Statistical Fault Attacks on Nonce-Based Authenticated Encryption Schemes

Christoph Dobraunig[1], Maria Eichlseder[1], Thomas Korak[1], Victor Lomné[2], and Florian Mendel[1](✉)

[1] Graz University of Technology, Graz, Austria
florian.mendel@iaik.tugraz.at
[2] ANSSI, Paris, France
victor.lomne@ssi.gouv.fr

Abstract. Since the first demonstration of fault attacks by Boneh et al. on RSA, a multitude of fault attack techniques on various cryptosystems have been proposed. Most of these techniques, like Differential Fault Analysis, Safe Error Attacks, and Collision Fault Analysis, have the requirement to process two inputs that are either identical or related, in order to generate pairs of correct/faulty ciphertexts. However, when targeting authenticated encryption schemes, this is in practice usually precluded by the unique nonce required by most of these schemes.

In this work, we present the first practical fault attacks on several nonce-based authenticated encryption modes for AES. This includes attacks on the ISO/IEC standards GCM, CCM, EAX, and OCB, as well as several second-round candidates of the ongoing CAESAR competition. All attacks are based on the Statistical Fault Attacks by Fuhr et al., which use a biased fault model and just operate on collections of faulty ciphertexts. Hereby, we put effort in reducing the assumptions made regarding the capabilities of an attacker as much as possible. In the attacks, we only assume that we are able to influence some byte (or a larger structure) of the internal AES state before the last application of MixColumns, so that the value of this byte is afterwards non-uniformly distributed.

In order to show the practical relevance of Statistical Fault Attacks and for evaluating our assumptions on the capabilities of an attacker, we perform several fault-injection experiments targeting real hardware. For instance, laser fault injections targeting an AES co-processor of a smartcard microcontroller, which is used to implement modes like GCM or CCM, show that 4 bytes (resp. all 16 bytes) of the last round key can be revealed with a small number of faulty ciphertexts.

Keywords: Fault attacks · Authenticated encryption · CAESAR · Differential Fault Attacks (DFA) · Statistical Fault Attacks (SFA)

1 Introduction

Fault attacks pose a serious threat for cryptographic implementations. For this kind of attacks, the analyzed device is operated outside its defined operating

© International Association for Cryptologic Research 2016
J.H. Cheon and T. Takagi (Eds.): ASIACRYPT 2016, Part I, LNCS 10031, pp. 369–395, 2016.
DOI: 10.1007/978-3-662-53887-6_14

conditions, which can lead to erroneous outputs. By analyzing the erroneous output data, secret information can be revealed. In the worst case, a single fault can reveal the entire secret key of a block cipher like AES, which has been shown to be feasible by many researchers in the last decade [7,33]. Popular techniques to inject faults include modifications of the power supply [50] or the clock source [6] by injecting glitches. Other methods, such as laser fault injection [45], have been proven even more powerful, because they additionally allow a precise localization of the fault injection.

While fault attacks on block ciphers and stream ciphers have received a great deal of attention from the scientific community, authenticated ciphers have been arguably less popular targets among researchers. At the same time, they describe an important class of cryptographic algorithms with many applications in information security. Authenticated encryption provides both confidentiality and authentication of data to two parties communicating via an insecure channel. This is essential for many applications such as SSL/TLS, IPSEC, SSH, or hard-disk encryption. In most applications, there is not much value in keeping the data secret without ensuring that it has not been intentionally or unintentionally modified. For this reason, in practical applications, block ciphers like AES are typically used mainly as a building block for an authenticated encryption scheme.

An authenticated encryption scheme is usually modeled as a function with four inputs: a unique nonce N, associated data A, plaintext P, and secret key K. It generates two outputs: the ciphertext C, and the authentication tag T:

$$\mathcal{E}(K, N, A, P) = (C, T).$$

The corresponding decryption algorithm takes the secret key K, nonce N, authenticated data A, ciphertext C, and tag T, and either outputs the plaintext P if the verification tag is correct, or $\perp$ if the verification of the tag failed:

$$\mathcal{D}(K, N, A, C, T) \in \{P, \perp\}.$$

It is usually assumed (and typically essential for the security of the authenticated encryption scheme) that nonces never repeat for encryptions $\mathcal{E}$ under the same key K. We refer to such schemes as nonce-based authenticated encryption. While some schemes claim a certain level of robustness even in misuse settings (such as repeated nonces, or release of unverified plaintext), this does not mean that they are intended to be intentionally misused in practical implementations: repeating nonces always incurs a certain loss of security.

An interesting consequence of the unique nonce in the encryption procedure is the implicitly provided protection against several classes of fault attacks [11, 12,49]. In particular, Differential Fault Analysis (DFA) [11] is rendered almost impossible, since an attacker is unable to observe both the correct and the faulty output for the same input, if the attacker cannot fix the value of the nonce. Moreover, in contrast to nonce-based (but unauthenticated) encryption schemes (such as CBC, CTR, etc.), where the decryption procedure (with a fixed nonce) is still susceptible to DFA, this is not the case for nonce-based authenticated encryption schemes that only return the plaintext if the tag is correct. For this reason, all published fault attacks on authenticated encryption schemes so far are in settings where either the nonce is repeated, or unverified plaintext is released [42,43].

These observations might lead to the impression that nonce-based authenticated encryption schemes are not susceptible to fault attacks and thus, no dedicated fault attack countermeasures might be necessary to protect the implemented scheme against these attacks. However, in this work, we show that this assumption is not true, and present the first fault attacks on authenticated encryption schemes that are not performed in some kind of misuse scenario. We show that countermeasures against fault attacks are essential for implementations of authenticated encryption schemes operating in hostile environments.

Our Contribution. We present fault attacks for a wide range of authenticated encryption schemes. Our attacks do not require any misuse scenario, such as nonce reuse or release of unverified plaintext. We focus our discussion on various AES-based schemes, including the ISO/IEC standards CCM [48], GCM [32], EAX [9], and OCB [40], as well as several second-round CAESAR [46] candidates. However, our analysis is applicable to a broader range of constructions and is not limited to AES-based schemes.

All our attacks are based on an enhancement of the Statistical Fault Attack (SFA) presented by Fuhr et al. [18], which requires only very limited assumptions about the attacker's capabilities: the ability to induce a fault that leads to a biased (non-uniform) distribution in certain bytes. In case of AES, we assume that the attacker is able to influence some byte (or a larger structure) of the internal state of AES before the last application of MixColumns, so that the value of this byte is non-uniformly distributed. Particularly, we do not have to rely on the exact position of a fault, the number of faults injected during a single encryption, or even the knowledge that a certain fault has happened at all in an individual encryption. All we need to do is to collect ciphertexts and estimate the distribution of a single byte for various key guesses.

In order to evaluate the assumptions on the capabilities of an attacker, we also perform fault-injection experiments targeting three different hardware platforms. In the first setting, clock glitch attacks on a GCM software implementation executed on an 8-bit microcontroller are performed. In addition, we evaluate implementations using AES co-processors on a smartcard chip and a general-purpose microcontroller by means of laser fault injection and clock tampering, respectively. In all three settings, 4 bytes of the last round key of AES could be successfully recovered with 30, 16, and 1 200 faulty ciphertexts, respectively. In all practical scenarios, the attack has to be repeated three more times to recover the full last round key (in case of AES-128).

Outline. The remainder of the paper is organized as follows. In Sect. 2, we give some background on fault attacks in general, recapitulate the work of Fuhr et al. [18] on SFA, and introduce our attack model. In Sect. 3, we show how SFA can be applied to various AES-based authenticated encryption schemes.

Finally, we present practical experiments and verify the practicality of SFA on three different hardware platforms in Sect. 4.

2 Background

In this section, we revisit the Statistical Fault Attacks on AES underlying our attacks. We start with a general overview of different types of fault attacks, and briefly describe the biased fault model in the attack of Fuhr et al. [18]. Finally, we discuss the modified, much more general biased fault model we use in this paper, and how to identify the best key candidates.

2.1 Fault Attacks

Since the seminal work of Boneh et al. [13], it has been shown that many cryptographic algorithms are susceptible to Fault Attacks (FA). Indeed, numerous papers have proposed FA on most cryptographic primitives, including symmetric ciphers (DES [11], AES [37], etc.) as well as asymmetric schemes (RSA [13], Elliptic Curve Cryptography [10], etc.).

Fault attacks induce a logical error by physical means in one of the intermediate variables of a cryptographic primitive, and exploit the erroneous result to get information on the secret key. The means to inject a logical error can consist in over- or under-powering the device during a short time period, tampering its clock, or injecting a light beam or an electro-magnetic field inside the device [7,31,45].

Several cryptanalytic methods have been developed to exploit erroneous results in order to retrieve the key. In Differential Fault Analysis (DFA) [11], the attacker runs a cryptographic function twice on the same input and introduces a fault near the end of one of the computations. Then, information on the key can be retrieved from the differences between the correct and the faulty output. The Safe Error Attack (SEA) [49] fixes part of the cryptographic secret to a known value. Then, the observation of a collision on the result of a correct and faulted computation for identical inputs leaks information on the secret. In Collision Fault Analysis [12], one runs a cryptographic operation on two related inputs, and introduces a fault near the beginning of one of the computations. The adversary then exploits cases where a collision on the outputs occurs.

A common requirement of all these fault attacks is the necessity of processing two inputs that are either identical or related, in order to generate pairs of correct/faulty ciphertexts. Therefore, the attacker needs to be able to control the input of a cryptographic operation, which classifies them as chosen-plaintext attacks. Some of these FA require only one pair of correct/faulty outputs obtained from the same input, whereas others require several pairs to retrieve the secret key.

2.2 Statistical Fault Attacks

In 2013, Fuhr et al. proposed a new type of fault attack, called Statistical Fault Attack (SFA) [18]. In contrast to most previous attacks, the adversary only requires a collection of faulty ciphertexts encrypted with the same key. Hence, SFA works with random and unknown plaintexts.

Fault Model. Unlike most traditional fault attacks, SFA requires a slightly different fault model. Assuming that intermediate variables get uniformly distributed towards the last rounds for secure cryptographic primitives like AES, an attacker has to be able to induce faults which change the distribution of some intermediate values to be non-uniform. In particular, Fuhr et al. considered the following three fault models:

(a) the stuck-at-0 fault model with probability 1,
(b) the stuck-at-0 fault model with probability 1/2,
(c) the stuck-at model to an unknown and random value e with probability 1.

Using these non-uniform fault models, Fuhr et al. were able to show several attacks on AES based on simulations. Their attacks target the last 4 rounds with a small number of faulty ciphertexts and practical complexity.

Description of the AES. AES is a byte-oriented block cipher following the wide-trail design strategy. It operates on a state of 4×4 bytes and updates it in 10, 12, or 14 rounds, depending on the key size of 128, 192, or 256 bits. In each round (except the last one with no MixColumns), the following four transformations are applied.

SubBytes (SB): This step is the only non-linear transformation of the cipher. It is a permutation consisting of an S-box S applied to each byte of the state.
ShiftRows (SR): This step is a byte transposition that cyclically shifts each row of the state by different offsets. Row j is shifted right by j byte positions.
MixColumns (MC): This step is a permutation operating on the state column by column. To be more precise, it is a left-multiplication by a 4×4 circular MDS matrix M over $\mathbb{F}_{2^8}$.
AddRoundKey (AK): In this transformation, the state is modified by combining it with a round key with a bitwise `xor` operation.

Attack Procedure and Complexity. While Fuhr et al. proposed several attack variants, we will focus only on the attack that targets the 9th round of AES. When changing the distribution of one byte of AES before the last MixColumns, they showed that with these fault models, 4 bytes of the last round key could be recovered with high probability using the Squared Euclidean Imbalance (SEI) distinguisher with only 6, 14, and 80 faulty ciphertexts, respectively. We briefly recount the attack below, but refer to [18] for a more detailed description.

If we denote our target state before the last $\mathsf{MixColumns}$ in the encryption to the i^{th} ciphertext by $\tilde{S}_9^i$, we can express one byte of this state as a function of the ciphertext $\tilde{C}^i$, 4 bytes of the last round key K_{10}, and one byte of $\mathsf{MC}^{-1}(K_9)$, as follows. Our target state is

$$\begin{aligned}\tilde{S}_9^i &= \mathsf{MC}^{-1}(\mathsf{SB}^{-1} \circ \mathsf{SR}^{-1}(\tilde{C}^i \oplus K_{10}) \oplus K_9) \\ &= \mathsf{MC}^{-1}(\mathsf{SB}^{-1} \circ \mathsf{SR}^{-1}(\tilde{C}^i \oplus K_{10})) \oplus \mathsf{MC}^{-1}(K_9).\end{aligned}$$

Each byte of $\tilde{S}_9^i$ can therefore be deduced using one hypothesis on 4 bytes of K_{10} and on one particular byte of $\mathsf{MC}^{-1}(K_9)$. As shown by Fuhr et al., the xor with $\mathsf{MC}^{-1}(K_9)$ does not modify the distance of the biased distribution from uniform. Hence, it can be omitted in the attack. In other words, this allows to mount the attack on a modified $\tilde{S}_9^{i}{}'$:

$$\tilde{S}_9^{i}{}' = \mathsf{MC}^{-1} \circ \mathsf{SB}^{-1} \circ \mathsf{SR}^{-1}(\tilde{C}^i \oplus K_{10}).$$

This allows us to recover 4 bytes of the last round key K_{10} by making 2^{32} hypotheses on their value and predicting one byte of $\tilde{S}_9^{i}{}'$. By repeating the attack 4 times, one can recover the complete last round key K_{10}.

2.3 A Generalized Fault Model

In this work, we want to go beyond specific fault models like in Sect. 2.2. The only assumption we make is that the attacker is able to influence some byte (or a larger structure) of the internal state of AES before the last $\mathsf{MixColumns}$ such that this value becomes clearly non-uniformly distributed. We make no assumptions about the details of this non-uniformity, nor do we require that the attacker knows the new distribution. To exploit this type of fault, the attacker will collect faulty (biased) ciphertexts, compute backwards to the target byte for different key guesses, and try to reject wrong key guesses that would result in an approximately uniform measured distribution of the biased target byte. In the remainder of this section, we discuss how to identify the non-uniform distribution for the wrong key guesses.

We do not consider the distribution on bit-level, but for example on byte-level. Exploiting such non-uniform distributions of multi-bit values (more specifically, distributions of several sums of single bits) has already been investigated in the context of multidimensional linear cryptanalysis [21]. However, the distributions in this context are typically very close to uniform, unlike the distributions we expect in the case of SFA. Unfortunately, as noted by Samajder and Sarkar [44], the state-of-the-art framework for multidimensional linear cryptanalysis is not suitable for handling distributions which are significantly different from uniform. On the positive side, testing the closeness of discrete distributions [41] is a well-established field of research. Here, the central challenge is to determine whether two discrete distributions are the same (or close to each other) with the help of as few samples as possible. In our case, we want to determine whether our given samples are distributed uniformly or not.

The algorithms needing the fewest samples to perform this task are based on an idea of Goldreich and Ron [19]. Their algorithm makes use of collisions between sampled values to test for uniformity, since the expected number of collisions is lowest for uniformly distributed samples. Hence, the further a distribution deviates from the uniform distribution, the more collisions and multi-collisions we expect.

Of course, it is possible to directly base the testing of the key hypothesis on uniformity testing. For instance, Batu et al. [8] present a test which requires $O\left(\epsilon^{-4} \cdot \sqrt{2^s} \cdot \log(1/\gamma)\right)$ samples for distributions over 2^s-element sets. Their test accepts with probability $1-\gamma$ if the samples come from a distribution with ℓ_1-norm distance smaller than $\epsilon/\sqrt{3 \cdot 2^s}$ to the uniform distribution. It rejects with probability $1-\gamma$ if the samples come from a distribution which is more than ϵ away from the uniform distribution.

However, for our use-case, an approach that ranks keys according to some metric, like the number of collisions, is more suitable than a binary decision whether the measured distribution is uniform or not. Significantly more samples are needed to clearly separate the distribution for the right key hypothesis from the wrong ones to enforce a binary decision, whereas for the ranking, it is usually sufficient if the right key is ranked somewhere among the top candidates. Since the uniformity tests of Batu et al. [8] and Paninski [36] are actually based on counting collisions, they also provide us with a starting point for a ranking algorithm. This algorithm ranks the key hypothesis according to the number of collisions, and gives multi-collisions a higher weight. In our experiments, this ranking algorithm performs as good as ranking based on the SEI.

Interestingly, the key ranking mechanism based on the SEI used in [18,38] can also be linked to counting collisions. Let s be the bitsize of our biased intermediate value $S_i = f^{-1}(\hat{K}, \tilde{C}_i)$, computed from the faulty ciphertext $\tilde{C}_i$ under the key hypothesis $\hat{K}$. Assuming that we have N faulty ciphertexts, the SEI d is calculated as

$$d(\hat{K}) = \sum_{\delta=0}^{2^s-1} \left(\frac{\#\{i \mid f^{-1}(\hat{K}, \tilde{C}_i) = \delta\}}{N} - \frac{1}{2^s} \right)^2 .$$

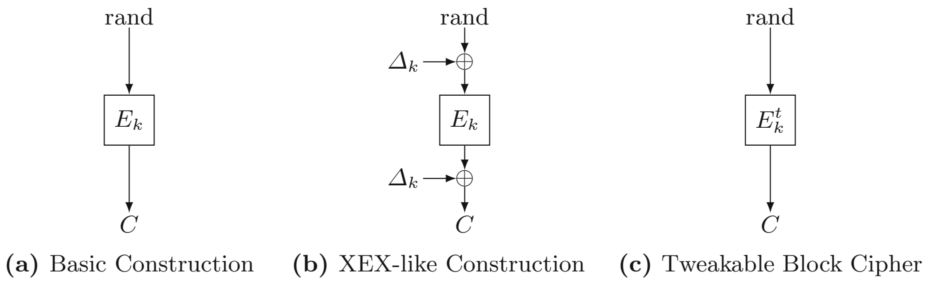
This distinguisher assigns high values to key hypotheses $\hat{K}$ that lead to distributions of intermediate values S_i with many collisions. For instance, consider a sample size of $N = 2^s$ samples. Then, the SEI is essentially counting collisions, since only events that occur exactly once do not increase d. Moreover, since the deviation from uniform is squared, a greater deviation, or in our sense a multi-collision, contributes more to d.

To sum up, it turned out that the SEI cannot be outperformed in practice by a new ranking algorithm based on counting collisions, since the SEI is actually doing that. Hence, we decided to stick to the more common SEI to measure if the distribution of one byte value becomes clearly non-uniformly distributed. So for AES, the 4-byte key guesses of the last round key are ranked according to the resulting SEI of one byte before the last MixColums when decrypting faulty ciphertexts for one round. To be able to observe non-uniformness and to evaluate the SEI, we require the input to the block cipher to be different for each fault and the block cipher output to be known.

3 Statistical Fault Attacks on Authenticated Encryption

In this section, we evaluate the applicability of the Statistical Fault Attack to several authenticated encryption modes for AES. This includes the widely-used ISO/IEC-standardized modes like CCM [48], EAX [9], GCM [32] and OCB [40], as well as new authenticated encryption modes proposed in the CAESAR initiative [46]. For evaluating the applicability of the fault attacks to these authenticated encryption schemes, we only need very limited assumptions. As already stated in Sect. 2, we assume that the attacker is able to influence some byte (or a larger structure) of the internal state of AES before the last MixColumns operation in a way that this value becomes clearly non-uniformly distributed.

We classify the investigated authenticated encryption modes into three categories, as illustrated in Fig. 1:



(a) Basic Construction **(b)** XEX-like Construction **(c)** Tweakable Block Cipher

Fig. 1. Classification of AES-based authenticated encryption schemes.

Basic Construction. The schemes in this category allow to directly observe the output of the block cipher. This includes schemes based on classical encryption schemes such as CTR [15], CBC [17], CFB [17], etc., but also schemes based on the XE construction [39], which masks the input of the block cipher using secret masks Δ_k. More generally, we assume that the input to the block cipher is a secret random value, but the output is observable to the attacker.

XEX-like Construction. This construction is similar to XE, but unlike XE, both the input and the output of the block cipher are masked using secret, nonce-dependent masks Δ_k. Constructions following the XEX construction [39] include for instance IAPM [28], OCB [40], and several of the CAESAR candidates.

Tweakable Block Cipher. The third category covers schemes that use a dedicated tweakable block cipher, which depends on a (typically nonce-dependent) tweak in addition to the secret key. Since the focus of this work is on

AES-based modes, we will restrict ourselves to constructions using the AES round function and following the TWEAKEY framework [27], such as for instance the CAESAR candidates KIASU [26] and Deoxys [24].

In the remainder of this section, we will discuss the applicability of Statistical Fault Attacks to schemes of these three categories in turn.

3.1 Application to the Basic Construction

In this construction, the output of the block cipher is directly known to the attacker, or can trivially be recovered by, say, xoring observable values with public values or constants. It is easy to see that in this case, the Statistical Fault Attack described in Sect. 2 can be applied in a straight-forward way to recover the secret key k. As an example, we discuss the application of Statistical Fault Attacks on AES in counter (CTR) modes as used in GCM, CCM and EAX (all standardized by ISO/IEC).

Statistical Fault Attack on CCM, EAX and GCM. As a representative example for the three modes, we will discuss the attack on CCM, which is shown in Fig. 2. As its name implies, the CTR-with-CBC-MAC mode (CCM) can be split into an encryption part using AES in counter mode to encrypt the plaintext P and an authentication part using CBC-MAC to authenticate the nonce N, associated data A, and plaintext P, which generates the tag T. For clarity, we have substituted the first part of the CBC-MAC, where the associated data is processed, with its outcome V in Fig. 2. Since the fault attack is solely performed on the encryption part, the following observations also hold for EAX and GCM that both use AES in CTR mode for encryption.

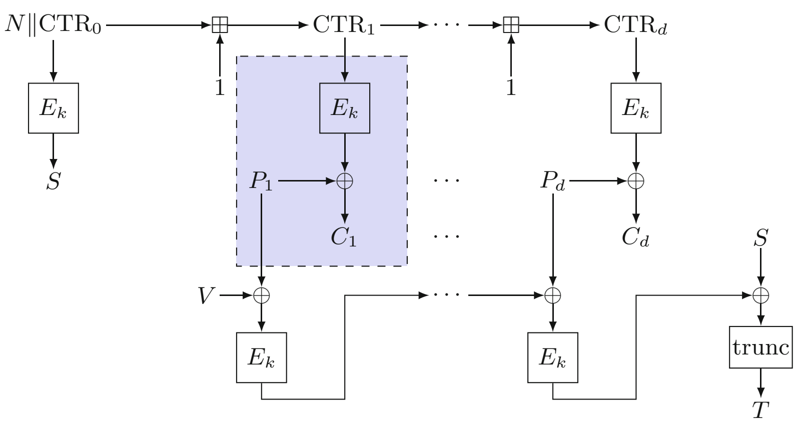


Fig. 2. The counter with CBC-MAC mode.
]

For the sake of simplicity, we restrict our fault attack to the encryption E_k of the first plaintext block (marked by the dashed rectangle in Fig. 2). Let us recall the conditions of Sect. 2 that are necessary for the Statistical Fault Attack to work:

1. The inputs of the block cipher need to be different for each fault.
2. The block cipher output needs to be known.

Condition 1 is always fulfilled, since it is required that the nonce N changes for each encryption and thus, the input to E_k changes as well. Condition 2 is fulfilled assuming a known plaintext attack, where the plaintext block P_1 is known to the attacker. Then, one can compute the keystream part for encrypting this plaintext block by xoring it with C_1. The resulting keystream is the output of the block cipher E_k. To sum up, we are able to observe outputs of the block cipher E_k for various inputs. Thus, we have the same preconditions as for the fault attack on plain AES described in Sect. 2. Hence, the attack can be applied to CCM (and any other scheme based on CTR mode) in a straight-forward way. We want to stress that the attacker does not require to know the input of the block cipher, it is just necessary that it changes. Therefore, the attack also applies to modes where the value of the counter is unknown, such as EAX.

Statistical Fault Attack on OCB. Although ISO/IEC-standard OCB is based on the XEX construction, we show that it is also vulnerable to the attack on the basic construction. The reason for this is that if the last plaintext block is incomplete, it is instead processed using the XE construction, as shown in Fig. 3. Therefore, the knowledge of this incomplete last plaintext and ciphertext block allows an attacker to compute the output of the block cipher E_k and thus, the Statistical Fault Attack is again applicable.

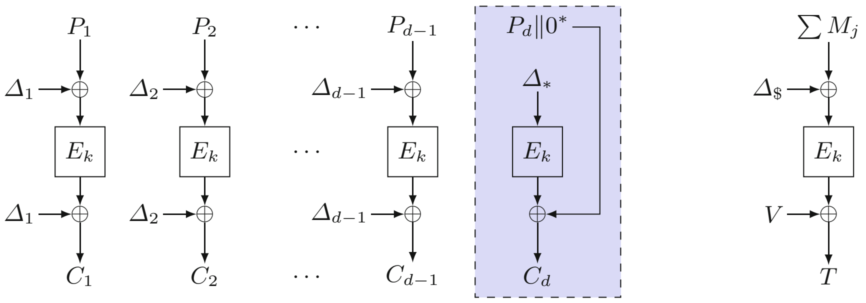


Fig. 3. Encryption in OCB.

Application to Other Modes. Besides CCM, EAX, GCM, and OCB, the fault attack discussed in this section also applies to several other authenticated encryption modes. For instance, to the CAESAR candidates Cloc [22] and

Silc [23], which are based on cipher-feed-back mode (CFB), where the ciphertext is the xor of the output of a block cipher E_k and the plaintext blocks. Another example is AES-OTR [34], which uses a balanced two-round Feistel network for encryption. The round function of this network is AES in an XE mode. Since the balanced Feistel network has only two rounds, knowledge of the plaintext and ciphertext implies knowledge of the block cipher output. Thus, again, the Statistical Fault Attack is directly applicable.

3.2 Application to XEX-Like Constructions

In this construction, the output of the block cipher is masked with a secret value Δ_k, which prevents a straightforward application of the basic attack. However, depending on how Δ_k is computed, the Statistical Fault Attack may nevertheless be applicable. In the simplest case, Δ_k is not nonce-dependent. This allows to repeatedly observe ciphertexts masked with a secret, but constant value Δ_k. We demonstrate how to exploit this in an attack on the CAESAR candidate AES-COPA [4].

Statistical Fault Attack on AES-COPA. AES-COPA uses an XEX-like construction for encrypting the plaintext, which is shown in Fig. 4. The input V of the plaintext processing is the result of a PMAC-like processing of the associated data A and the nonce N. Thus, V will change for different nonce values. Each processed ciphertext block requires two invocations of the block cipher E_k. AES-COPA masks both the input of the block cipher processing the plaintext blocks P_j, and the output of the block cipher that generate ciphertext blocks C_j. The masks are based on a secret value $L = E_k(0)$. We focus our attack on the block cipher call that generates C_1, as marked in Fig. 4.

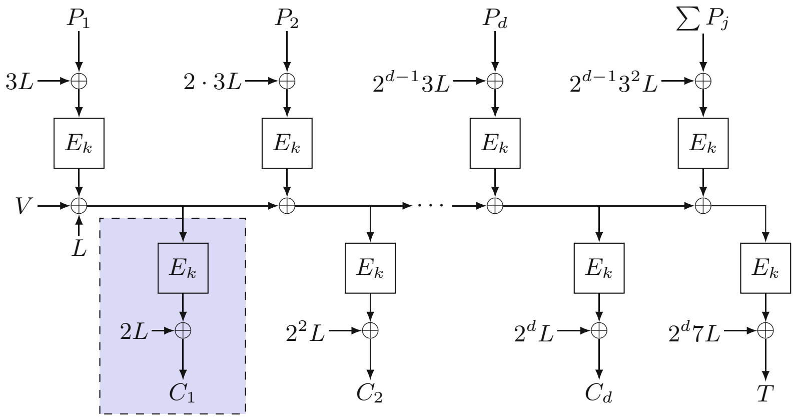


Fig. 4. Plaintext processing of AES-COPA, $L = E_k(0)$.

So far, only one of our two prerequisites for the SFA from Sect. 2 is fulfilled. We can vary the input of the block cipher calls by changing, for example, the nonce, associated data, or plaintext. However, the output of the block cipher is unknown, since it is masked with the secret value $\Delta_k = 2 \cdot E_k(0)$ to get C_1. To overcome this obstacle and since Δ_k solely depends on the secret key k, we consider Δ_k as a part of the key schedule to compute the last round key. Thus, instead of the last round key K_{10} of AES, we get $K'_{10} := K_{10} \oplus (2 \cdot E_k(0))$ as the last round key.

Hence, instead of recovering the last round key K_{10} of AES as in the attacks before, we now can recover K'_{10} by using SFA as described in Sect. 2. For recovering K'_{10}, the complexity and the needed numbers of faults are the same as for the attack on AES itself. However, the knowledge of K'_{10} does not directly lead to a key recovery attack of the master key k. Therefore, we need to perform the Statistical Fault Attack a second time. One option is to target again the first plaintext block and use our knowledge of K'_{10} to now target the AES round key K_9. Alternatively, we repeat the attack for the second plaintext block to recover $K_{10} \oplus (4 \cdot E_k(0))$ and thus get K_{10} by solving the resulting linear system. In both cases, the master key can then easily be recovered from K_9 and K_{10}, respectively.

Application to Other Modes. Besides COPA, other schemes that use a nonce-independent Δ_k and allow the Statistical Fault Attack include ELmD [14] and Shell [47]. In contrast, some schemes, such as IAPM, OCB, or some CAESAR candidates, also include the nonce in the computation of Δ_k. All these schemes have in common that Δ_k changes unpredictably for each block cipher call, which prevents a straight-forward application of Statistical Fault Attacks.

Instead of relying on misuse settings like repeated nonces, we will have a closer look at how these schemes typically compute Δ_k. In many cases, Δ_k can be decomposed into two values: a known, nonce-dependent part δ_N, and a secret, key-dependent part δ_k, which are then for example combined with a linear function to produce Δ_k. In this case, we can adapt our attack as follows, similar to the COPA case. First, we recover the modified last round key $K'_{10} = K_{10} \oplus \delta_k$. Depending on the key schedule and the function δ_k, this may already be sufficient to recover the master key (e.g., if δ_k and the key schedule are linear). Otherwise, we repeat the attack a second time to the round before to recover K_9 as described before.

3.3 Application to Modes Based on Tweakable Block Ciphers

In this construction, the authenticated encryption scheme uses a tweakable block cipher E_k^t instead of a regular block cipher as basic building block. In this case, the Statistical Fault Attack is not generally applicable. However, for some tweakable block ciphers such as the ones presented within the TWEAKEY framework [27], we can adapt our attack. In particular, this is possible if the last subkeys of the tweakable block cipher can be described by the composition of two values, $\delta_t \oplus \delta_k$. We illustrate the working principle of the attack for the CAESAR candidate Deoxys [24], but the same attack is also applicable to KIASU [26], where the tweak t is only xored to each round-key.

Statistical Fault Attack on Deoxys. Deoxys offers two modes of operation, both using two variants of the underlying tweakable block cipher Deoxys-BC. We focus on Deoxys$^{\neq}$-128-128, which uses Deoxys-BC-256 as underlying tweakable block cipher. As shown in Fig. 5, Deoxys$^{\neq}$ encrypts the individual plaintext blocks P_j in an ΘCB3-like [30] way. This ensures both the variation of the tweakable block cipher inputs, and knowledge of the outputs. However, since the tweak is partly defined by the nonce, we have to determine the influence of this nonce on the last round key that we want to recover using SFA. Thus, we have to have a closer look at the definition of the tweakable block cipher Deoxys-BC-256.

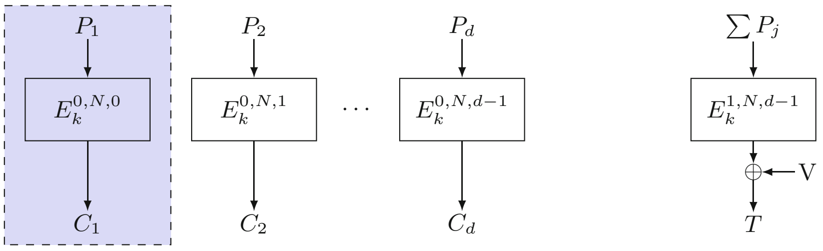


Fig. 5. Plaintext processing for Deoxys$^{\neq}$.

Figure 6 shows how Deoxys-BC-256 uses the round function f of the AES, but computes different round keys K_i based on the master key k and tweak t. Here, K_i is the xor sum of three values: a key-dependent round key K_i^k, a tweak-dependent round tweak K_i^t, and a round constant c_i. The values are updated using a simple byte permutation h. For instance, $K_0^k = k$, $K_0^t = t$, $K_1^k = 2\,h(k)$, $K_1^t = h(t)$, $K_r^k = 2\,h(2\,h(\ldots 2\,h(k)\ldots))$, and $K_r^t = h(h(\ldots h(t)\ldots))$.

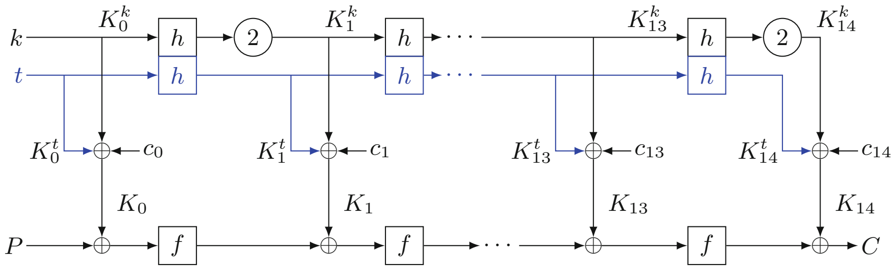


Fig. 6. Block cipher Deoxys-BC-256.

Since the value of the tweak used for encryption is publicly known, the varying part K_i^t of the round keys K_i can be easily calculated. The unknown parts K_i^k of the round key are constant for multiple calls of the block cipher under the same key k. Hence, the last round key K_{14}^k can be recovered with the SFA on AES described in Sect. 2.

3.4 Summary and Discussion of Results

We demonstrated in the previous sections that several authenticated encryption modes for AES are susceptible to Statistical Fault Attacks. A summary of the results is given in Table 1. However, Statistical Fault Attacks are applicable to a broader range of authenticated encryption schemes, and are not limited to AES-based modes. Natural targets for the attack include, for instance, the CAESAR candidates Joltik [25] and Scream [20], which also follow the TWEAKEY framework [27], or Prøst [29], which applies the modes of COPA [5] and OTR [35] to an Even-Mansour block cipher.

Moreover, the attack is not limited to block cipher based constructions. For instance, the APE construction [3] uses a secret key in the finalization for tag generation, making it a natural target for the attack. Also the sponge-based CAESAR candidates Ascon [16] and PRIMATEs [2] both employ a keyed finalization, with similar effects. However, the fact that large parts of the internal state are truncated to generate the authentication tag might complicate the attack.

Table 1. Statistical fault attacks on AES-based authenticated encryption modes in the nonce-respecting setting.

Primitive	Classification	Comments	Reference
CCM	Basic	CTR	Sect. 3.1
GCM	Basic	CTR	Sect. 3.1
EAX	Basic	CTR	Sect. 3.1
OCB	Basic	XE (incomplete blocks)	Sect. 3.1
Cloc/Silc[a]	Basic	CFB	Sect. 3.1
OTR[a]	Basic	XE	Sect. 3.1
COPA[a]	XEX		Sect. 3.2
ELmD[a]	XEX		Sect. 3.2
SHELL[a]	XEX		Sect. 3.2
KIASU[a]	TBC	TWEAKEY	Sect. 3.3
Deoxys[a]	TBC	TWEAKEY	Sect. 3.3

[a]CAESAR candidates.

4 Practical Verification/Implementation of the Attacks

In order to demonstrate the practical relevance of Statistical Fault Attacks and to validate the assumptions from previous sections, we performed three fault-injection experiments targeting real hardware.

An AES-GCM implementation executed on an off-the-shelf microcontroller served as target for the first experiment. In this context we used the ASM AES version from [1] to realize the block cipher. Due to the lack of embedded platforms implementing GCM or CCM completely in hardware, we put the focus of the following analysis on hardware AES co-processors available on a smartcard microcontroller and on a general-purpose microcontroller, respectively. The remaining parts for realizing the authenticated encryption modes are then implemented in software.

In all settings, the fault injections aim to induce a bias on at least one byte of the AES state before the last MixColumns transformation, and allow to reveal 32 bits of the last AES round key. For full key recovery, the attack has to be repeated three more times. The following list provides an overview of the fault-injection methods and the attack results for the three settings:

1. Clock tampering has been used to disturb the execution of the AES software implementation running on an ATxmega 256A3 general-purpose microcontroller. This setting allowed to reveal 4 bytes of the last round key with less than 30 faulted ciphertexts.
2. Laser fault injections on an AES co-processor on a smartcard microcontroller. Our experiments show that less than 16 faulty ciphertexts are sufficient to reveal 4 bytes of the last round key.
3. Clock tampering on a hardware AES co-processor implemented on a general-purpose microcontroller. In this setting, we need approximately 1 200 faulted ciphertexts for recovering 4 bytes of the last round key.

For all attacks, 4 bytes of the last round key can be recovered out of the faulted ciphertexts in less than one hour using an Intel Core i7 3770K. In the following, we give a detailed description and summary of the practical fault-injection attacks.

4.1 AES Software Implementation on an 8-Bit Microcontroller

In the following setting, we used clock glitches to provoke faults during an AES computation implemented in software on an 8-bit microcontroller. In particular, we used the ASM AES version from [1] for realizing the GCM AE mode.

For the clock-glitch experiments, we used a nominal clock frequency of 24 MHz ($T_{\mathrm{clk}} = 41.7\,\mathrm{ns}$). According to [1], one 128-bit encryption requires 2 555 clock cycles. For simplicity, we used one general-purpose I/O pin of the microcontroller for indicating the start of the AES encryption. This trigger pin together with the knowledge of the length of the AES encryption procedure allows to find the correct time interval for inserting the clock glitch. Next to that, our results show that faults in consecutive clock cycles also lead to successful key recovery. As a consequence, this behavior allows to relax the precision prerequisite of the trigger information.

With the found parameters, we collected two sets, each containing 80 faulty ciphertexts. For the first set, a single clock glitch was inserted. For the second set, clock glitches in 50 consecutive clock cycles were inserted. Next, we performed SFA attacks using an increasing number of faulty ciphertexts on both sets individually. The results containing the set size N, the SEI value for the correct subkey (SEI_c), and the maximum SEI value of the wrong subkey guesses (SEI_w) were stored in two separate lists (one list for each set) in the format $[N, \mathrm{SEI}_c, \max(\mathrm{SEI}_w)]$. For this attack scenario, we started with $N = 4$ and increased N in every iteration by 4.

Figure 7 displays the evolution of the SEI values for increasing number of ciphertexts in the single clock glitch setting. Values corresponding to the correct subkey are plotted in red, the maximum SEI values of the wrong subkey guesses are plotted in blue. With 30 faulty ciphertexts, SEI_c exceeds $\max(\mathrm{SEI}_w)$, which allows to reveal the correct subkey value.

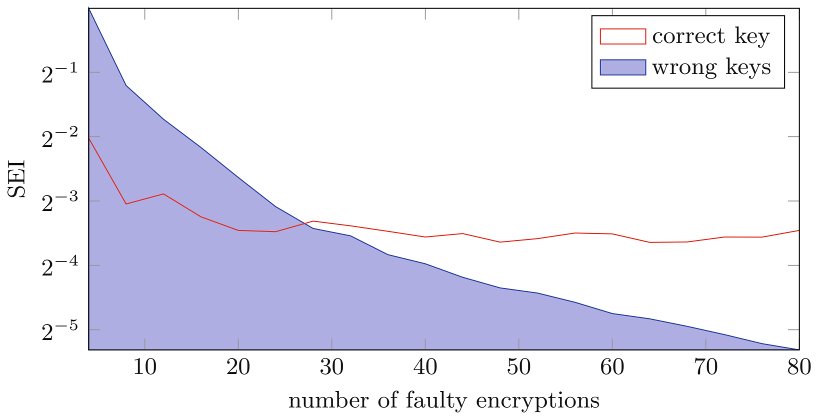


Fig. 7. SEI values for correct key (SEI_c) plotted against best SEI for a wrong key ($\max(\mathrm{SEI}_w)$) for increasing number of faulty encryptions. Setup: AES software implementation, single clock glitch (Table 2). (Color figure online)

Figure 8 displays the evolution of the SEI values for an increasing number of ciphertexts for the setting with 50 consecutive clock glitches. In this setting, 24 ciphertexts are sufficient for SEI_c to exceed $\max(\mathrm{SEI}_w)$, which allows to reveal the correct subkey value.

Results of the fault attacks targeting the AES software implementations using clock glitches show that with 30 faulty ciphertexts, it is possible to reveal the 32-bit subkey if a single clock glitch is inserted. Furthermore, if the clock glitch is inserted in 50 consecutive clock cycles, approximately 25 faulty ciphertexts are sufficient for subkey recovery. We did not further investigate the approach

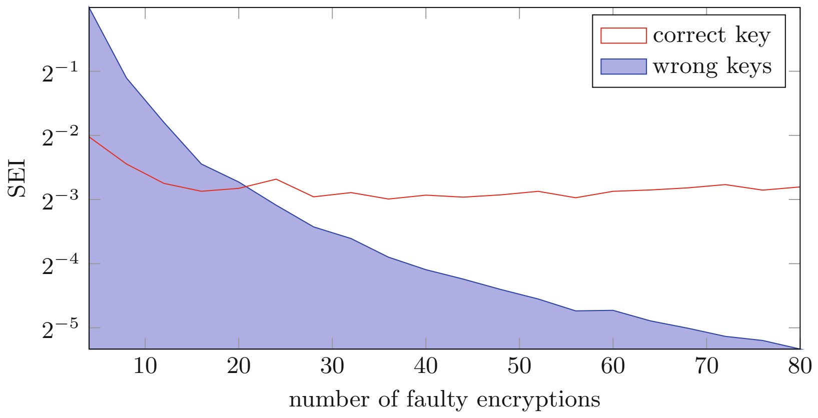


Fig. 8. Evolution of the SEI values with increasing number of faulty encryptions. Setup: AES software implementation, multiple clock glitches (Table 2).

of inserting the clock glitch in consecutive clock cycles because this is out of scope of the current work. Nevertheless, by carefully trimming the fault injection parameters, the number of faulty ciphertexts for successful subkey recovery could probably be further decreased.

4.2 AES Hardware Co-Processor of a Smartcard Microcontroller

In this experiment, we used a laser fault injection system to induce faults during encryptions of an AES Hardware co-processor of a smartcard microcontroller. This co-processor can easily be used as building block for realizing authenticated encryption modes like GCM or CCM on the smartcard.

The laser fault injection system consists of an infrared laser diode module and a microscope allowing to focus the laser spot depending on the microscope objective used. Here an objective with a 10× magnification is used. The whole system is mounted on a motorized X-Y-Z stage.

As the smartcard microcontroller runs its own operating system, the only signal available for triggering the laser injection system is the sending of the encryption command through APDU command. Therefore, a temporal delay is added to postpone the laser injection during the AES encryption thanks to a remotely controllable pulse generator. Furthermore, as the smartcard microcontroller runs on its own internal clock network, an inherent temporal jitter is present due to the asynchronism between the laser injection system and the smartcard microcontroller clock network. These experimental conditions are very close to the ones present in real world scenarios.

By applying a spatial fault injection cartography, we have been able to find a spatial position where only one byte of the AES state is faulted. Furthermore, by trying different delays, we found a spatio-temporal setting where only 4 bytes of the ciphertext were faulted with a high reliability. By studying the indices of

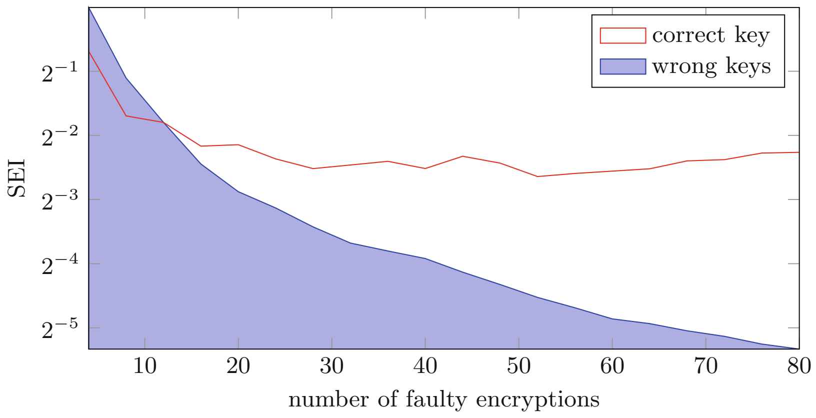


Fig. 9. Evolution of the SEI values with increasing number of faulty encryptions. Setup: AES hardware co-processor of a smartcard microcontroller, laser (Table 3). (Color figure online)

the faulted ciphertext bytes, we concluded that we successfully induced a fault on one byte of the AES state just before the last MixColumns. The fact that the hardware AES module can also be used outside of the context of authenticated encryption, i.e., for encrypting single plaintext blocks, simplified this profiling. However, if the stand-alone usage of the AES co-processor is not possible on the attacked platform, the search for the right fault injection parameters becomes more complicated, but is still feasible.

With the found parameters, we collected again 80 faulty ciphertexts. With the collected faulty ciphertexts, the same evaluation as in the previous section was conducted. We started again with an initial attack set size $N = 4$ and increased the size of the attack set by 4 in every iteration. The evolution of the SEI values with increasing set size is depicted in Fig. 9. Values corresponding to the correct subkey are plotted in red, the maximum SEI values of the wrong subkey guesses are plotted in blue.

As depicted on Fig. 9, SEI_c already exceeds $\max(\text{SEI}_w)$ with only $N = 16$ ciphertexts. Therefore, this number of ciphertexts allows to retrieve 4 bytes of the correct last round key. This result validates the practicability of the fault model and even shows that laser-based fault injection systems are well suitable for this kind of attacks.

4.3 AES Co-Processor on a General-Purpose Microcontroller

In this setting, we use clock glitches to inject faults during the encryption procedure of an AES co-processor integrated on a general-purpose microcontroller. This co-processor can on the one hand be used as stand-alone block cipher to encrypt plaintext blocks, on the other hand it can be used in the context of AE for realizing a mode of operation like GCM or CCM. The co-processor in

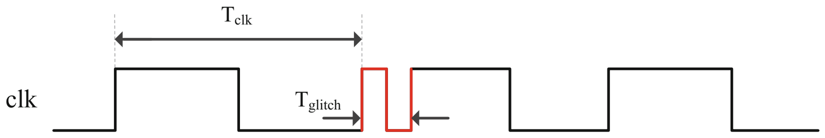


Fig. 10. Clock signal with intentionally inserted additional positive clock edge.

stand-alone mode allows profiling the hardware in order to find suitable fault-injection parameters. The target of the fault injection is the output of the byte substitution (SubBytes) in the 9th AES round. The AES co-processor implements the SubBytes function with pure combinational logic. Since one column of the state is processed in a single clock cycle, this allows to create faults in 4 bytes of the state with a single clock glitch.

We define with T_{glitch} the time interval between two subsequent positive clock edges in case of a clock glitch. This value is smaller compared to the nominal clock period T_{clk}, as illustrated in Fig. 10. If T_{glitch} is smaller than the path delay of the combinational SubBytes block, the output value of this block has not settled to its correct, stable value. As a result, a wrong value is sampled by the registers at the output of the block, which leads to faults in the ciphertext.

For the clock glitch experiments, we used a nominal clock frequency of 10 MHz ($T_{\text{clk}} = 100$ ns). Preliminary fault experiments allowed to find the correct clock cycle (i.e., the delay between the start of the encryption and the targeted instruction) to disturb the SubBytes operation in the 9th round before the MixColumns step. With $T_{\text{glitch}} = 10.2$ ns, we achieved a fault probability of 99.5 %.

With these parameters, we executed the AES encryption to receive 2 000 faulty ciphertexts. The increased number of ciphertexts was required because preliminary experiments revealed that the bias introduced with the clock glitch was significantly smaller compared to the bias introduced by the laser attack. With the collected faulty ciphertexts, the same evaluation as in the previous section was conducted. Due to a smaller bias, we started with an initial attack set size $N = 32$ and increased the size of the attack set by 32 in every iteration. The evolution of the SEI values with increasing set size is depicted in Fig. 11. Values corresponding to the correct subkey are again plotted in red, the maximum SEI values of the wrong subkey guesses are plotted in blue.

As depicted on Fig. 11, starting at 1 200 ciphertexts, SEI_c exceeds $\max(\text{SEI}_w)$. This allows to reveal the correct subkey in an attack setting. Compared to the results presented in the previous section, the number of required ciphertexts is nearly 100 times higher, but the number is still practical and this amount of ciphertexts can be collected within minutes. However, the effort for performing clock-glitch attacks compared to laser fault attacks (e.g., preparing the fault-injection environment, finding good fault-injection parameters) is significantly smaller, which has to be taken into account.

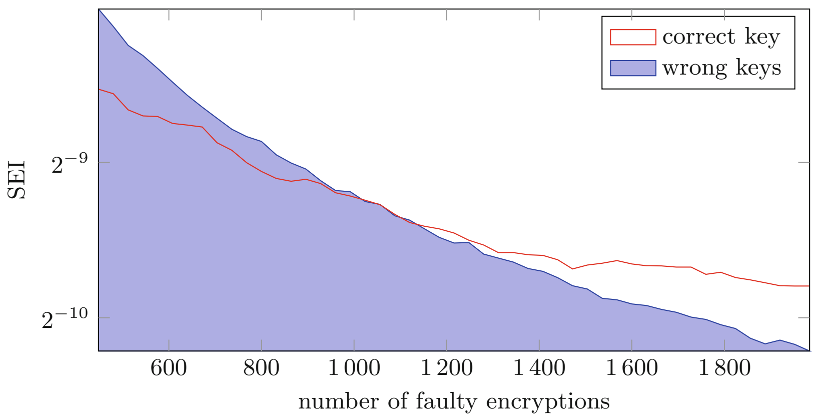


Fig. 11. Evolution of the SEI values with increasing number of faulty encryptions. Setup: AES co-processor on a general-purpose microcontroller, clock glitch (Table 4). (Color figure online)

4.4 Discussion and Remarks

The goal of the attacks presented in this section is a feasibility study proving that the assumed biased fault model is indeed valid on different platforms using different fault-injection mechanisms.

For the software implementation, a general-purpose I/O pin indicating the start of the AES encryption has been used, which allowed a precise fault injection using clock glitches. Real-world scenarios, like the second experiment targeting the smartcard microcontroller, typically do not allow the usage of a trigger pin. In such scenarios, other sources for synchronizing the fault-injection procedure can be applied, like spying the communication or the power profile. This can decrease the precision of the fault injections.

But it is important to note that the outcome of the SFA attack does not strictly rely on a precise fault injection. If only a subset of the received ciphertexts are affected by the expected fault pattern, the remaining ciphertexts (fault-free or fault hitting another location during the cipher rounds) are treated as noise. A more reliable fault injection process however minimizes the number of required ciphertexts for successful key recovery.

Furthermore, when the attacked platform allows the usage of the AES co-processor for stand-alone encryption (e.g., as in the previous experiments), one can easily perform a profiling step which simplifies the search for appropriate fault injection parameters. Nevertheless, if the AES co-processor can only be used in the context of the authenticated encryption mode, it is still possible to find the appropriate fault injection parameters. Of course, the number of attempts and the search space for the parameters increase, resulting in a more time-consuming setup phase for the fault injection.

With the practical results presented in this section, we showed that implementations of AES-based authenticated encryption modes on different hardware platforms are vulnerable to the proposed fault attacks introduced in this work.

5 Conclusion

In this work, we demonstrate for the first time that a wide range of nonce-based authenticated encryption schemes, including the widely used ISO/IEC standards CCM, GCM, EAX, and OCB, are susceptible to fault attacks. All our attacks need only very limited assumptions about the attacker's capabilities. To confirm these assumptions and to show the practical relevance of the attacks, we perform several fault-injection experiments targeting real hardware. This highlights the need for dedicated fault attack countermeasures for authenticated encryption schemes. Although our analysis focus only on AES-based constructions, we want to note that it is applicable to a broader range of authenticated encryption schemes. This is part of future work.

Acknowledgments. The authors would like to thank the organizers and participants of ASK 2015 that initiated this work and the anonymous reviewers for useful comments. The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644052 (HECTOR). Furthermore, this work has been supported in part by the Austrian Research Promotion Agency (FFG) under grant number 845589, by the Austrian Science Fund (project P26494-N15) and by the French ANR-14-CE28-0015 project.

A Data of Practical Verification/Implementation

Table 2. Evolution of the SEI values for correct key (SEI_c) and the best wrong key ($\max(\mathrm{SEI}_w)$) for increasing number of faulty encryptions N. Setup: AES software implementation, single clock glitch (left) and multiple clock glitches (right).

N	SEI_c	$\max(\mathrm{SEI}_w)$
4	0.25	1.00
8	0.12	0.43
12	0.13	0.30
16	0.11	0.22
20	0.09	0.16
24	0.09	0.12
28	0.10	0.09
32	0.10	0.09
36	0.09	0.07
40	0.08	0.06
44	0.09	0.05
48	0.08	0.05
52	0.08	0.05
56	0.09	0.04
60	0.09	0.04
64	0.08	0.04
68	0.08	0.03
72	0.08	0.03
76	0.08	0.03
80	0.09	0.03

N	SEI_c	$\max(\mathrm{SEI}_w)$
4	0.25	1.00
8	0.18	0.46
12	0.15	0.29
16	0.14	0.18
20	0.14	0.15
24	0.16	0.12
28	0.13	0.09
32	0.13	0.08
36	0.13	0.07
40	0.13	0.06
44	0.13	0.05
48	0.13	0.05
52	0.14	0.04
56	0.13	0.04
60	0.14	0.04
64	0.14	0.03
68	0.14	0.03
72	0.15	0.03
76	0.14	0.03
80	0.14	0.02

Table 3. Evolution of the SEI values for correct key (SEI_c) and the best wrong key ($\max(\mathrm{SEI}_w)$) for increasing number of faulty encryptions N. Setup: AES hardware co-processor of a smartcard microcontroller, laser.

N	SEI_c	$\max(\mathrm{SEI}_w)$
4	0.62	1.00
8	0.31	0.46
12	0.29	0.29
16	0.22	0.18
20	0.23	0.14
24	0.19	0.11
28	0.17	0.09
32	0.18	0.08
36	0.19	0.07
40	0.17	0.07
44	0.20	0.06
48	0.19	0.05
52	0.16	0.04
56	0.17	0.04
60	0.17	0.03
64	0.17	0.03
68	0.19	0.03
72	0.19	0.03
76	0.21	0.03
80	0.21	0.02

Table 4. Evolution of the SEI values for correct key (SEI_c) and the best wrong key ($\max(\mathrm{SEI}_w)$) for increasing number of faulty encryptions N. Setup: AES co-processor on a general-purpose microcontroller, clock glitch.

N	SEI_c	$\max(\mathrm{SEI}_w)$	N	SEI_c	$\max(\mathrm{SEI}_w)$
32	0.02930	0.08203	1 024	0.00165	0.00164
64	0.01514	0.03369	1 056	0.00162	0.00162
96	0.01020	0.02040	1 088	0.00155	0.00154
128	0.00769	0.01489	1 120	0.00150	0.00151
160	0.00625	0.01125	1 152	0.00147	0.00145
192	0.00521	0.00971	1 184	0.00145	0.00140
224	0.00474	0.00817	1 216	0.00143	0.00136
256	0.00430	0.00693	1 248	0.00138	0.00137
288	0.00398	0.00620	1 280	0.00135	0.00130
320	0.00355	0.00535	1 312	0.00131	0.00128
352	0.00341	0.00492	1 344	0.00131	0.00125
384	0.00304	0.00448	1 376	0.00130	0.00122
416	0.00284	0.00416	1 408	0.00129	0.00120
448	0.00271	0.00388	1 440	0.00127	0.00117
480	0.00266	0.00359	1 472	0.00122	0.00113
512	0.00247	0.00330	1 504	0.00124	0.00111
544	0.00241	0.00315	1 536	0.00125	0.00107
576	0.00240	0.00297	1 568	0.00126	0.00106
608	0.00233	0.00280	1 600	0.00124	0.00104
640	0.00231	0.00264	1 632	0.00123	0.00103
672	0.00229	0.00250	1 664	0.00123	0.00101
704	0.00213	0.00238	1 696	0.00123	0.00100
736	0.00206	0.00227	1 728	0.00123	0.00098
768	0.00195	0.00219	1 760	0.00119	0.00097
800	0.00188	0.00215	1 792	0.00120	0.00095
832	0.00182	0.00202	1 824	0.00117	0.00093
864	0.00180	0.00195	1 856	0.00116	0.00089
896	0.00181	0.00190	1 888	0.00114	0.00087
928	0.00178	0.00180	1 920	0.00113	0.00088
960	0.00171	0.00173	1 952	0.00113	0.00087
992	0.00168	0.00172	1 984	0.00113	0.00084

References

1. AVR crypto lib. http://avrcryptolib.das-labor.org. Accessed 13 Jan 2016
2. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mendel, F., Mennink, B., Mouha, N., Wang, Q., Yasuda, K.: PRIMATEs. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/primatesv102.pdf
3. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: APE: authenticated permutation-based encryption for lightweight cryptography. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 168–186. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46706-0_9
4. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: AES-COPA. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/aescopav2.pdf
5. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and authenticated online ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8269, pp. 424–443. Springer, Heidelberg (2013). doi:10.1007/978-3-642-42033-7_22
6. Balasch, J., Gierlichs, B., Verbauwhede, I.: An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs. In: Fault Diagnosis and Tolerance in Cryptography - FDTC 2011, pp. 105–114. IEEE (2011)
7. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. In: Fault Diagnosis and Tolerance in Cryptography - FDTC 2004, pp. 330–342 (2004)
8. Batu, T., Fortnow, L., Rubinfeld, R., Smith, W.D., White, P.: Testing closeness of discrete distributions. J. ACM **60**(1), 4 (2013)
9. Bellare, M., Rogaway, P., Wagner, D.: The EAX mode of operation. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 389–407. Springer, Heidelberg (2004). doi:10.1007/978-3-540-25937-4_25
10. Biehl, I., Meyer, B., Müller, V.: Differential fault attacks on elliptic curve cryptosystems. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 131–146. Springer, Heidelberg (2000). doi:10.1007/3-540-44598-6_8
11. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997). doi:10.1007/BFb0052259
12. Blömer, J., Krummel, V.: Fault based collision attacks on AES. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 106–120. Springer, Heidelberg (2006). doi:10.1007/11889700_11
13. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997). doi:10.1007/3-540-69053-0_4
14. Datta, N., Nandi, M.: ELmD. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/elmdv20.pdf
15. Diffie, W., Hellman, M.E.: Privacy and authentication: an introduction to cryptography. Proc. IEEE **67**(3), 397–427 (1979)
16. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/asconv11.pdf
17. Dworkin, M.: Recommendation for block cipher modes of operation. NIST Spec. Publ. **800**(38A), 1–59 (2001)

18. Fuhr, T., Jaulmes, É., Lomné, V., Thillard, A.: Fault attacks on AES with faulty ciphertexts only. In: Fischer, W., Schmidt, J. (eds.) Fault Diagnosis and Tolerance in Cryptography - FDTC 2013, pp. 108–118. IEEE Computer Society, Washington, DC (2013)
19. Goldreich, O., Ron, D.: On testing expansion in bounded-degree graphs. Electron. Colloquium Comput. Complex. (ECCC) **7**(20), 1–6 (2000)
20. Grosso, V., Leurent, G.L., Standaert, F., Varici, K., Journault, A., Durvaux, F., Gaspar, L., Kerckhof, S.: SCREAM. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/screamv3.pdf
21. Hermelin, M., Cho, J.Y., Nyberg, K.: Multidimensional extension of Matsui's Algorithm 2. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 209–227. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03317-9_13
22. Iwata, T., Minematsu, K., Guo, J., Morioka, S., Kobayashi, E.: CLOC. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/clocv2.pdf
23. Iwata, T., Minematsu, K., Guo, J., Morioka, S., Kobayashi, E.: SILC. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/silcv2.pdf
24. Jean, J., Nikolic, I., Peyrin, T.: Deoxys. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/deoxysv13.pdf
25. Jean, J., Nikolic, I., Peyrin, T.: Joltik. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/joltikv13.pdf
26. Jean, J., Nikolic, I., Peyrin, T.: KIASU. Submission to the CAESAR Competition (Round 1). http://competitions.cr.yp.to/round1/kiasuv1.pdf
27. Jean, J., Nikolić, I., Peyrin, T.: Tweaks and keys for block ciphers: the `TWEAKEY` framework. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 274–288. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45608-8_15
28. Jutla, C.S.: Encryption modes with almost free message integrity. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 529–544. Springer, Heidelberg (2001). doi:10.1007/3-540-44987-6_32
29. Kavun, E.B., Lauridsen, M.M., Leander, G., Rechberger, C., Schwabe, P., Yalçin, T.: Prøst. Submission to the CAESAR Competition (Round 1). http://competitions.cr.yp.to/round1/proestv11.pdf
30. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21702-9_18
31. Maurine, P.: Techniques for EM fault injection: equipments and experimental results. In: Bertoni, G., Gierlichs, B. (eds.) Fault Diagnosis and Tolerance in Cryptography - FDTC 2012, pp. 3–4. IEEE Computer Society, Washington, DC (2012)
32. McGrew, D.A., Viega, J.: The security and performance of the galois/counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30556-9_27
33. Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential fault analysis of the advanced encryption standard using a single fault. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 224–233. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21040-2_15
34. Minematsu, K.: AES-OTR. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/aesotrv2.pdf

35. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 275–292. Springer, Heidelberg (2014). doi:10.1007/978-3-642-55220-5_16
36. Paninski, L.: A coincidence-based test for uniformity given very sparsely sampled discrete data. IEEE Trans. Inf. Theory **54**(10), 4750–4755 (2008)
37. Piret, G., Quisquater, J.-J.: A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 77–88. Springer, Heidelberg (2003). doi:10.1007/978-3-540-45238-6_7
38. Rivain, M.: Differential fault analysis on DES middle rounds. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 457–469. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04138-9_32
39. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30539-2_2
40. Rogaway, P., Bellare, M., Black, J.: OCB: a block-cipher mode of operation for efficient authenticated encryption. ACM Trans. Inf. Syst. Secur. **6**(3), 365–403 (2003)
41. Rubinfeld, R.: Taming big probability distributions. ACM Crossroads **19**(1), 24–28 (2012)
42. Saha, D., Chowdhury, D.R.: SCOPE: on the side channel vulnerability of releasing unverified plaintexts. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 417–438. Springer, Heidelberg (2016). doi:10.1007/978-3-319-31301-6_24
43. Saha, D., Kuila, S., Roy Chowdhury, D.: EscApe: diagonal fault analysis of APE. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 197–216. Springer, Heidelberg (2014). doi:10.1007/978-3-319-13039-2_12
44. Samajder, S., Sarkar, P.: Another look at normal approximations in cryptanalysis. Cryptology ePrint Archive, Report 2015/679 (2015). http://ia.cr/2015/679
45. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski Jr., B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003). doi:10.1007/3-540-36400-5_2
46. The CAESAR committee: CAESAR: Competition for authenticated encryption: Security, applicability, and robustness (2014). http://competitions.cr.yp.to/caesar.html
47. Wang, L.: Shell. Submission to the CAESAR Competition (Round 2). http://competitions.cr.yp.to/round2/shellv20.pdf
48. Whiting, D., Ferguson, N., Housley, R.: Counter with CBC-MAC (CCM). RFC 3610 (2003)
49. Yen, S., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Trans. Comput. **49**(9), 967–970 (2000). http://dx.doi.org/10.1109/12.869328
50. Zussa, L., Dutertre, J.M., Clediere, J., Tria, A.: Power supply glitch induced faults on FPGA: an in-depth analysis of the injection mechanism. In: On-Line Testing Symposium - IOLTS 2013, pp. 110–115. IEEE (2013)