')}{\theta(x)}\,.$$

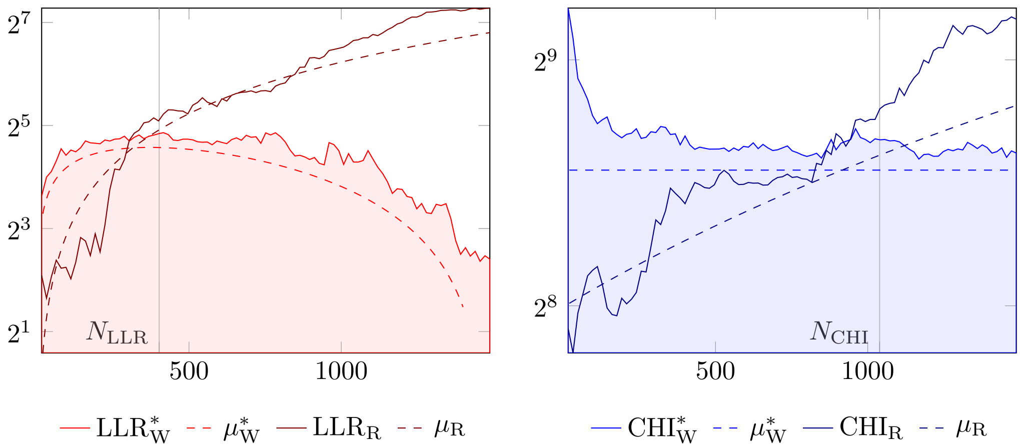
To reflect this in the model and evaluate the probability that the correct 24-bit K is ranked highest, we set the advantage to $a = \kappa + 8 = 32$, so $\alpha = 1 - 2^{-32}$. Based on the model of subsection 3.3, we expect the statistics $\mathrm{LLR_R}$ of the right key, $\mathrm{LLR_W}$ of any wrong key, and $\mathrm{LLR_W^*}$ of the best wrong key to be normally distributed with the following parameters:

$$\begin{aligned} \mu_\mathrm{R} &= ND(p\|\theta) \approx 0.075N & \sigma^2_\mathrm{R} &= ND_\Delta(p\|\theta) \approx 0.252N \\ \mu_\mathrm{W} &= -ND(\theta\|p) \approx -0.064N & \sigma^2_\mathrm{W} &= ND_\Delta(\theta\|p) \approx 0.157N \\ \mu^*_\mathrm{W} &= \mu_\mathrm{W} + \Phi^{-1}_{0,1}(\alpha)\,\sigma_\mathrm{W} \approx -0.064N + 2.469\sqrt{N} & \sigma^{2*}_\mathrm{W} &\ll \sigma^2_\mathrm{W}\,. \end{aligned}$$

For the CHI statistic, we use $a = \kappa = 24$ and expect the statistics $\mathrm{CHI_R}$, $\mathrm{CHI_W}$, and $\mathrm{CHI_W^*}$ to be normally distributed with the following parameters:

$$\begin{aligned} \mu_\mathrm{R} &= k + N\,C(p,\theta) \approx 255 + 0.131N & \sigma^2_\mathrm{R} &= 2k + 4N\,C(p,\theta) \approx 510 + 0.525N \\ \mu_\mathrm{W} &= k = 255 & \sigma^2_\mathrm{W} &= 2k = 510 \\ \mu^*_\mathrm{W} &= \mu_\mathrm{W} + \Phi^{-1}_{0,1}(\alpha)\,\sigma_\mathrm{W} \approx 375 & \sigma^{2*}_\mathrm{W} &\ll \sigma^2_\mathrm{W}\,. \end{aligned}$$

Figure 3 compares the resulting model (dashed: $\mu_\mathrm{R}, \mu^*_\mathrm{W}$) with the statistics obtained in the practical key-recovery attack (solid: $S(\hat{p}_\mathrm{R}), S(\hat{p}^*_\mathrm{W})$). The predicted necessary number of samples N for success probability $P = 0.8$ and with advantage $a = 24$ (for CHI) or $a = 32$ (for LLR) is marked as N_CHI and N_LLR, respectively. This estimate quite accurately matches the practically necessary N. It is worth noting that for both statistics, the best wrong key candidate scores slightly better than predicted with μ_W. This can be partly explained with the not-entirely-uniform distribution of the target value for partially correct key guesses, as discussed in subsection 3.3. In case of $\mathrm{CHI}(\hat{p})$, both the right and wrong keys scored slightly higher than expected.



(a) $\mathrm{LLR}(\hat{p})$ statistic

(b) $\mathrm{CHI}(\hat{p})$ statistic

Figure 3: Simulation results for infection countermeasure and random-and fault.

We repeated simulations for other fault models (bit-stuck-at-0, byte-stuck-at-0) and for both countermeasures (detection, infective). The model matches the practical results similarly well when the capacity $C(p,\theta)$ is not too large and thus N is not too small to justify the normal approximation. In particular, for the byte-stuck-at-0 fault, $C(p,\theta) \gg 1$, and the model predicts fewer than the practically necessary $N \approx 4$ ($\sigma = 1$, detection) or $N \approx 15$ ($\sigma = 0.315$, infective) samples.

In summary, having insight in the concrete effect of a fault allows to model the scores of the correct and best wrong key and accurately predict the necessary number of samples for a successful attack. In such a case, the LLR outperforms the CHI (or SEI) statistic in terms of required samples. However, for LLR, even small errors in the estimate of p can significantly increase the necessary number of samples [BGN12]. Hence, in practice, the CHI (or SEI) statistic is preferable, since the attacker can reliably and efficiently recover the key in the presence of countermeasures without any knowledge of p as demonstrated in section 4.

4 Practical Evaluation

For the practical evaluation of SIFA we have performed multiple experiments implemented on various microcontrollers listed in Table 3. First, we show the practical applicability of SIFA against a time redundant 8-bit software AES, a time redundant hardware AES co-processor, and a time redundant 32-bit bitsliced AES. We then show practical attacks against the infective countermeasure by Tupsamudre et al. [TBM14] for several security parameterizations.

Table 3: Target microcontrollers of our attack evaluation

Name	ALU Size	Core	CPU Freq.
ATXmega 256A3	8-bit	Atmel AVR	12 MHz
ATXmega 128D4	8-bit	Atmel AVR	7 MHz
STM32 F3	32-bit	ARM Cortex-M4	7 MHz

Fault Setup. In order to induce the faults we have used clock glitches. To be more precise, we xor an additional fast clock edge on the original clock signal to violate the critical path. By additionally varying width and offset of the induced clock edge, it is possible to influence the fault induction success rate and its impact on the faulted instruction. We have used an FPGA for generating both the original clock signal and the clock glitch for the device under test. For sake of simplicity, we determined our attack parameters with the help of an unprotected implementation in the case of the detection-based countermeasure. In case that an unprotected implementation is not accessible to an attacker, determining the fault parameters is much more time consuming, but still feasible. We want to point out that the demonstrated attacks do not require a profiling of the actual distribution of the induced fault. In fact, we performed the key recovery attacks without any knowledge about the distribution of the targeted byte.

All experiments are performed in a fully automated attack setup. By using this setup, we are able to perform about 20 faulted encryptions per second, or 72 000 per hour. The time required to collect enough correct ciphertexts for key recovery is somewhere between 1 minute and 2 hours.

Possible Effects of Clock Glitches. We first give some intuition explaining the possible scenarios that can occur when using clock glitches for fault induction on various platforms. Later, when discussing the individual attack results, we will refer to these scenarios to help explain our results. The following three scenarios can arise:

1. *Missed Fault.* Occurs if the timing of the induced fast clock edge is incorrect and can occur in two situations. (1) The induced clock edge is too short and the microprocessor never sees a logical '1' on the clock signal. (2) The induced clock edge is too long and the microprocessor is able to execute the current instruction correctly.

2. *Successful and Effective Fault.* Occurs if the induced fast clock edge influences the target instruction/register in such a way that the outcome of the whole computation is affected.

3. *Successful but Ineffective Fault.* Occurs if the induced fast clock edge influences the target instruction/register but no effect on the outcome of the whole computation is observed.

Generally speaking, traditional fault attacks exploit *Successful and Effective Faults*, while IFA, and SIFA exploit *Successful but Ineffective Faults.* Depending on the attacked implementation and the quality of laboratory equipment, the occurrence of *Missed Faults* varies. For an attacker, it is usually not possible to distinguish between a *Missed Faults* and a *Successful but Ineffective Fault*, since both have no effect on ciphertexts. In fact even if we know the key in our attacks, we cannot reliably distinguish between those two cases. However, distinguishing between both cases is not needed in the attack.

Properties of (Ineffective) Biased Faults. Besides the scenarios discussed before, also the actual properties of an *Successful but Ineffective Faults* on a certain instruction/register determines the efficiency of SIFA combined with the occurrence of *Missed Faults.* Hence, we define the following two properties:

1. *Bias.* The resulting bias of a certain intermediate value/byte defines its distance from a uniform random distribution, when observed over multiple encryptions. In the context of SIFA, we measure this distance via the Squared Euclidean Imbalance (SEI). The bias that we observe and exploit in SIFA stems from the combination of *Missed Faults* and *Successful but Ineffective Faults.*

2. *Fault Ineffectivity Rate.* This rate determines how often there is no effect on the computational outcome after fault induction:

$$\frac{\#(\textit{Successful but Ineffective Faults}) + \#(\textit{Missed Faults})}{\#(\textit{Successful and Effective Faults})}$$

The performance of a fault attack is often measured in the number of required faulted encryptions. An ideal fault for SIFA would cause a strong bias, have a high *Fault Ineffectivity Rate*, and would never miss. Such ideal faults are difficult to achieve in practice, since the requirements are somewhat contradictory. When considering the common fault models the stuck-at fault on bit-level is a good candidate for SIFA, since it has a very high *Fault Ineffectivity Rate* of 50% while still causing a decent bias. Higher rates are possible if, e.g., *Missed Faults* occur but this is not desired.

In our practical evaluation of SIFA, we deal with faults that are not optimal for SIFA. On some platforms we observe strong biases but in combination with very low *Fault Ineffectivity Rates.* On other platforms we observe high *Fault Ineffectivity Rates* but weak biases. Nevertheless, we are able to perform practical attacks on various platforms with different countermeasures in place. This demonstrates the versatility of SIFA and biased faults in general. At the end of each practical experiment we shortly discuss our findings regarding fault scenarios in combination with the fault properties described above.

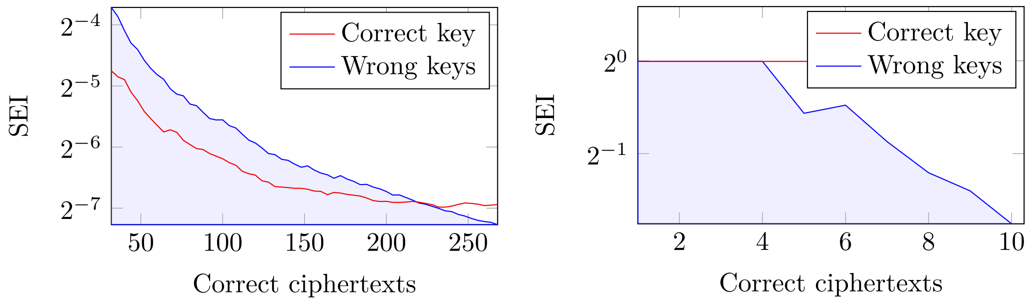
4.1 Attacks on Detection-based Countermeasure

We first target a detection-based countermeasure that uses simple time redundancy with subsequent comparison (Algorithm 1). Here, the encryption is executed twice and only if the results of both encryptions are identical, the ciphertext is returned. Note that our attack is just as effective in case more than two redundant executions are performed. We evaluated our attack both for pure software AES implementations and AES co-processor implementations. The attacks against software AES were evaluated using an 8-bit register-based AES implementation on the ATXmega 256A3, and a 32-bit-bitsliced AES on the STM32 F3. The attack against the hardware co-processor AES was performed on the ATXmega 256A3.

4.1.1 8-bit Software AES

We used the AES implementation from the AVRCryptoLib [avr] as a basis for our protected implementation and performed experiments on both a ATXmega 256A3 and a ATXmega 128D4. Our attacks target the output of the S-box calculation in round 9, and we only induce a fault in one of the two AES encryptions.

The results presented in Figure 4 show the number of correct ciphertexts (≈ 220 and 5) needed until the correct 4-byte key candidate has the highest SEI and thus can be reliably distinguished. In both cases, roughly 1 000 faulted encryptions were necessary to collect the required amount of unaffected and correct ciphertexts. From these results we can see that our fault inductions had quite different effects on both types of microprocessors. On the ATXmega 128D4 platform we are able to induce reliable faults that affect single instructions/bytes. Here, the *Fault Ineffectivity Rate* of 0.39% is very low but the induced bias is strong. In contrast to that, the *Fault Ineffectivity Rate* of 34% is comparably high on the ATXmega 256A3 platform. Here, we suspect that (1) the induced fault affects individual bits of a byte only with a certain probability, (2) the resulting induced bias is rather weak, and (3) *Missed Faults* might occur.



(a) ATXmega 256A3: 220 correct ciphertexts stemming from about **1 000** faulted encryptions are required.

(b) ATXmega 128D4: 5 correct ciphertexts stemming from about **1 300** faulted encryptions are required.

Figure 4: Attacks on 8-bit software AES, detection countermeasure. SEI of the correct key ($\mathrm{SEI_R}$) vs. best SEI for a wrong key ($\mathrm{SEI_W^*}$) for N correct encryptions.

4.1.2 32-bit-bitsliced Software AES on STM32 F3

In order to evaluate our attack for bitsliced AES implementations, we have used the constant-time bitsliced implementation by Schwabe et al. [SS16]. The attack setup itself is similar to the one in the previous section.

About 22 000 correct ciphertexts were required to reliably recover 4 bytes of the AES key, as shown in Figure 5. In total we have performed 130 000 faulted AES encryptions and received about 26 000 correct ciphertexts. Hence, the *Fault Ineffectivity Rate* was 20 % in this setting. Again, we do expect a mixture of weak biases and *Missed Faults*. Please note that this result is meant as a proof of concept rather than a concrete performance estimation of SIFA against 32-bit platforms. With a more precise fault setup, e.g., a laser fault induction setup, we expect significantly better results.

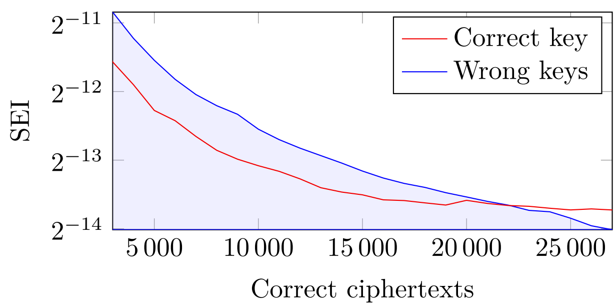


Figure 5: Attacks on 32-bit bitsliced SW AES, STM32 F3, detection countermeasure. SEI of the correct key ($\mathrm{SEI_R}$) vs. best SEI for a wrong key ($\mathrm{SEI_W^*}$) for N correct encryptions. 22 000 correct ciphertexts were required, stemming from about **130 000** faulted encryptions.

4.1.3 Hardware Co-Processor AES on ATXmega 256A3

In our attack against the integrated AES co-processor on the ATXmega 256A3, approximately 550 correct ciphertexts were required for recovering 4 bytes of the AES key as shown in Figure 6. In total about 800 faulted encryptions were required, the *Fault Ineffectivity Rate* is hence about 69 %. As explained before, such a high rate is only possible if *Missed Faults* occur. We strongly expect that the former is the case, maybe in combination with a weak bias.

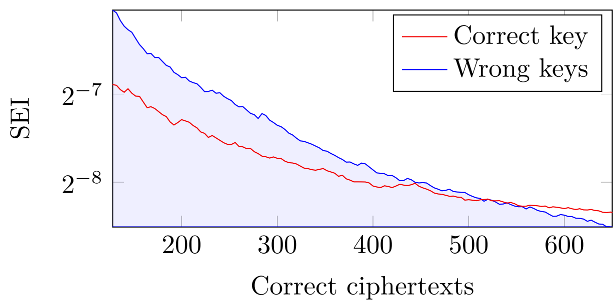


Figure 6: Attacks on HW AES co-processor, ATXmega 256A3, detection countermeasure. SEI of the correct key ($\mathrm{SEI_R}$) vs. best SEI of a wrong key ($\mathrm{SEI_W^*}$) for N correct encryptions. 550 correct ciphertexts were required, stemming from about **800** faulted encryptions.

4.2 Attacks on Infective Countermeasures

We evaluated our attack on the infective countermeasure by Tupsamudre et al. [TBM14] from CHES 2014 (Algorithm 2). Since the hardware co-processor of the ATXmega 256A3 only computes one complete call of AES, we limit this attack evaluation to purely software-based implementations on the ATXmega 128D4.

We extended the AES implementation from the AVRCryptoLib [avr] according to Algorithm 2. The implementation of the AES round functions itself was not modified. Since the authors in [TBM14] did not give any recommendations for t, we have evaluated our attack for $t \in [11, 22, 66]$, leading to AES encryptions that require 33, 44, and 88 AES round function calls, respectively.

We started with a simulation of multiple encryption runs in order to determine when a penultimate, non-dummy AES round is performed with highest probability. Clearly, the best time for the attack depends on t. According to the simulation results in Table 4 we hit a penultimate, non-dummy round with highest probability when targeting the 31st, 41st, and 83rd round, respectively. Once we know the best round for the attack, we can use a similar fault parameterisation as in the other experiments with the ATXmega 128D4. In contrast to the detection-based scenario, we cannot detect ineffective faults by observing just one encryption when infection is used. Hence, we always perform one encryption twice, one with fault induction, and one without.

Table 4: Occurrence of dummy round hits for infective AES

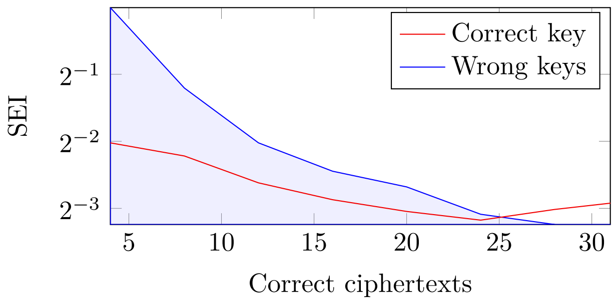
Dummy Rounds t	Total AES Rounds	Target Round	Correct Round Hit Probability
11	33	31	44 %
22	44	41	25 %
66	88	83	11 %

As mentioned earlier the efficiency of SIFA is, among others, determined by the bias of the induced fault, the fault's *ineffectivity rate*, and thus might vary between experiments. In order to allow for an easier comparison between the results for various t we only show practical results where the *ineffectivity rate* and the strength of the bias are similar. Both properties can be roughly estimated by an attacker after successful key recovery.

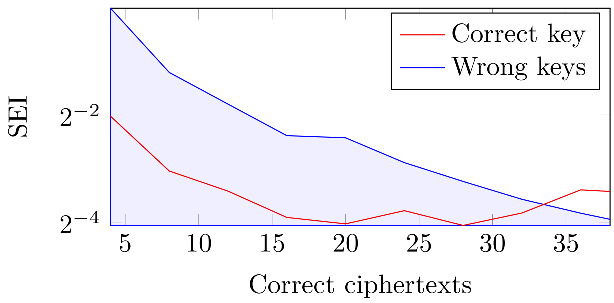
Our attack evaluation was performed for a variable number of dummy rounds with $t \in [11, 22, 66]$, the results are shown in Figure 7. In our practical evaluation on the ATXmega 128D4 platform, and depending on t, 6 500, 9 000, 46 000 faulted encryptions were necessary to gather the 25, 34, 180 correct ciphertexts that allowed us to recover 4 key bytes. Here we conclude that the bias of the induced fault was strong, the *ineffectivity rate* was low, and the number of *Missed Faults* was also low.

Compared to the analysis in subsubsection 3.3.2 and Table 4, the observed increase in the necessary number of ciphertexts roughly matches the predictions, in particular from $t = 22$ to $t = 66$ (increase roughly $\times 5.9$, predicted $\times 5.2$). The observed increase compared to $t = 11$ (roughly $\times 1.4$ to $t = 22$ and $\times 8.0$ to $t = 66$) is less than predicted using the the measured probabilities from Table 4 (predicted $\times 3.1$ and $\times 16.0$, respectively). This may indicate that the probabilities are closer to the theoretical estimates σ_+ from Figure 1 (predicted $\times 1.4$ and $\times 11.0$, respectively) and/or a relatively high number of ciphertexts required in the specific experiment for $t = 11$. The latter is very likely because the estimates and normal approximations of subsection 3.3 assume a relatively low capacity and are not accurate for high capacities and small N, as we have for $t = 11$.

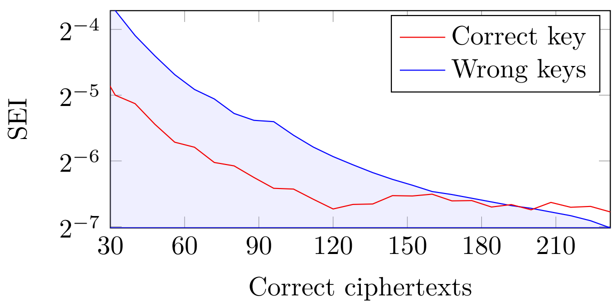
The small available number of samples is not sufficient to derive a detailed fault model (for the sake of comparing our results in more detail with the theoretical model; of course, we do not require the model to perform the attack). However, based on the available data for $t = 22$ and $t = 66$, we can make an educated guess at the effects of our fault setup. The fault ineffectivity rate is very close to 1/256, as we would expect from a fault that affects a whole byte and a very small number of missed faults, as discussed above. The observed distribution among the ineffective faults during the key recovery phase also suggests a noisy stuck-at distribution, with a signal σ less strong than expected for the infection countermeasure. This may indicate that if the stuck-at fault hits the correct round, it hits the correct byte (correct fault effect) in a fraction around half or three quarters of the cases. This model is also a good fit to explain the necessary number of correct ciphertexts for $t = 22$ and $t = 66$ (for $t = 11$, the number of samples is too low to make any useful statements): For example, for $t = 66$ based on $N = 237$ collected samples, assuming $\sigma \approx 0.065$ (see Figure 1, and close to $0.5 \cdot 0.11$ from Table 4 as assumed above), we would expect about 16.5 stuck-at hits (observed: 18). Using advantage $a = 32$ and target success probability $p = 0.8$, the model would predict that we need roughly 160 ciphertexts, which is only slightly less than we actually needed (Figure 7c). For the other cases, similar models also slightly underestimate the necessary number of ciphertexts.



(a) ATXmega 128D4, **t=11**. 25 correct ciphertexts were required, stemming from about **6 500** faulted encryptions.



(b) ATXmega 128D4, **t=22**. 34 correct ciphertexts were required, stemming from about **9 000** faulted encryptions.



(c) ATXmega 128D4, **t=66**. 180 correct ciphertexts were required, stemming from about **46 000** faulted encryptions.

Figure 7: Attacks on infective countermeasure. SEI of the correct key ($\mathrm{SEI_R}$) vs. best SEI of a wrong key ($\mathrm{SEI_W^*}$) for N correct encryptions.

5 Discussion of other Implementation Countermeasures

We have demonstrated the effectiveness of SIFA on two different countermeasures using various platforms in the previous section. Nevertheless, more countermeasures exist and we discuss the impact of some of them on SIFA in this section. Overall, SIFA seems to be a powerful attack vector and so far, the main point we can say regarding countermeasures is that more noise, e.g., by using dummy rounds increases the attack complexity.

5.1 Infection by Patranabis et al.

The infection countermeasure by Patranabis et al. [PCM15] can be seen as an extended version of the one described in subsubsection 2.1.2. Hence, we limit our description solely to the actual differences between both designs. The extended countermeasure aims at tackling a shortcoming of the previous design that allowed successful attacks if an attacker is able to alter the control flow or force precise instruction skips. To mitigate this attack vector two adaptations were proposed.

First, an additional randomized string *cstr* is introduced that raises the uncertainty in the execution order of cipher state R_0 and redundant state R_1 in each round. *cstr* is of length $2t$ and is made up of t 2- bit tuples $[x_i, y_i]$, each of which has either the value $[0, 1]$ or $[1, 0]$. While the execution order of R_0 and R_1 within one round was fixed in the previous design, *cstr* can now be used to additionally shuffle their execution.

Second, temporary masking is introduced that hides R_0 and R_1 at the end of an odd round and reveals them at the beginning of the corresponding even round. This has the effect that neither R_0 nor R_1 expose the output of the previous round after an odd round.

In SIFA, every time we receive a correct ciphertext stemming from an ineffective fault induction on an AES with correct key, we can reduce the number of key candidates. Since both R_0 and R_1 use the correct key, their execution order within one round is irrelevant for SIFA. The additional temporary masking of states does not affect SIFA either, since all round function calls still work with the original states. Hence, we expect SIFA to perform against the extended version of infection as well as against the version by Tupsamudre et al. [TBM14].

5.2 Fault Space Transformation

Fault Space Transformation (FST), is a novel fault countermeasure proposed by Patranabis et al. [PCMC17]. This countermeasure works with two redundant states R_0 and R_1, similar as a detection-based countermeasure that uses redundancy with subsequent comparison. So the encryption is performed on R_0 and R_1, but under a special linear encoding $R_1 = W(R_0)$ such that it is difficult to induce similar faults in both states.

While there are many possible choices for the linear encoding W, the authors propose the usage of the AES-MixColumns function. This choice of W has the beneficial side effect that a one-byte fault in one state is mapped to a 4-byte fault in the other state and vice versa. This linear dependency between R_0 and R_1 increases the difficulty of inducing two equivalent faults or the exploitation of two biased faults up to a point where they can be considered infeasible in practice. The threat of both DFA as well as DFIA is hence prevented. For a more detailed description of FST we refer to the original paper [PCMC17].

However, SIFA solely relies on observing whether a fault induction in one AES state (either R_0 and R_1) is ineffective. Since the state R_0 is calculated without linear encoding, an attacker, who is able to only fault the branch calculating R_0 can expect the same attack complexities as on an ordinary detection-based countermeasures (subsection 4.1). Faulting the encoded state would work too, the observed bias would be different but still be as strong as in the non-encoded state. Like mentioned before, in SIFA the existence of a bias is sufficient for key recovery. The exact distribution of the bias does not need to be known by the attacker. Hence, SIFA performs against FST as well as shown in the attacks against AES with detection-based countermeasure.

5.3 Majority Voting

SIFA relies on detecting whether an induced fault is ineffective. A complete mitigation of this attack vector would require a compensation of any fault induction such that every observed encryption is correct. One technique that attempts to do this is Majority Voting

(MV) to select which ciphertext is returned. In MV the same computation is performed n times, where n is odd and ≥ 3. The final output is then defined as the majority over all computed outputs. It depends on the implementations what happens if all ciphertexts differ. Here, we assume that in this case no ciphertext is returned. Such a scheme would prevent SIFA, using single faults per encryption, since one faulty ciphertext is always "overruled" by at least two correct ciphertexts.

In the case of SIFA against MV with $n = 3$ one can simply perform one ordinary targeted 9^{th} round biased fault induction in one computation and any random fault induction in one of the other computations. That way a correct ciphertext has the majority if the biased fault induction is ineffective. For this implementation of the countermeasure, we expect that an attack works with a similar complexity as for detection-based countermeasures, but requiring two faults per execution.

Note that different implementations of a majority voting are possible. For instance, a majority voting on bit level is possible. If $n = 3$, and at least two computations are erroneous, an attacker will get an erroneous ciphertext returned. Still, SIFA is possible as described before, however, an attacker now needs additional computations in order to identify erroneous ciphertexts.

5.4 Masking

Masking is a widely-deployed countermeasure against side-channel attacks, where the secret intermediate values are split into d-shares. On the first glance, masking prevents a direct application of SIFA. One way to apply SIFA on masking is to use multiple faults on all shares of a single intermediate variable. While using only single faults for their proposed attack, using faults on multiple locations is a strategy already proposed by Clavier [Cla07] to apply IFA on masked implementations. As a simple example consider an attacker, who is able to induce a fault that sets one value more likely to 0. If this fault is applied on all shares of the same variable, also the unshared value will likely be biased and hence, SIFA will work.

A straight-forward application scenario for SIFA against masking are bitsliced implementations that work on all shares concurrently [JS17]. Here, one biased fault is likely to cause a joint non-uniform distribution over all shares that can be exploited by an attacker in the exact same way as described in this paper. However, SIFA against masking is not restricted to these implementations, or multiple faults. In particular, recent follow-up work demonstrates that SIFA is applicable on masked implementations of cryptographic primitives with fault countermeasures using single faults [DEG+18].

6 Conclusion

In this work, we provide an extensive insight on ineffective faults, where faults are being induced, but not showing an effect. The introduced statistical ineffective fault attacks (SIFA) can be seen as an intersection of the principles exploited by ineffective fault attacks (IFA) [Cla07] and by statistical fault attacks (SFA) [FJLT13]. While previous work on IFA relies on strong models like stuck-at faults, we were able to relax these conditions up to a point were we only require that intermediate values follow an unknown but non-uniform distribution in cases where fault inductions have been ineffective. Hence, no special fault profiling of a targeted device is necessary.

SIFA inherits the ability from IFA that it only exploits the output of valid computations, which makes the attack independent of the degree of redundancy used in a countermeasure. As a consequence, it is not harder to attack a detection-based countermeasure performing 16 redundant operations compared to a countermeasure just performing 2. On the other hand, like SFA, SIFA works with minimal assumptions on the effect of the faults. Thus, similar as it has been shown for SFA (e.g., in [DEK+16]), we are able to demonstrate the feasibility of SIFA on various platforms in practice. However, in contrast to SFA, the practical attacks with SIFA are possible even in the presence of countermeasures against fault attacks.

We demonstrate the improvements of our work over IFA, amongst others by showing the applicability of SIFA on detection-based countermeasures utilizing 32-bit-bitsliced software AES implementations, or hardware co-processor AES implementations. In both cases the induction of precise stuck-at faults in certain bytes, as required by IFA, is considerably harder and was not possible in our fault setup.

Ultimately, we show that SIFA has new applications where neither SFA nor IFA are applicable by demonstrating attacks against an infective AES countermeasure presented at CHES 2014 [TBM14]. Here, SFA does not work, since the induced errors are amplified up to a point where the faulty output is unexploitable. Also IFA is not possible in this case, because even if precise faults were feasible, IFA cannot deal with the large amount of noise resulting from a low fault induction success rate that is caused by the presence of dummy rounds at random points in time.

Acknowledgments

This work has been supported in part by the Austrian Science Fund (project P26494-N15), the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402), the European Union's Horizon 2020 research and innovation programme under grant agreement No 644052 (HECTOR), and the Austrian Research Promotion Agency (FFG) via the K-project DeSSnet, which is funded in the context of COMET – Competence Centers for Excellent Technologies by BMVIT, BMWFW, Styria and Carinthia.

References

[AMT13] Subidh Ali, Debdeep Mukhopadhyay, and Michael Tunstall. Differential fault analysis of AES: towards reaching its limits. *J. Cryptographic Engineering*, 3(2):73–97, 2013.

[avr] AVR crypto lib. `https://git.cryptolib.org/?p=avr-crypto-lib.git`.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *LNCS*, pages 37–51. Springer, 1997.

[BDQ04] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On multiple linear approximations. In Matthew K. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *LNCS*, pages 1–22. Springer, 2004.

[BECN+06] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[BG16] Alberto Battistello and Christophe Giraud. A note on the security of CHES 2014 symmetric infective countermeasure. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design – COSADE 2016*, volume 9689 of *LNCS*, pages 144–159. Springer, 2016.

[BGN12] Céline Blondeau, Benoît Gérard, and Kaisa Nyberg. Multiple differential cryptanalysis using LLR and χ^2 statistics. In Ivan Visconti and Roberto De Prisco, editors, *Security and Cryptography for Networks – SCN 2012*, volume 7485 of *LNCS*, pages 343–360. Springer, 2012.

[BJV04] Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Joong Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 432–450. Springer, 2004.

[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.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (2. ed.)*. Wiley, 2006.

[CW13] Christophe Clavier and Antoine Wurcker. Reverse engineering of a secret AES-like cipher by ineffective fault analysis. In Wieland Fischer and Jörn-Marc Schmidt, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2013*, pages 119–128. IEEE Computer Society, 2013.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Hannes Gross, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. Cryptology ePrint Archive, 2018. https://eprint.iacr.org/2018/357.

[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.

[DKMO89] Feike C. Drost, Wilbert C. M. Kallenberg, D. S. Moore, and J. Oosterhoff. Power approximations to multinomial tests of fit. *Journal of the American Statistical Association*, 84(405):130–141, 1989.

[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.

[GST12] Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology – LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 305–321. Springer, 2012.

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

[HCN09] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional extension of Matsui's Algorithm 2. In Orr Dunkelman, editor, *Fast Software Encryption – FSE 2009*, volume 5665 of *LNCS*, pages 209–227. Springer, 2009.

[HS13] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *Smart Card Research and Advanced Applications – CARDIS 2013*, volume 8419 of *LNCS*, pages 219–235. Springer, 2013.

[JS17] Anthony Journault and François-Xavier Standaert. Very high order masking: Efficient implementation and security evaluation. Cryptology ePrint Archive, 2017. https://eprint.iacr.org/2017/637.

[LSG+10] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault sensitivity analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *LNCS*, pages 320–334. Springer, 2010.

[NP33] Jerzy Neyman and Egon S. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Trans. of the Royal Society of London*, pages 289–337, 1933.

[PCM15] Sikhar Patranabis, Abhishek Chakraborty, and Debdeep Mukhopadhyay. Fault tolerant infective countermeasure for AES. In Rajat Subhra Chakraborty, Peter Schwabe, and Jon A. Solworth, editors, *Security, Privacy, and Applied Cryptography Engineering – SPACE 2015*, volume 9354 of *LNCS*, pages 190–209. Springer, 2015.

[PCMC17] Sikhar Patranabis, Abhishek Chakraborty, Debdeep Mukhopadhyay, and P. P. Chakrabarti. Fault space transformation: A generic approach to counter differential fault analysis and differential fault intensity analysis on AES-like block ciphers. *IEEE Transactions on Information Forensics and Security*, 12(5):1092–1102, 2017.

[PCNM15] Sikhar Patranabis, Abhishek Chakraborty, Phuong Ha Nguyen, and Debdeep Mukhopadhyay. A biased fault attack on the time redundancy countermeasure for AES. In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design – COSADE 2015*, volume 9064 of *LNCS*, pages 189–203. Springer, 2015.

[SBHS15] Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise laser fault injections into 90 nm and 45 nm SRAM-cells. In *Smart Card Research and Advanced Applications – CARDIS 2015*, volume 9514 of *LNCS*, pages 193–205. Springer, 2015.

[Sel08] Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, 21(1):131–147, 2008.

[SS16] Peter Schwabe and Ko Stoffelen. All the AES you need on Cortex-M3 and M4. In Roberto Avanzi and Howard M. Heys, editors, *Selected Areas in Cryptography - SAC 2016*, volume 10532 of *LNCS*, pages 180–194. 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.

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