Practical Improvements to Statistical Ineffective Fault Attacks

Barış Ege[1], Bob Swinkels[1], Dilara Toprakhisar*[2], and Praveen Kumar Vadnala[1]

[1] Riscure B.V., Delft, The Netherlands
lastname@riscure.com
[2] COSIC, KU Leuven, Leuven, Belgium,
dilara.toprakhisar@esat.kuleuven.be

Abstract. Statistical Fault Attacks (SFA), introduced by Fuhr *et al.*, exploit the statistical bias resulting from injected faults. Unlike prior fault analysis attacks, which require both faulty and correct ciphertexts under the same key, SFA leverages only faulty ciphertexts. In CHES 2018, more powerful attacks called Statistical Ineffective Fault Attacks (SIFA) have been proposed. In contrast to the previous fault attacks that utilize faulty ciphertexts, SIFA exploits the distribution of the intermediate values leading to fault-free ciphertexts. As a result, the SIFA attacks were shown to be effective even in the presence of widely used fault injection countermeasures based on detection and infection. In this work, we build upon the core idea of SIFA, and provide two main practical improvements over the previously proposed analysis methods. Firstly, we show how to perform SIFA from the input side, which in contrast to the original SIFA, requires injecting faults in the earlier rounds of an encryption or decryption operation. If we consider the start of the operation as the trigger for fault injection, the cumulative jitter in the first few rounds of a cipher is much lower than the last rounds. Hence, performing the attack in the first or second round requires a narrower parameter range for fault injection and hence less fault injection attempts to recover the secret key. Secondly, in comparison to the straightforward SIFA approach of guessing 32-bits at a time, we propose a chosen input approach that reduces the guessing effort to 16-bits at a time. This decreases the key search space for full key recovery of an AES-128 implementation from 2^{34} to 2^{19}.

Keywords: Fault attacks · SIFA · AES · Chosen plaintext attack

1 Introduction

Since the seminal work of Boneh et al. [7], which introduced fault attacks on RSA, numerous publications highlighted the susceptibility of the implementations of cryptographic algorithms to such active attacks exploiting their physical characteristics. Fault attacks involve deliberately injecting faults during the

* Corresponding author.

execution of a cryptographic algorithm through physical means, followed by an analysis of the reaction of the device under attack. These faults can be injected through various methods, including voltage/clock glitching [2], temperature manipulation [22], white light [29], electromagnetic waves [16], or laser injections [21,31,3,23,12,1], as well as software-based faults [30,26].

Since Biham and Shamir [5] have extended Differential Fault Analysis (DFA), the work of Boneh *et al.*, on symmetric key algorithms, numerous techniques improving DFA have been published [18,6,20,24,25,27,14,4]. In general, DFA-like analysis techniques retrieve the secret key by utilizing the characteristics of the induced fault and the faulty ciphertexts, which use the value of the faulty and the corresponding correct ciphertexts to retrieve the key. In a different vein, Fuhr *et al.* [19] introduced Statistical Fault Attacks (SFA), exploiting the statistical distributions of the targeted intermediate values derived from the faulty ciphertexts. In contrast to DFA-like attacks, SFA only requires access to the faulty ciphertexts, and exploits the non-uniform distribution of a state byte value caused by the fault injection.

DFA and SFA involve modifying the value of an intermediate variable during the computation, and exploiting the faulty output. However, Ineffective Fault Analysis (IFA), as introduced by Clavier *et al.*[13], takes a different approach. IFA exploits the fault-free ciphertexts by using ineffective faults to probe the intermediate value. In other words, IFA exploits the cases where the injected fault does not affect the output, utilizing only correct ciphertexts. Building upon the core ideas of SFA and IFA, Dobraunig *et al.* proposed Statistical Ineffective Fault Attacks (SIFA)[17] in CHES 2018. SIFA exploits the non-uniform distributions of the intermediate values targeted by ineffective faults derived from correct ciphertexts. SIFA is akin to IFA in that both exploit faults that do not alter the output. As with SFA, they both exploit the bias in the statistical distribution of an intermediate value. Notably, SIFA exploiting the correct ciphertexts circumvents simple redundancy based countermeasures that protect against SFA and DFA-like attacks. Moreover, it does not require an adversary to know the injected fault, enhancing its practical applicability. In this regard, SIFA has more relaxed requirements compared to IFA, and it can utilize various fault models creating bias in the targeted variable. However, this attack requires analysis of 2^{32} key candidates to perform the analysis on AES, which imposes practical limitations especially when a large number of ciphertexts need to be analyzed per key candidate. When analyzing an AES-128 implementation for instance, this leads to a total of 2^{34} analysis steps for full key recovery. In this work, we adopt the chosen input approach in SIFA as an attempt to decrease the computational complexity of the SIFA analysis.

Contributions. First, we show how to perform SIFA from the input side. This involves injecting biased faults early in the encryption or decryption operation, and analyzing the distribution of the targeted intermediate value computed from the collected plaintexts corresponding to fault-free ciphertexts, where the injected fault is ineffective. Performing SIFA from the input side has an advantage in practice due to less jitter introduced. As the cumulative jitter in the first few

rounds is expected to be lower than the last few rounds, this attack has the advantage of narrower fault injection parameter range, thus, less number of fault injection attempts to recover the secret key.

Secondly, we propose a chosen-input SIFA, which requires only two key bytes to be guessed, as opposed to the four byte key guesses in the original attack, decreasing the size of the key search space to 2^{16} from 2^{32}. This reduces the total number of key guesses from 2^{34} to 2^{19} when applied to AES-128.

Outline. In Section 2, we discuss SFA, IFA, and SIFA which our work is based on. Then, in Section 3, we describe our first contribution, performing SIFA from the input side. We first present the attack description, then, the results of the simulations evaluating the presented attack. In Section 4, we present the attack description of the chosen input SIFA, and the simulation results. Then, we evaluate both attacks in practice, and present the practical experiment results in Section 5. Finally, in Section 6, we discuss the impacts of the proposed techniques that improve SIFA in practice, and pose open questions regarding future work.

2 Preliminaries

In this section, we recall Statistical Fault Attacks (SFA) and Ineffective Fault Attacks (IFA). Next, we describe how both these ideas were combined and extended in Statistical Ineffective Fault Attacks (SIFA).

2.1 Statistical Fault Attacks

SFA was introduced by Fuhr *et al.* [19] as a technique to recover the secret key in AES. SFA exploits the bias in the distribution of an intermediate value obtained from the faulty ciphertexts. For symmetric key algorithms, the intermediate values computed during the execution are expected to be uniformly distributed. However, if the distribution of intermediates changes due to the injected faults, then it is possible to exploit these biases to recover the secret key. To perform SFA, an attacker needs to collect faulty ciphertexts and follow an appropriate key recovery strategy depending on the targeted round. The same idea applies to all key recovery strategies: the attacker is required to decrypt the faulty ciphertexts back to the intermediate value targeted by the induced fault with the key hypotheses, and apply a distinguisher to recover the secret key.

The working principle of SFA depends on changing an intermediate value and obtaining faulty ciphertexts. Therefore, SFA can be prevented with the existing countermeasures as the output is suppressed (in case of detection) or randomized (in case of infection) in the presence of a fault.

2.2 Ineffective Fault Attacks

IFA [13] uses fault injection as a probing tool on the targeted intermediate value by comparing the value of this variable with the expected value (*i.e.,* when no

fault was injected). These values are equal when the fault has no effect on the targeted intermediate value. The goal here is to find a fault model that does not cause a change in the intermediate value and yields the correct output. Therefore, detection-based countermeasures are not effective against IFA as they work only with the existence of faulty ciphertexts.

IFA works as follows: the attacker induces a stuck-at-x fault to an intermediate value in one execution where the other execution is fault-free. If the attacker receives the same output regardless of the induced fault, the targeted intermediate value must have been x already. The secret key can be recovered by performing the cipher operations in reverse up to the targeted operation. Correct guess of the corresponding byte of the key should lead to the value x in the intermediate value. However, one of the problems of this attack is determining if the fault was actually successful or not. Moreover, in practice, stuck-at faults occur less frequently compared to other faults, *e.g.,* bit-flips.

2.3 Statistical Ineffective Fault Attacks

SIFA [17] was proposed as a novel fault attack technique that works under detection-based and infective countermeasures. It extends the ideas of SFA and IFA so as to overcome the limitations of both. It exploits the bias in the distribution of an intermediate value that is targeted by fault injection leading to a fault-free ciphertext.

For the SIFA attack to be successful, the only requirement is the existence of a bias in the distribution of the intermediate value. The probability of changing an intermediate value by a fault is not the same for all possible values of a byte. Table 1 shows the non-uniformity in the probability distribution when a random- and fault model, where each bit has a 50% probability of being reset, is applied to two-bit values. When the fault is injected, the possibility of an intermediate value 00 staying the same is 1, whereas the value 11 staying the same is 1/4. The bias in this probability distribution makes a cryptographic implementation susceptible to SIFA.

Table 1: Fault distribution table for random-and fault model

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

In the original paper, the faults were injected to one byte before the last MixColumns operation. Then, the distribution of the targeted byte value ob-

tained from collected fault-free ciphertexts will be non-uniformly distributed. This non-uniformity can be exploited by using a key revealing technique similar to the one used in SFA [19] to recover four bytes of the last round key. To obtain the partial state S_9, the collected ciphertexts need to be partially decrypted:

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

Next, the distribution of the partial state S_9 obtained from the collected ciphertexts is evaluated by computing χ^2 statistic (or SEI) for each key hypothesis (using the formula in Equation 2). If enough successful faults were injected around the target operation, the key hypothesis leading to the highest χ^2 statistic will be the correct key.

$$\chi^2(\hat{p}, \theta) = N \sum_{x \in \mathcal{X}} \frac{(\hat{p_k}(x) - \theta(x))^2}{\theta(x)} \tag{2}$$

Here $\theta(x) = 1/256$ is the probability distribution of the byte values in the uniform distribution, and $\hat{p_k}(x)$ is the probability distribution of the byte values in the observed distribution using the key hypothesis k.

This attack allows an attacker to exploit any ineffective fault that causes a non-uniform distribution in the targeted value. The SIFA is robust against dummy rounds or unsuccessful fault injections and the attacker does not necessarily need to know the distribution of the injected faults.

If the AES implementation is protected with detection-based countermeasures, only the fault-free ciphertexts will be collected. In case of infective countermeasures, the fault-free ciphertexts need to be filtered by performing encryption or decryption operations without injecting fault, comparing the results, and keeping the ciphertexts that are same as the non-faulty ones.

3 Performing SIFA on the Inputs

In this section, we describe how to perform SIFA from the input side on AES. Fundamentally, the fault analysis strategy remains the same as the original attack. However, in this approach, faults are injected very early in the AES encryption/decryption. Subsequently, the fault analysis is performed on the inputs corresponding to the fault-free outputs. Our simulations demonstrate that bias introduced to the distribution of the targeted intermediate value can be exploited, which facilitates the key recovery when the attack is executed from the input side. Moreover, with the increasing cumulative jitter during the encryption or decryption operation, injecting faults in the early rounds demands fewer fault injection attempts, thanks to the narrower parameter range. We first outline the assumptions related to the capabilities of the adversary performing the fault injection. Subsequently, we describe how the injected fault is exploited. Then, we present the simulation results of the attack.

Adversarial Model. The adversary is capable of injecting a biased fault with precise control on the timing of the injection (*i.e.,* targets a specific operation). Moreover, the adversary has some control over the location of the injected fault, *i.e.,* allowing the injection of the fault affecting any set of bits within the target byte value.

Attack Description. Considering the encryption operation, to perform SIFA on the plaintexts, an adversary with the described capabilities injects a fault between the first and second MixColumns operations. This differs from the original SIFA approach, which requires a fault to be injected before the last MixColumns operation. We assume an adversary injects a fault after the second SubBytes operation as shown in Figure 1. As a result of the fault injection, the adversary obtains a set of filtered plaintext-ciphertext pairs under the detection-based countermeasures, where the targeted intermediate value after the first MixColumns exhibits a non-uniform distribution.

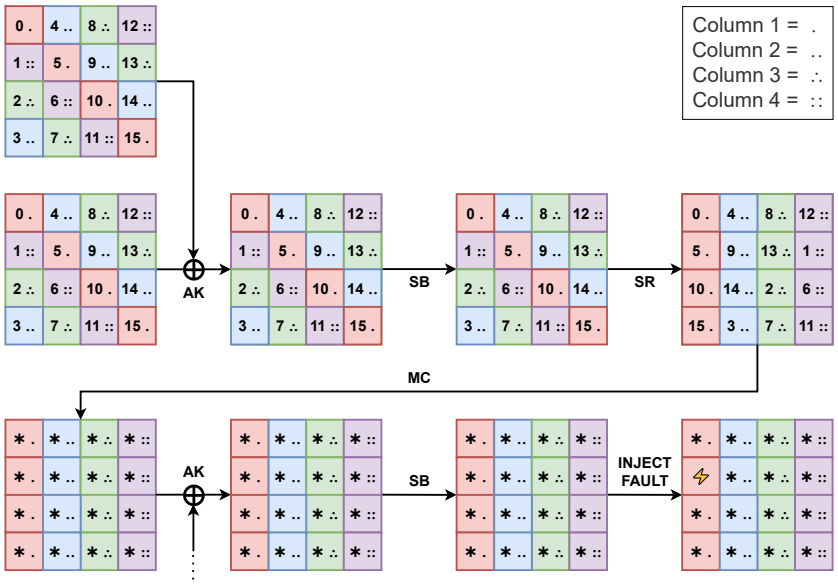


Fig. 1: Sketch of the attack on round 2 using a fault injection after the second SubBytes operation.

The key recovery also follows the same strategy as the original attack. Specifically, the adversary guesses four bytes of the first round key, aligning with the bytes of the plaintext that influence the distribution of the targeted intermediate value (*i.e.*, bytes in Column 1 in Figure 1). Subsequently, the adversary partially computes the first and second round operations of AES encryption for

each plaintext corresponding to a correct ciphertext, resulting in the derivation of the partial state denoted as S_1:

$$S_1 = ((PT \oplus K_0) \circ SB \circ SR \circ MC \tag{3}$$

Then, for each key guess, the adversary analyzes the distribution of the targeted byte within the state S_1, and computes an χ^2 statistic. The highest χ^2 statistic value serves as the indicator of the correct key guess.

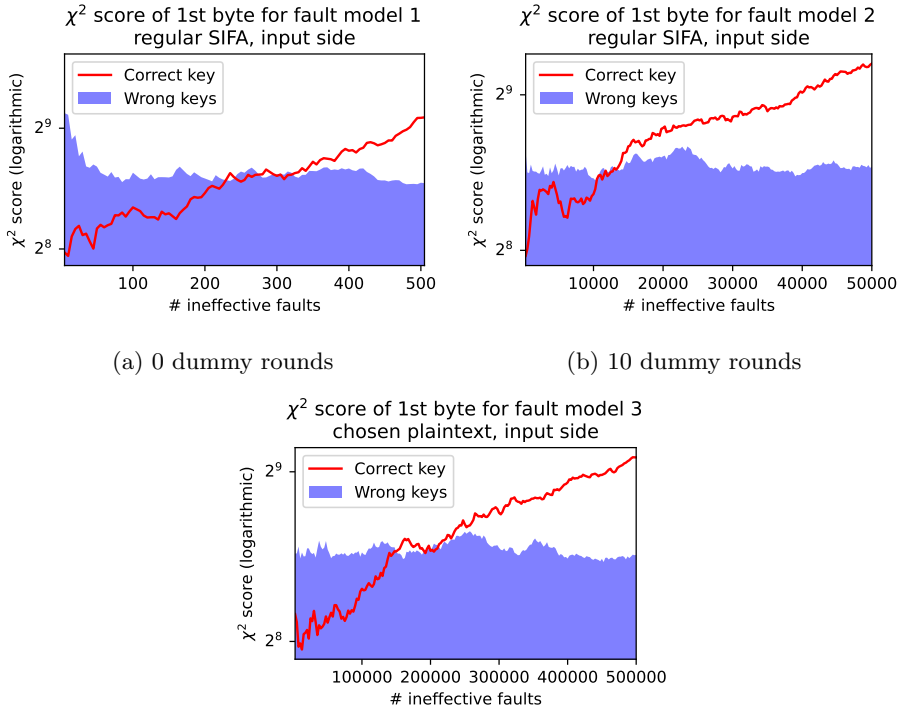
Practical Benefits. For AES implementations with clock-jitter, clock randomization countermeasures, or random delays, it is beneficial to target an operation that is performed early in the encryption/decryption operation. This is because the cumulative jitter during the encryption or decryption operation is smaller, which improves the precision of the fault injection. Moreover, the number of fault injection attempts needed to inject a fault repeatedly at the same location is reduced.

For instance, consider the clock randomization countermeasure proposed in [8], where the randomized clock implementation can generate pulses of at least 403 different frequencies ($n = 403$) with one fixed base frequency ($m = 1$). When attacking the 2nd round ($r = 2$) rather than the 9th round ($r = 9$), of an AES encryption operation that takes 10 clock cycles to complete, the number of different times to complete the operation gets reduced from ${}_{r+n-1}C_r \cdot m = {}_{9+403-1}C_9 \cdot 1 \approx 8.44 \cdot 10^{17}$ to ${}_{r+n-1}C_r \cdot m = {}_{2+403-1}C_2 \cdot 1 = 81.4 \cdot 10^3$. This substantial reduction minimizes the number of fault injection attempts needed to repeatedly inject a fault that targets the same operation.

Simulations. To simulate the attack from the input side, a four-bit random-AND fault model is used. In this model, each of the four least significant bits in the targeted byte has a probability of 50% to be reset. In the simulations, an AES implementation protected with a detection countermeasure was used which implements various degrees of dummy rounds; namely no dummy rounds, 10 dummy rounds and 40 dummy rounds.

When attacking an implementation with no dummy rounds, four key bytes can be recovered by collecting approximately 325 plaintexts corresponding to correct ciphertexts (*i.e.,* ineffective faults), as shown in Figure 2a. When 10 dummy rounds are used, four key bytes can be recovered by collecting approximately 13,500 plaintexts corresponding to correct ciphertexts, as shown in Figure 2b. When 40 dummy rounds are used, four key bytes can be recovered by collecting approximately 212,500 plaintexts corresponding to correct ciphertexts, as shown in Figure 2c.

These simulations show that it is possible to perform SIFA from the input side on AES. The number of traces required to recover the secret key is comparable to the number of traces required to perform SIFA from the output side. Note that the actual benefit of our approach is seen in practical evaluation, when there is clock jitter during the execution as explained above.



(a) 0 dummy rounds

(b) 10 dummy rounds

(c) 40 dummy rounds

Fig. 2: Comparison of χ^2 score for actual key and wrong keys with detection countermeasure

4 Chosen Input SIFA

In this section, we describe the working principle of the chosen input SIFA. Essentially, the attack follows the process described in Section 3, except with a notable difference that, in this attack, the inputs are explicitly chosen by the adversary. In addition to the adversarial capabilities detailed in Section 3, the adversary has the capability to feed the encryption or decryption algorithm with inputs of their choosing.

Attack Description. We consider the encryption operation to describe the attack. Similar to the described attack from the input side, the adversary injects a fault between the first and second MixColumns operations, where we assume a byte after the second SubBytes is targeted as depicted in Figure 1. However, in this attack, two out of the four bytes of the plaintext inputs that influence the distribution of the targeted byte are held constant, while the other two bytes are generated randomly. Then, instead of attempting to guess four bytes of the key corresponding to these four bytes of the plaintext at a time, the adversary guesses

only two bytes of the key corresponding to the random bytes of the plaintext. Subsequently, the adversary applies the first round operations for the chosen plaintexts corresponding to a correct ciphertext, resulting in the partial state S_1 as described in Equation 3. Then, the same key recovery strategy described in Section 3 is followed to determine the correct key. This improvement reduces the key search space to 2^{16} (*i.e.,* two bytes) from 2^{32} (*i.e.,* four bytes) for a single guess, utilizing an equal or fewer number of traces depending on the fault capabilities of the fault injection adversary.

It's essential to note that utilizing a statistical test-based distinguisher (*e.g.,* SEI) does not allow reducing the search space for a single guess to 2^8. Achieving a size of 2^8 may be possible by holding only one of the four bytes influencing the distribution of the targeted byte as random, and only guessing the corresponding key byte, while keeping the others constant. Among the operations shown in Equation 3 that the adversary applies to obtain the partial state S_1, only the MixColumns operation changes the probability distribution of the targeted byte. That is, until the MixColumns operation, the random plaintext byte retains a random value, while the other constant bytes retain constant values across a number of chosen plaintexts. Let r denote this random byte value. Then, the MixColumn operation applied to this state results in a state where r becomes $r + c$ with c being a constant. The operations applied after MixColumns, likewise, do not change the probability distribution of the targeted byte, making the distribution obtained by the key guesses indistinguishable. Hence, it is not possible to reduce the search space further.

Attack method. By using a methodical approach, the above described analysis can be performed on all columns simultaneously. This is desirable in practice since the specific column targeted by the fault injection is often unknown beforehand. To achieve simultaneous analysis of all columns, we generate the crafted plaintext inputs in two steps as follows:

1. Attack the key bytes with even indices (Figure 3).
2. Attack the key bytes with odd indices (Figure 4).

First, every odd byte of plaintext input is set to a fixed value, *e.g.,* zero, while keeping the other bytes random. Then, a fault is injected after the second SubBytes operation during the encryption operation. The crafted plaintexts corresponding to correct ciphertexts are collected (*i.e.,* when the injected fault is ineffective). As shown in Figure 3, for the key recovery, if a key hypothesis for the first column outranks the other key guesses by a large margin, it indicates the correct guess for the key bytes with the indices 0 and 10. Then, if a key hypothesis for the second, third, and the last column, outranks the other guesses by a large margin, it also indicates the correct guess for the key bytes with indices 4 and 14, 2 and 8, and 6 and 12, respectively.

This process is then repeated for the second step where every even byte of the plaintext input is set to a fixed value. As shown in Figure 4, for the key recovery, if a key hypothesis for the first column outranks the other key guesses

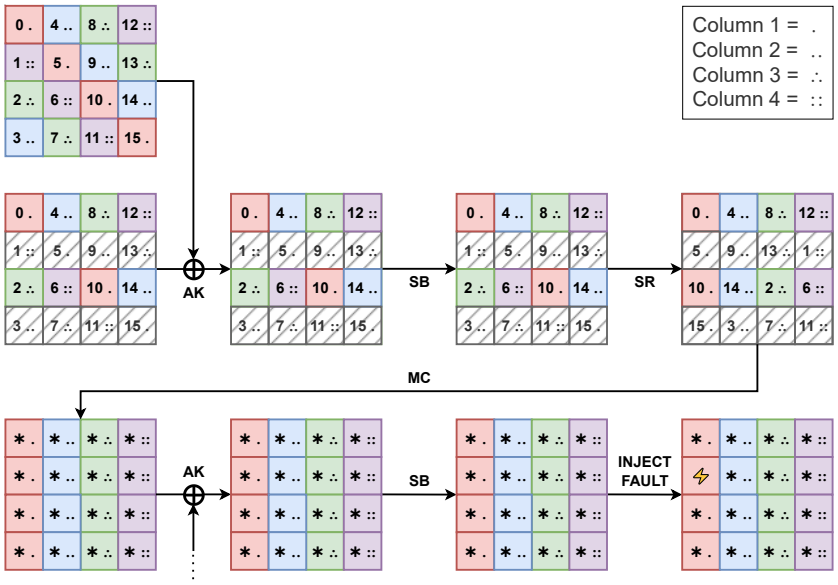


Fig. 3: Sketch of the chosen plaintext attack on round 1 using a fault injection after the second substitute bytes operation and every odd byte set to zero

by a large margin, it indicates the correct guess for the key bytes with the indices 5 and 15. Then, if a key hypothesis for the second, third, and the last column, outranks the other guesses by a large margin, it indicates the correct guess for the key bytes with indices 3 and 9, 7 and 13, and 1 and 11, respectively.

If no key hypothesis significantly outranks the other key guesses, it indicates that key recovery is not possible. This can be attributed to factors such as insufficient fault injection attempts, the fault model not being effective enough, or unsuccessful fault injection.

Simulations. To simulate the chosen input SIFA, a four-bit random-AND fault model is used. In this model, each of the four least significant bits of the targeted byte has a probability of 50% to be reset. Similar to the simulations of SIFA from the input side, an AES implementation protected with a detection countermeasure was used which implements various degrees of dummy rounds; namely no dummy rounds, 10 dummy rounds, and 40 dummy rounds.

When attacking an implementation with no dummy rounds, two key bytes can be recovered by collecting approximately 235 chosen plaintexts corresponding to correct ciphertexts, as shown in Figure 5a. When 10 dummy rounds are used, two key bytes can be recovered by collecting approximately 25,750 chosen plaintexts corresponding to correct ciphertexts, as shown in Figure 5b. When 40 dummy rounds are used, two key bytes can be recovered by collecting approximately 240,000 chosen plaintexts corresponding to correct ciphertexts, as shown in Figure 5c.

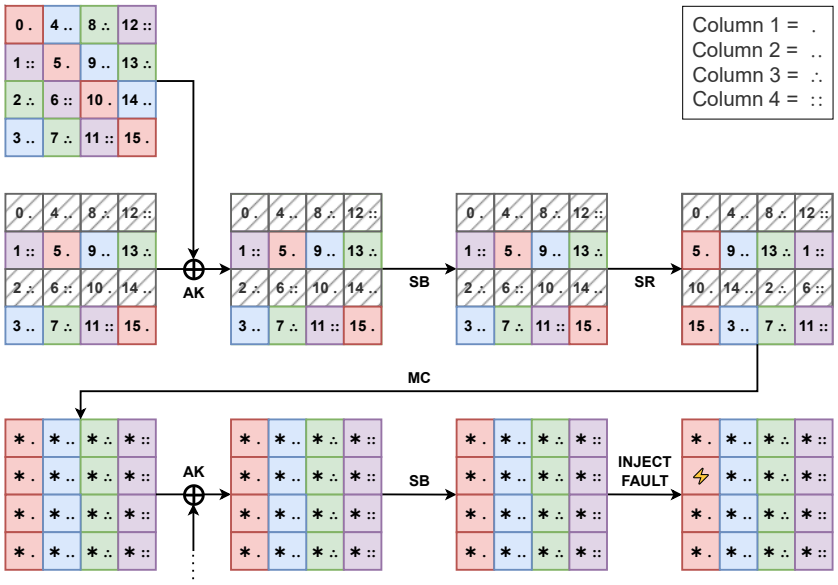


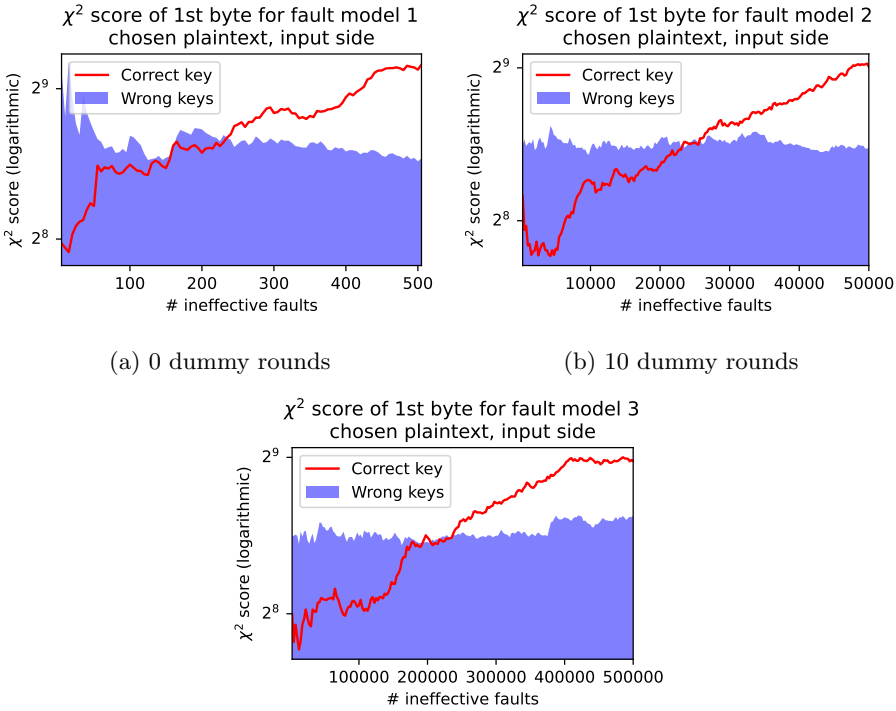
Fig. 4: Sketch of the chosen plaintext attack on round 1 using a fault injection after the second substitute bytes operation and every even byte set to zero

These simulations show that it is possible to perform the chosen input SIFA on AES. The number of traces required to recover the secret key is comparable to the number of traces required to perform SIFA from the input side. However the search space to recover four key bytes is decreased by a factor of $2^{15} = 32\,768$, therefore the brute force attack takes significantly less time.

5 Practical Evaluation

SIFA from the input side and the chosen input SIFA were evaluated using an 8-bit software AES implementation (as described in Section 4.1 of [15]) and a 32-bit t-table software AES implementation (as described in Section 4.2 of [15]) running on an STM32F407IG Arm Cortex-M4 core. For this, a Piñata development board [10] was used, which has been physically modified by removing the decoupling capacitors to make it more susceptible to voltage glitches. The board is shown in Figure 6.

Fault Setup. To inject faults into the target device, voltage glitches were used. For this, a device called Spider [11] was used to control the glitch timing and the glitch voltage, and a device called Glitch Amplifier [9] was used to amplify the glitches and inject the fault into the target device. Communication with the target device was realized through the use of an FTDI FT232RL USB to serial UART interface. To determine a range for the parameters, side channel profiling was performed. The parameters used for the voltage glitching experiments are shown in Table 2.



(a) 0 dummy rounds

(b) 10 dummy rounds

(c) 40 dummy rounds

Fig. 5: Comparison of χ^2 score for actual key and wrong keys with detection countermeasure

Four different experiments were carried out evaluating the SIFA from the input side and the chosen input SIFA on both a textbook software AES and a t-table software AES implementation. For the experiments performing SIFA from the input side, the χ^2 distributions of the correct key and 100,000 wrong keys were compared. For the experiments performing chosen input SIFA, the χ^2 distributions of the correct key and all possible $2^{16} - 1$ wrong keys were compared. Analysis was performed on a Debian system with an AMD Ryzen Threadripper 3970X 32-Core processor and 256GB RAM. Using a pure python implementation we can iterate over 135 key candidates per second. This leads to an estimated running time of about a year for SIFA from the input side and about 16 minutes for chosen input SIFA to recover 32 bits of the secret key.

In the experiment evaluating SIFA from the input side on a textbook AES implementation, 76% of the glitches were successful. After 8278 glitches, 1150 ineffective faults were observed, and the χ^2 score of the correct key exceeded the χ^2 score of the wrong keys, as shown in Figure 7a. Similarly, when attacking the t-table implementation with SIFA from the input side, 73% of the glitches were

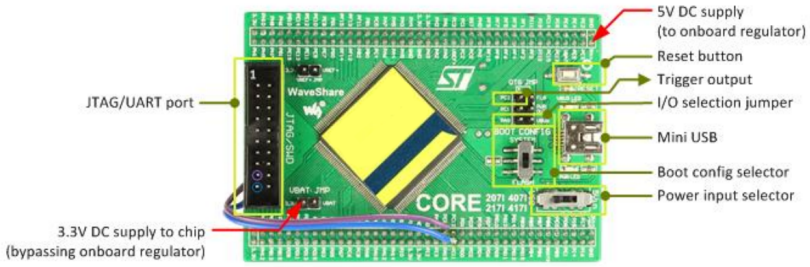


Fig. 6: Piñata board

Table 2: Parameters used in the experiments

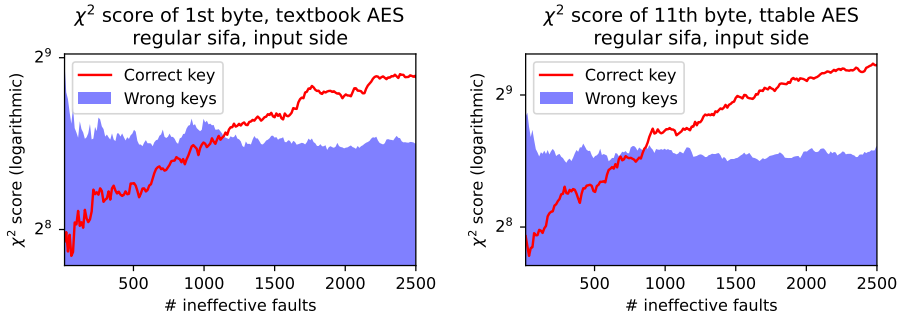
	Input side SIFA		Chosen Input SIFA	
Parameters	**Textbook**	**T-Table**	**Textbook**	**T-Table**
Normal voltage	3.3 V	3.3 V	3.3 V	3.3 V
Glitch voltage	1.0 V	1.0 V	1.0 V	1.0 V
Glitch length	123 ns	123 ns	123 ns	123 ns
Glitch delay	32500 ns	5550 ns	32500 ns	5550 ns

successful. After 4366 glitches, 865 ineffective faults were observed, and the χ^2 score of the correct key exceeded the χ^2 score of the wrong keys, as shown in Figure 7b. Even though these results are quite similar to the results obtained from the simulations, the number of fault injection attempts needed to recover the key is higher than the number of attempts needed in the simulations. This is due to the jitter that occurs when performing fault injection in practice.

In the experiment evaluating the chosen plaintext attack on a textbook AES implementation, the successful fault rate was 89%. After 17706 glitches, 1085 ineffective faults were observed, and the χ^2 score of the correct key exceeded the χ^2 score of the wrong keys, as shown in Figure 8a. Similarly, when attacking the t-table implementation with the chosen plaintext attack, 65% of the glitches were successful. After 5390 glitches, 1310 ineffective faults were observed, and the χ^2 score of the correct key exceeded the χ^2 score of the wrong keys, as shown in Figure 8b. Similar to the experiments evaluating SIFA from the input side, the number of attempts needed to recover the key was higher than the number of attempts needed in the simulations due to the jitter occurring in practice.

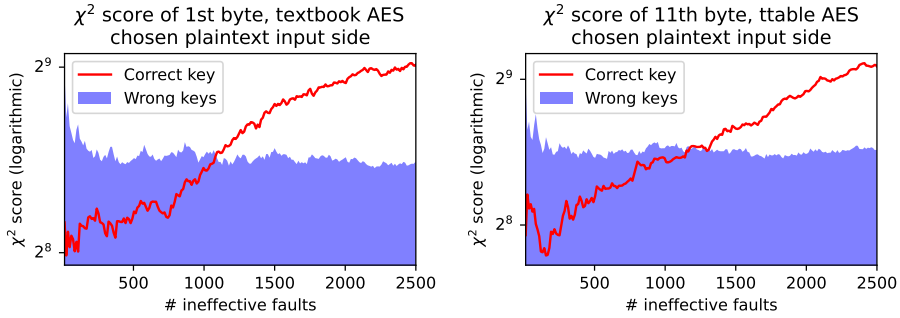
6 Discussion & Conclusion

In this work, we provided techniques to improve SIFA in practice. First, we showed with the help of both simulations as well as practical experiments that the SIFA attack can also be performed in the first rounds of a cipher. A successful



(a) textbook AES (b) t-table AES

Fig. 7: Evaluation of SIFA from input side



(a) textbook AES (b) t-table AES

Fig. 8: Evaluation of chosen plaintext SIFA

fault injection attack greatly depends on choosing the right parameters among which the exact time to inject fault is very critical. This is typically measured from the time of the trigger (*e.g.,* start of a cryptographic operation) and is chosen randomly from a range around the target operation. The range of values to be chosen from greatly impacts the number of fault injection attempts required to get a successful fault. The bigger the range, the higher the number of required attempts. This range depends on the time jitter from the trigger point to the target operation. This jitter could be natural or artificially induced, *e.g.,* due to a countermeasure. Naturally, the farther the operation is from the trigger, the higher is the jitter and hence the range for glitch offset is also higher. As a result, the ability to inject faults in the first few rounds of a cipher operation leads to a better success rate in practice than injecting in the last few rounds as we discussed in the paper.

Secondly, the SIFA attack requires brute force on a part of the key, which is 32 bits in case of AES. For recovering a full AES-128 key, we needed to perform this 4 times which leads to brute force complexity of 2^{34}. We showed that by using chosen inputs we can reduce the key search space to 16-bits a time. This is only possible because, as we showed in this paper, the SIFA works also when we inject faults in the first round. As a result, it requires 2^{16} key guess for a 16-bit subkey and improves the overall complexity to 2^{19} for full key recovery of AES-128. Using this technique, we could reduce the estimated attack time to recover 32 bits of the secret key from around a year to about 16 minutes. Note that this implementation was not fully optimized for performance but gives an indication of the speed up that can be achieved even for optimized implementations.

One promising avenue for future research involves investigating the applicability of the Statistical Ineffective Fault Injection (SIFA) methods to other symmetric cryptographic implementations. While our current work has primarily focused on AES, it is crucial to assess the effectiveness and adaptability of SIFA across a broader spectrum of symmetric ciphers. This could involve examining different algorithms with varying design principles, key sizes, or modes of operation.

While our current study has shown that SIFA can be performed using voltage glitching on certain software implementations of AES, an important direction for future work is to reproduce the practical results with AES hardware implementations. By extending our investigation to AES hardware implementations, we can deepen our understanding of the robustness of SIFA across different platforms.

Furthermore, a valuable area for future investigation lies in enhancing the input-side and chosen plaintext attack methods through the utilization of advanced statistical fault analysis techniques, such as Fault Intensity Map Analysis (FIMA) [28], instead of SIFA. This approach holds the potential to further diminish the number of required traces for key retrieval, thereby enhancing the efficiency and practicality of the proposed attacks.

Acknowledgements This work was partially supported by CyberSecurity Research Flanders with reference number VR20192203.

References

1. Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *16th IEEE International On-Line Testing Symposium (IOLTS 2010), 5-7 July, 2010, Corfu, Greece*, pages 235–239. IEEE Computer Society, 2010.
2. Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1997.

3. Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *IACR Cryptol. ePrint Arch.*, 2004:100, 2004.
4. Eli Biham, Louis Granboulan, and Phong Q. Nguyen. Impossible fault analysis of RC4 and differential fault analysis of RC4. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 359–367. Springer, 2005.
5. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.
6. Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). In Rebecca N. Wright, editor, *Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003, Revised Papers*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.
7. 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, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.
8. Martin Brisfors, Michail Moraitis, and Elena Dubrova. Do not rely on clock randomization: A side-channel attack on a protected hardware implementation of aes. In Guy-Vincent Jourdan, Laurent Mounier, Carlisle Adams, Florence Sèdes, and Joaquin Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 38–53, Cham, 2023. Springer Nature Switzerland.
9. Riscure B.V. Glitch amplifier. `https://getquote.riscure.com/en/quote/2101070/glitch-amplifier.htm`. Accessed: 2023-12-01.
10. Riscure B.V. Pinata S (software crypto). `https://getquote.riscure.com/en/quote/2101127/pinata-s-software-crypto.htm`. Accessed: 2023-12-01.
11. Riscure B.V. Spider. `https://getquote.riscure.com/en/quote/2492015/spider.htm`. Accessed: 2023-12-01.
12. Gaetan Canivet, Paolo Maistri, Régis Leveugle, Jessy Clédière, Florent Valette, and Marc Renaudin. Glitch and laser fault attacks onto a secure AES implementation on a sram-based FPGA. *J. Cryptol.*, 24(2):247–268, 2011.
13. 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, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.
14. Nicolas T. Courtois, David Ware, and Keith M. Jackson. Fault-algebraic attacks on inner rounds of des. In *The eSmart 2010 European Smart Card Security Conference*, 2010.
15. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
16. Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic transient faults injection on a hardware and a software implementations

of AES. In Guido Bertoni and Benedikt Gierlichs, editors, *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*, pages 7–15. IEEE Computer Society, 2012.

17. Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):547–572, 2018.
18. Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on A.E.S. *IACR Cryptol. ePrint Arch.*, 2003:10, 2003.
19. 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, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 108–118. IEEE Computer Society, 2013.
20. Christophe Giraud. DFA on AES. *IACR Cryptol. ePrint Arch.*, 2003:8, 2003.
21. D. H. Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *IEEE Transactions on Nuclear Science*, 12(5):91–100, 1965.
22. Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. *IACR Cryptol. ePrint Arch.*, 2014:190, 2014.
23. Oliver Kömmerling and Markus G. Kuhn. Design principles for tamper-resistant smartcard processors. In Scott B. Guthery and Peter Honeyman, editors, *Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999*. USENIX Association, 1999.
24. Yang Li, Shigeto Gomisawa, Kazuo Sakiyama, and Kazuo Ohta. An information theoretic perspective on the differential fault analysis against AES. *IACR Cryptol. ePrint Arch.*, 2010:32, 2010.
25. Amir Moradi, Mohammad T. Manzuri Shalmani, and Mahmoud Salmasizadeh. A generalized method of differential fault attack against AES cryptosystem. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 91–100. Springer, 2006.
26. Kit Murdock, David F. Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against intel SGX. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1466–1482. IEEE, 2020.
27. Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.
28. Keyvan Ramezanpour, Paul Ampadu, and William Diehl. Fima: Fault intensity map analysis. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 63–79, Cham, 2019. Springer International Publishing.
29. Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic*

Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

30. Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. CLKSCREW: exposing the perils of security-oblivious energy management. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 1057–1074. USENIX Association, 2017.
31. Fan Zhang, Yiran Zhang, Huilong Jiang, Xiang Zhu, Shivam Bhasin, Xinjie Zhao, Zhe Liu, Dawu Gu, and Kui Ren. Persistent fault attack in practice. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):172–195, 2020.