Fault Attacks on Nonce-Based Authenticated Encryption: Application to Keyak and Ketje

Christoph Dobraunig[1], Stefan Mangard[1], Florian Mendel[2], and Robert Primas[1](✉)

[1] Graz University of Technology, Graz, Austria
{christoph.dobraunig,stefan.mangard,robert.primas}@iaik.tugraz.at
[2] Infineon Technologies AG, Neubiberg, Germany
florian.mendel@infineon.com

Abstract. In the context of fault attacks on nonce-based authenticated encryption, an attacker faces two restrictions. The first is the uniqueness of the nonce for each new encryption that prevents the attacker from collecting pairs of correct and faulty outputs to perform, e.g., differential fault attacks. The second restriction concerns the verification/decryption, which releases only verified plaintext. While many recent works either exploit misuse scenarios (e.g. nonce-reuse, release of unverified plaintext), we turn the fact that the decryption/verification gives us information on the effect of a fault (whether a fault changed a value or not) against it.

In particular, we extend the idea of statistical ineffective fault attacks (SIFA) to target the initialization performed in nonce-based authenticated encryption schemes. By targeting the initialization performed during decryption/verification, most nonce-based authenticated encryption schemes provide the attacker with an oracle whether a fault was ineffective or not. This information is all the attacker needs to mount statistical ineffective fault attacks. To demonstrate the practical threat of the attack, we target software implementations of the authenticated encryption schemes Keyak and Ketje. The presented fault attacks can be carried out without the need of sophisticated equipment. In our practical evaluation the inputs corresponding to 24 ineffective fault inductions were required to reveal large parts of the secret key in both scenarios.

Keywords: Fault attack · Statistical ineffective fault attack · SIFA
Authenticated encryption · Keyak · Ketje

1 Introduction

With the rise of the Internet of Things (IoT), devices implementing authenticated encryption schemes will become ubiquitous. A trend, NIST is planning to address with standardization efforts in the area of lightweight authenticated

C. Cid and M. J. Jacobson, Jr. (Eds.): SAC 2018, LNCS 11349, pp. 257–277, 2019.
https://doi.org/10.1007/978-3-030-10970-7_12

encryption schemes [21,24]. As a consequence, authenticated encryption schemes will be more and more applied on devices in areas, where the physical access of malicious entities is unavoidable. Hence, implementation attacks like side-channel attacks and fault attacks, are a major concern for such devices as demonstrated, e.g., by Ronen, Shamir, Weingarten, and O'Flynn [28] in their attack on smart lamps. To identify and protect against the potential threats raised by implementation attacks, research in the practicability and applicability of implementation attacks on authenticated encryption schemes is needed.

As observed by many publications [29–31], the uniqueness of the nonce in authenticated encryption schemes prohibits the straight-forward application of prominent fault attacks like differential fault analysis (DFA) [10] to the authenticated encryption. In the case of authenticated decryption, the built-in validation of the authenticity of the processed data often provides an implicit detection of induced faults. Therefore, a lot of attacks published so far assume scenarios, where the uniqueness of the nonce is not ensured [31] or unverified plaintext is released [29], or even require a precise induction of faults at multiple locations during one execution of the authenticated encryption scheme [30]. Recently, statistical fault attacks (SFA) that are applicable to a wide-range of AES-based authenticated encryption schemes including popular schemes like GCM, CCM and OCB have been published [15]. However, the presented attacks face some limitations. In particular, they are only applicable to schemes where the secret key is processed right before the data is output. Thus, it is typically not applicable to sponge or stream cipher-based constructions. Moreover, they only work in the case of authenticated encryption, leaving fault attacks targeting authenticated decryption (assuming that the unverified plaintext is not released) as an open problem.

Our Contribution. In this work, we close the aforementioned gaps. We present the—to the best of our knowledge—first fault attacks targeting authenticated decryption/verification that are applicable to a broad range of nonce-based authenticated encryption schemes. In particular, the presented attacks are applicable whenever the nonce is mixed with the secret key during the initialization as it is the case in a wide range of authenticated encryption schemes. This includes sponge and stream cipher-based authenticated encryption schemes for which most of the existing fault attacks are not applicable.

We focus our analysis on Keyak and Ketje designed by Bertoni, Daemen, Peeters, Van Assche, and Van Keer [6,7]. Both designs are based on the Keccak-f permutation [4], which also underlies Keccak/SHA-3 [23]. Please note that the presented attacks do not exploit a weakness inherent in the design of Keyak and Ketje, these two primitives just serve as an example to show the applicability of fault attacks on sponge and stream cipher-based authenticated encryption schemes.

Our attacks are based on statistical ineffective fault attacks [14,16] and do not require an extensive profiling or characterization of the attacked device. Additionally, they are resilient against "errors" induced by miss-located faults, or in general fault inductions that do not behave as intended. As a consequence, they can be easily applied in practice as demonstrated by our attack targeting 8-bit software implementations of Keyak and Ketje running on an AVR Xmega 128D4. After inducing faults during authenticated decryptions and filtering for the inputs of 24 unaffected computations, we can recover large parts of the secret keys. The remaining unknown key bits can then either be bruteforced or further reduced by repeating the attack and inducing the fault at a different point in time.

Outline. In Sect. 2, we cover the required background of our attack. After, we describe the state-of-the-art of fault attacks, we give a short overview of authenticated encryption schemes. We provide a more detailed description of Keyak and Ketje, the two authenticated encryption schemes that are the main target of our practical attack evaluation, in Sect. 3. In Sect. 4, we discuss the idea and working principle of the attack. Section 5 describes the practical evaluation of our fault attack on a real microprocessor. We conclude the paper in Sect. 6.

2 Background

In this section, we give a brief introduction to fault attacks in general and state the idea behind Statistical Ineffective Fault Attacks (SIFA), recently proposed by Dobraunig, Eichlseder, Korak, Mangard, Mendel, and Primas [16], in more detail. We then recall the concept of nonce-based authenticated encryption with associated data.

2.1 Fault Attacks

The threat of fault attacks was demonstrated by Boneh, DeMillo, and Lipton [11] in 1997 when they showed the vulnerability of several asymmetric primitives like RSA to erroneous computations. Since then, fault attacks have been demonstrated targeting many other cryptographic schemes [9], including symmetric ones [10,25].

The way in which faults can be induced into a cryptographic computation is manifold. Originally, the most popular fault attacks were based on clock glitches or variations on the supply voltage. However, by the time, more and more sophisticated fault induction methods were presented like attacks based on lasers [32], EM-pulses [19], or even X-rays [1].

While the induction of (more or less) precise faults into a cryptographic computation is an essential prerequisite for the attack, the exploitation of the observed erroneous behavior is equally important. Biham and Shamir [10] proposed Differential Fault Analysis (DFA) as an effective key recovery method for DES. DFA requires the collection of pairs of valid and faulty ciphertexts where a fault was induced in the last few rounds of the encryption. The difference between valid and faulty ciphertexts together with knowledge about the faulted operation can then be used to recover the used secret key. Later it has been shown that DFA is not limited to DES and can be applied to broad range of block ciphers.

One immediate consequence of fault attacks was the evaluation of possible countermeasures that can prevent such attacks. One commonly used countermeasure is the detection of the induced fault by means of redundancy like double-execution [2]. Here, the cryptographic computation is performed twice and the output is only released, if the results of both computations match up. While double-execution does prevent the attacks presented so far, a more powerful attacker can still succeed by either inducing a fault that skips the final comparison or by inducing a fault with equivalent effect during both computations. On top of that, Safe Error Attacks (SEA) [34] or Ineffective Fault Analysis (IFA) [13] solely rely on valid outputs of faulted cryptographic computations and hence are unaffected by double-execution.

So far, most fault attacks require the attacker to send specific inputs multiple times to the attacked cryptographic implementation. This raises the question whether or not such attacks also apply to nonce-based authenticated encryption schemes where unique nonces prevent attackers from doing so. Indeed, the feasibility of fault attacks has been shown by Dobraunig, Eichlseder, Korak, Lomné, and Mendel [15] for various block cipher-based authenticated encryption schemes by using Statistical Fault Attacks (SFA) [18]. However, their attacks face some limitations. For instance, they are not applicable in a straight-forward manner to most sponge-based and stream cipher-based authenticated encryption schemes. In our attack, we make use of Statistical Ineffective Fault Attacks (SIFA) [16] that build upon the concepts of both SFA [18] and IFA [13].

2.2 Statistical Ineffective Fault Attacks

The Statistical Ineffective Fault Attack (SIFA) [16] is a technique that exploits distributions of faults that have been induced, but do not affect the outcome of a computation (ineffective faults). Concretely, the effect of an induced fault depends on the values that are currently processed by a device. As a consequence, the distribution of the values where an induced fault does not change the processed value is often biased in practice. This distribution can then be exploited in attacks, which cannot be precluded by popular detection/infection countermeasures [16]. As shown in [14], even additional masking does not preclude such attacks.

To discuss the basic working principles of SIFA, let us consider an encryption where an attacker is able to force (using fault inductions) one specific intermediate value to follow an unknown but non-uniform distribution during the computation. Such fault inductions are rather easy to achieve in practice as it has been shown, e.g. in [16] by using clock glitches for various microprocessors, or in [15] by using lasers on a hardware AES co-processor. If we continuously perform such faulted encryptions we will probably observe plain- or ciphertexts where the fault was ineffective. In those cases, the distribution of the targeted value, where the fault has been ineffective, might also follow a biased/non-uniform distribution.

Once the attacker has collected a sufficiently large set of unaffected plain- or ciphertexts, key recovery can be performed as follows. First, the attacker needs to identify all key bits that are involved in the calculation of the targeted value. Clearly, the time frame of the fault induction has to be either towards the beginning or the end of the encryption such that the targeted value only depends on parts of the key. Hence, when attacking sponge or stream cipher-based authenticated encryption schemes, the usual location for fault inductions is the initialization phase. Next, she calculates the targeted value for each collected unaffected plain- or ciphertext and every possible key candidate. The targeted value should, when calculated using the correct key candidate, follow a non-uniform distribution (which is usually not known to the attacker). In contrast, the calculated distribution for a wrong key guess is typically unrelated to the event that there has been an ineffective fault and hence, is expected to be closer to uniform. As a consequence, we are able to distinguish wrong key guesses from a right key guess. For a detailed description of the working principles of the attack including statistical background and on the effects of faults we refer to [14,16].

2.3 Authenticated Encryption

An authenticated encryption scheme provides confidentiality and authenticity for a given plaintext. It is usually modeled as a function of four input parameters: a secret key K, unique nonce N, associated data A and plaintext P [26]. The output of authenticated encryption is a tuple that consists of a ciphertext C and tag T:

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

The corresponding authenticated decryption takes the following five inputs: a secret key K, unique nonce N, associated data A, ciphertext C and tag T. During decryption T is used to verify the authenticity of A and C. If they are not authentic the original plaintext P is not released and the special error symbol $\perp$ is returned instead:

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

The concrete implementation of authenticated encryption schemes can differ significantly. Currently, many of the popular schemes like GCM [20], CCM [33], EAX [3], and OCB [27] are all based on block ciphers like AES. However, since the announcement of CAESAR [12], we can also see an increasing number of stream cipher-based and sponge-based authenticated encryption schemes. In the next section, we will present two such sponge-based designs: Keyak and Ketje, in more detail, since we will use them to describe the attack and for the practical evaluation.

3 Keyak and Ketje

Keyak [7] and Ketje [6] are sponge-based authenticated encryption schemes. Their design is heavily inspired by the hash function Keccak [4], the winner of the SHA-3 competition. While both schemes make use of variants of the permutation in Keccak, their modes of operation are slightly different. At first, we give a short description of Keccak and its underlying permutation. We then describe how Keyak and Ketje make use of the Keccak permutation in order to build an authenticated encryption scheme.

3.1 Keccak

Keccak is a sponge-based hash function that was selected as the winner of the SHA-3 competition. It is parameterized by the permutation Keccak-f, rate r, and capacity c.

Keccak-f, more precisely denoted by Keccak-$f[b]$, is an iterated permutation that operates on a b-bit state that is organized in 5×5 lanes of 2^l bits where l ranges from 0 to 6. The number of rounds n_r is determined by the width of the permutation and is equal to $12 + 2l$. Keccak-f consists of the 5 operations: $\theta, \rho, \pi, \chi, \iota$ that are applied to the state in the presented order in every round. From these 5 operations χ is the only non-linear transformation. The purpose of θ, π and ρ is to cause diffusion while ι breaks any symmetries.

In the case of Keccak, the lane size l equals 6, thus the state has a size of $5 \times 5 \times 64 = 1600$ bits and the number of rounds n_r is $12 + 2 \times 6 = 24$. Depending on the desired security, c is chosen as twice the desired preimage resistance in bits and $r = 1600 - c$. Following the sponge construction design principle, Keccak can be divided into two phases: an initial absorbing phase and a subsequent squeezing phase. During the absorbing phase input chunks of r bits are repeatedly XOR-ed into the state and subsequently processed by Keccak-f. Once all input chunks have been absorbed, a chunk of the desired hash bit-size can be extracted from the state (squeezing phase).

Besides Keccak-f, a variety of similar permutations Keccak-$p[b,n_r]$ were proposed by the Keccak designers. In contrast to Keccak-f, in Keccak-p the number of rounds n_r does not depend on the state size b anymore and can be set to any positive integer. This allows for more flexibility in the design of Keccak-based cryptographic primitives. The state size b is however still restricted to the same values. Next, we give basic descriptions of the authenticated encryption schemes Keyak and Ketje.

3.2 Keyak

Keyak is an authenticated encryption scheme that uses the *Motorist* mode of operation and is based on the Keccak-p permutation. Even though Keyak supports a parameterized degree of parallelism we limit our description to the (recommended) Lake Keyak variant that does not support parallelization and thus can be used even on constrained devices. Lake Keyak utilizes a 1600-bit state, uses the 12-round Keccak-$p[1600,12]$ permutation and performs authenticated encryption with 128 to 256 bits of secret key, up to 150 bytes of nonce and 128-bit tags. In the following, we describe the *Motorist* mode of operation, as used in Keyak. Whenever we refer to Keyak we mean Lake Keyak.

Motorist Mode. The *Motorist* mode defines how incoming messages are processed together with key, nonce, associated data and tag in Keyak. It is closely related to the duplex construction [5], with the main difference being the size of the input blocks. While the original duplex construction only allows input blocks as large as the outer part (rate r) of the underlying permutation, *Motorist* uses full-state keyed duplexes [22] that can make use of the full width of the permutation and thus allow higher throughput as shown in Fig. 1.

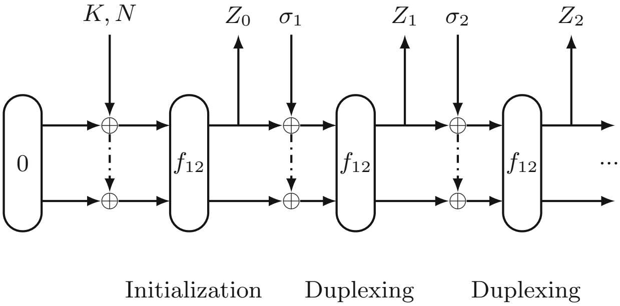

Fig. 1. Lake Keyak. f_{12} denotes the Keccak-p[1600,12] permutation, σ denotes the input string, and Z denotes the key stream.

3.3 Ketje

Ketje is an authenticated encryption scheme that consists of 2 parts: The mode of operation *MonkeyWrap* and the Keccak-p permutation. While 4 different versions of Ketje have been proposed by the designers for the 4 different permutation sizes of 200, 400, 800 and 1600 bits, our practical evaluation is performed on Ketje Jr. The main use case of Ketje Jr is lightweight authenticated encryption for constraint devices. Hence, the permutation is based on Keccak-$p[200, n_r]$, meaning that only a rather small 200-bit state is used and the number of permutation rounds n_r is variable. Ketje Jr performs authenticated encryption with a 96-bit secret key and up to 86-bits of nonce. Different to Keccak and Keyak, in Ketje every call of the permutation is slightly twisted. The twisted permutation Keccak-p^* is an extended version of the standard permutation Keccak-p. It always starts with an additional call of π^{-1} and ends with an additional call to π. In the following we describe the *MonkeyWrap* mode of operation, as used in Ketje. Whenever we refer to Ketje we mean Ketje Jr.

Monkey Wrap. The *MonkeyWrap* mode defines how incoming messages are processed together with key, nonce, associated data and tag in Ketje.

The initialization of *MonkeyWrap* is called *Start* which is similar to the *Motorist* mode. First, key K and nonce N are XOR-ed into the zero-initialized state. Then 12 rounds of twisted Keccak-p^* permutation are performed.

The key stream generation *Step* is accomplished by performing duplexing calls, yet this time not the full width of the permutation is utilized, as illustrated in Fig. 2. Since the rate r of the permutation in Ketje is very small only a 1-round twisted Keccak-p^* permutation is needed in between *Step* calls.

Before the extraction of the tag starts, a 6-round twisted Keccak-p^* permutation is performed.

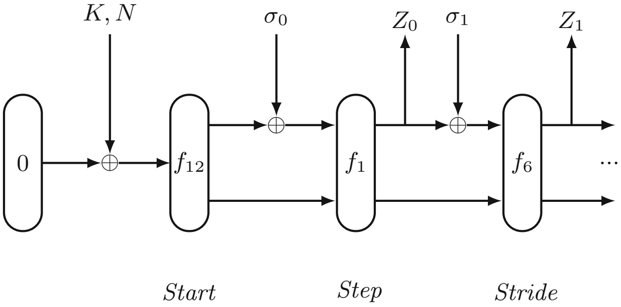

Fig. 2. Ketje Jr. f_{n_r} denotes the application of a n_r-round twisted Keccak-p^*[200] permutation, σ denotes the input string, and Z denotes the key stream.

4 Attack Strategy

In our attack, we target the decryption/verification of Lake Keyak and Ketje Jr ($\mathcal{D}(K, N, A, C, T)$). To be precise, we observe the behavior of the authenticated decryption of valid messages (N, A, C, T) in the presence of faults that are induced during the initialization phase. For both schemes, the initialization is the application of variants of Keccak-f to a state, which is composed out of the secret key K and a publicly known nonce N. If the fault induction affects and changes the outcome of this computation, also the value of the afterwards computed tag T will change compared to the value of the transmitted tag T and thus, the verification will fail. If the induced fault does not change the outcome of the initialization, the verification will succeed and the authenticated decryption will return a plaintext. Please note that the actual plaintext is not needed for the attack, we solely assume that the attacker is able to distinguish a failed verification from a successful one.

As shown in [16], inducing faults in multiple runs of the same computation with differing inputs, followed by a subsequent filtering for unaffected computations, most likely leads to biased distribution in the targeted intermediate value. In our case, unaffected computations (and thus ineffective faults) can be deduced from the condition that the verification succeeds. Hence, we assume that the attacker is able to affect one or multiple bits ($A_{\chi_2}[x, y, z]$) of the internal state before the application of χ in the 2nd round of the initialization, so that the distribution of these bits is non-uniform for the filtered inputs (N, A, C, T). More concretely, we assume that the attacker is able to collect several nonces N, which lead to one or multiple biased bits before the 2nd round χ-layer of the initialization.

Out of this knowledge, the attacker is able to extract information about the secret key. In the following section, we give a detailed description of how key recovery is achieved for Keyak. A very similar approach can then be used to perform key recovery for Ketje Jr. The major difference is the fact that a 200-bit permutation is used and hence bits of the equivalent key directly before the application of the 1st round χ-layer are guessed in the attack.

4.1 Involved Bits in Keyak

Information about the secret can be deduced by identifying key bits that are involved in the calculation of $A_{\chi_2}[x, y, z]$ and evaluate the value of $A_{\chi_2}[x, y, z]$ under every possible assignment of the key bits for every previously collected value of the nonce N. For the right key guess, we expect to observe the highest bias in the values of $A_{\chi_2}[x, y, z]$. But at first, we have to identify the involved bits.

First, we need to determine the bits at the input of the linear layer of the 2nd round, which are involved in the calculation of $A_{\chi_2}[x, y, z]$. The linear layer of one round of Keccak-$p[1600, 12]$ consists of the application of the single round functions θ, ρ, and π. The function π just swaps the words, so that

$$A_{\chi_2}[x, y, z] = A_{\pi_2}[(x + 3y) \bmod 5, x, z] \ .$$

The function ρ rotates each lane by a different offset $R[x, y]$. Hence,

$$A_{\chi_2}[x, y, z] = A_{\rho_2}[(x + 3y) \bmod 5, x, (z - R[(x + 3y) \bmod 5, x]) \bmod 64] \ .$$

Finally, θ computes its output by XOR-ing each bit with the parity of two columns in the array, thus, one bit $A_{\chi_2}[x, y, z]$ is the sum of 11 input bits to θ.

$$\begin{aligned} A_{\chi_2}[x, y, z] =& A_{\theta_2}[(x + 3y) \bmod 5, x, (z - R[(x + 3y) \bmod 5, x]) \bmod 64] \\ &\oplus \bigoplus_{y'=0}^{4} A_{\theta_2}[(x + 3y - 1) \bmod 5, y', (z - R[(x + 3y) \bmod 5, x]) \bmod 64] \\ &\oplus \bigoplus_{y'=0}^{4} A_{\theta_2}[(x + 3y + 1) \bmod 5, y', (z - R[(x + 3y) \bmod 5, x]-1) \bmod 64] \end{aligned}$$

Each of the 11 bits $A_{\theta_2}[x_i, y_i, z_i]$ can be calculated using three input bits to χ. Therefore,

$$\begin{aligned} A_{\theta_2}[x_i, y_i, z_i] =& A_{\chi_1}[x_i, y_i, z_i] \oplus \\ & ((A_{\chi_1}[(x_i + 1) \bmod 5, y_i, z_i] \oplus 1) \cdot A_{\chi_1}[(x_i + 2) \bmod 5, y_i, z_i]) \, . \end{aligned}$$

Note that two bits at the input of θ in the 2nd round needed in the calculation of $A_{\chi_2}[x, y, z]$ are adjacent bits of the same S-box, namely

$$A_{\theta_2}[(x + 3y) \bmod 5, x, (z - R[(x + 3y) \bmod 5, x]) \bmod 64]$$
$$A_{\theta_2}[(x - 3y - 1) \bmod 5, y, (z - R[(x + 3y) \bmod 5, x]) \bmod 64] \ .$$

As a consequence, only 31 bits of $A_{\chi_1}[x_j, y_j, z_j]$ are involved in the calculation of $A_{\chi_2}[x, y, z]$.

The bits at the input to the 1st round that are needed to compute the 31 bits $A_{\chi_1}[x_j, y_j, z_j]$ can be determined in a similar manner as done for the second round. However, doing so for general values of x and y gets a bit clumsy, hence, we focus on the restricted case of calculating $A_{\chi_2}[0, 0, 0]$. The necessary equations are given in Appendix B.

Determining the necessary bits to calculate $A_{\chi_2}[x, y, z]$ by hand is quite time consuming and also error prone. Thus, we have used a search tool [17], which has been developed to search for linear characteristics to identify the bits at the input of the Keccak-*f* permutation that are involved in the calculation of a certain $A_{\chi_2}[x, y, z]$. In Fig. 3, we give the involved bits for calculating $A_{\chi_2}[0, 0, 0]$. The figure represents one lane as hexadecimal value, where bits that are set to 1 are needed in the calculation of $A_{\chi_2}[0, 0, 0]$. A corresponding figure for Ketje Jr is given in Appendix A.

Input to	Bit positions
θ_1	C-62-C1---9C---1 8E-12----C3----C 6-1--45----E-3-4 -384983--118---6 1---4228184--181
	C-62-C1-189C---- 8E-12----C38---C 661--45----E-3-- -384982--118-3-6 1---5A28184--181
	D-62-C1---9C---- 8E212----C3----C 6-1--45---3E-3-- -384986--118---6 1---4228194--181
	C-62-C1---DC---- 8E-12----C34---C 6-11-45----E-3-- -3849C2--118---6 1-8-4228184--181
	C-624C1---9C---- CE-12----C3----C 6-1--45----E-3-8 -384982--118-1-6 1--44228184--181
χ_1	---------------1 8--------------1 8--------------1 8--------------- ---------------1
	---------------1 8--------------1 8--------------- 8--------------- ---------------1
	---------------1 8--------------1 8--------------- 8--------------- ---------------1
	---------------1 8--------------1 8--------------- 8--------------- ---------------1
	---------------1 8--------------1 8--------------- 8--------------- ---------------1
θ_2	---------------1 8--------------- ---------------- ---------------- ---------------1
	---------------- 8--------------- ---------------- ---------------- ---------------1
	---------------- 8--------------- ---------------- ---------------- ---------------1
	---------------- 8--------------- ---------------- ---------------- ---------------1
	---------------- 8--------------- ---------------- ---------------- ---------------1
χ_2	---------------1 ---------------- ---------------- ---------------- ----------------
	---------------- ---------------- ---------------- ---------------- ----------------
	---------------- ---------------- ---------------- ---------------- ----------------
	---------------- ---------------- ---------------- ---------------- ----------------
	---------------- ---------------- ---------------- ---------------- ----------------

Fig. 3. Bits involved in calculation of $A_{\chi_2}[0, 0, 0]$. The position of the 128-bit key is highlighted in gray. Zeros are replaced by - to improve readability.

4.2 Recovered Bits

In this section, we will discuss how much information on the key bits can be recovered by exploiting a bias in $A_{\chi_2}[x, y, z]$. For the sake of simplicity, we will stick to the example of $A_{\chi_2}[0, 0, 0]$. Bits having a gray background in Fig. 3 are bits that represent the 128 key bits. Hence, to compute $A_{\chi_2}[0, 0, 0]$, 25 bits of the key have to be guessed. However, from the equation given in Appendix B, we can see that only the 17 bits:

$$A_{\theta_1}[0,0,0]\ \ , A_{\theta_1}[0,0,18], A_{\theta_1}[0,0,20], A_{\theta_1}[0,0,23], A_{\theta_1}[0,0,36], A_{\theta_1}[0,0,43],$$
$$A_{\theta_1}[0,0,53], A_{\theta_1}[0,0,54], A_{\theta_1}[1,0,2]\ \ , A_{\theta_1}[1,0,20], A_{\theta_1}[1,0,21], A_{\theta_1}[1,0,27],$$
$$A_{\theta_1}[1,0,48], A_{\theta_1}[1,0,58], A_{\theta_1}[1,0,59], A_{\theta_1}[1,0,63], A_{\theta_1}[2,0,62]$$

can influence $A_{\chi_2}[0, 0, 0]$ in a non-linear manner, while the 8 bits:

$$A_{\theta_1}[0,0,19], A_{\theta_1}[0,0,42], A_{\theta_1}[0,0,49], A_{\theta_1}[1,0,3]\ \ , A_{\theta_1}[1,0,26], A_{\theta_1}[1,0,45],$$
$$A_{\theta_1}[1,0,57], A_{\theta_1}[2,0,61]$$

only have a linear influence.

As a consequence, we can at most uniquely identify the 17 bits that influence $A_{\chi_2}[0, 0, 0]$ in a non-linear way. For the 8-bits that influence $A_{\chi_2}[0, 0, 0]$ in a linear way, only their XOR-sum (parity) effects the value of $A_{\chi_2}[0, 0, 0]$. Since for 8 bits, half of the possible assignments have parity 0 and the other half has parity one, we get at least 2^7 key candidates that always lead to the same result. Please note that this is a rather simplistic evaluation and does not consider the dependencies of the non-linear bits and also the bits, which are used as nonce and constants. In fact, the key recovery depends on the value of these bits, since an unfortunate choices for the nonce can, for instance, lead to situations, where some S-boxes are linearized for some key bits, or some key bits are always blocked, so that they do not influence $A_{\chi_2}[0, 0, 0]$. For instance, let us have a look at the results of one of our concrete experiments given in Sect. 5. Instead of recovering 17 out of the 25 bits uniquely from 2^7 key candidates scoring best, we are able to recover 15 of the 25 bits uniquely out of 2^9 key candidates that score best.

5 Practical Evaluation

We now describe the practical evaluation of our attack on a microprocessor implementation. Although we have performed attacks on both Lake Keyak and Ketje Jr, we limit our description to Lake Keyak, since the attack procedure is similar for both schemes. We do, however, state the results for both schemes at the end of this section. We start this section by giving a quick overview of the attack procedure in Sect. 5.1. We then describe the hardware/software that we have used to perform our attack evaluation in Sect. 5.2. After that, we state requirements on a fault setup more generally in Sect. 5.3. Finally, we present the results of our fault attacks on Lake Keyak and Ketje Jr in Sect. 5.4.

5.1 Attack Procedure

As described in Sect. 4, our key recovery exploits the input of specific Keyak decryptions. We are interested in decryptions that have a bias in one or multiple bits of the Keccak state before χ in the 2^{nd} round. To achieve the required filtering of inputs we use statistical ineffective fault attacks (SIFA), as proposed in [16].

Before the attack we set the secret key of the microprocessor Keyak implementation to a constant and unknown value. During the attack we send inputs, consisting of random nonce and tag, to the microprocessor, induce a clock glitch with constant offset during the computation and observe the behavior. The tag verification is used to detect whether or not an induced fault was ineffective.

5.2 Attack Setup

The practical evaluation of our fault attack was done on an 8-bit Xmega 128D4 microprocessor. The attacked software implementation of Lake Keyak consists of two parts. The first part is a C implementation of the *Motorist* mode of operation. The second part is a fast 8-bit AVR optimized assembler implementation of the Keccak permutation. Both implementations are taken from the Keccak Code Package [8] and therefore represent a good target software implementation for our practical evaluation. The clock signal of the microprocessor is generated by a Spartan-6 FPGA running at 12 MHz. We additionally use this FPGA for the insertion of glitches onto the clock signal. The insertion of clock glitches is achieved by XOR-ing an additional fast clock edge onto the original clock signal at a specified time. By doing so, we can violate the critical path to force undefined behavior of the microprocessor.

In our practical evaluation we can force strong biases in virtually every state bit that is affected by χ, however only in blocks of 8 bits at a time (which is not surprising on a 8-bit architecture). We suspect that our glitch does skip one of the XOR instructions in the bit-sliced χ implementation, but we cannot say for sure though.

5.3 Attack Setup - Requirements

As we use SIFA [16], the requirements we have on the locality and especially the effect of the fault are quite relaxed. Basically, we only need some sort of bias in any bit at the input of χ in the 2^{nd} round. This can be achieved by e.g. faulting instructions in χ, slightly before χ, or by directly faulting registers using lasers. In the case of AES, such fault inductions have already been demonstrated for multiple microprocessors and even for hardware co-processors [15,16]. One way to find a suitable glitch location in practice would be to estimate the clock cycles until the targeted operation is executed. Hence, in our scenario, one can estimate the time frame of the 2^{nd} round and try to induce a glitch in several different clock cycles towards the end of that round.

5.4 Results

Keyak. As already mentioned in Sect. 4.2, when getting a bias in the bit $A_{\chi_2}[0,0,0]$ located at the input of the 2nd round χ-layer, 25 bits of the key are involved in its calculation. In our attack, we guess these 25 bits and evaluate the bias in $A_{\chi_2}[0,0,0]$ for each key guess. Since some of the guessed key bits only influence $A_{\chi_2}[0,0,0]$ in a linear manner, we get several equivalent key candidates having the same bias. As a consequence, Fig. 4 shows the advantage in bits the attacker gets from guessing key candidates down to a bias which also the correct key guess over just randomly guessing the key, which is $\log_2(\#\text{total keys}) - \log_2(\#\text{candidate keys})$.

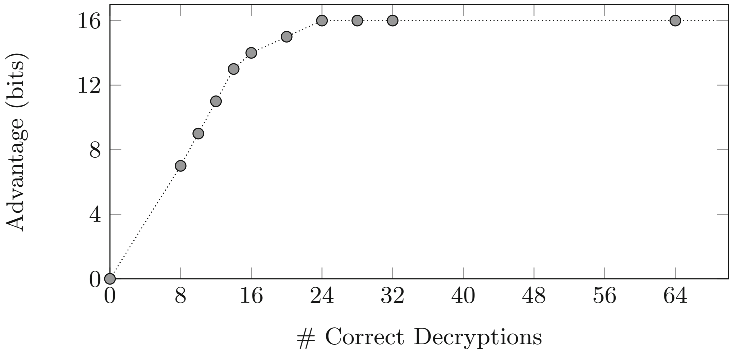

Fig. 4. Attack on Keyak. Advantage in bits when targeting $A_{\chi_2}[0,0,0]$ and guessing the associated 25 bits of the 128-bit key.

As shown in Fig. 4, 24 inputs of such unaffected decryptions are necessary to get a maximum advantage of 16 bits. In our case, we get 2^9 keys ranked top that have the same bias (not considering its sign). From those 2^9 keys, the values of 15 key bits can be uniquely determined. Due to the architecture of the implementation, we do not only get a bias in one bit, but one byte. By combining this information, we can uniquely determine 82-bits of the key.

In our attack setup, we are able to perform about 20 faulted decryptions per second. According to the practical evaluation, in about 1 out of 250 decryptions the induced fault is ineffective. The total time it took us to gather the required amount of inputs is roughly 5 min.

Ketje. In the attack on Ketje Jr we use the same fault location as in the attack on Lake Keyak. This is however not strictly necessary. Even though both schemes use variants of Keccak-f during initialization, the influence of key bits on one of our biased bits before χ in the 2nd round is quite different, mainly due to the fact that the lane sizes are different (see Fig. 6). In contrast to Lake Keyak, in Ketje Jr nearly all key bits influence each of our biased bits, most of the time in a linear way.

Hence, for Ketje Jr we instead guess the 200-bit equivalent key before χ in the 1$^{\text{st}}$ round (i.e. after the first linear layer). By doing so we can reduce the dependency on the equivalent key to 31 bit and guessing becomes feasible in practice.

In our attack setup we can recover about 19 bits of the equivalent key that correspond to one biased bit in about 10 h using a single thread on an Intel Xeon CPU. Note that this time can be significantly improved, since we used for our evaluation purposes just the unoptimized reference implementation. Furthermore, the task of key guessing can be parallelized trivially. If we parallelize the computations for e.g. the 8 bits that were affected by our fault induction we can recover 152 bits of the equivalent key in the same amount of time. The remaining bits can be determined either by brute-force or repeating key recovery for a different fault location.

In total, again 24 inputs of unaffected decryptions are necessary for key recovery as shown in Fig. 5. The total time it took us to gather the required amount of inputs is below 5 min. Hence, the time complexity of entire attack is dominated by the key guessing and was performed in about 10 h.

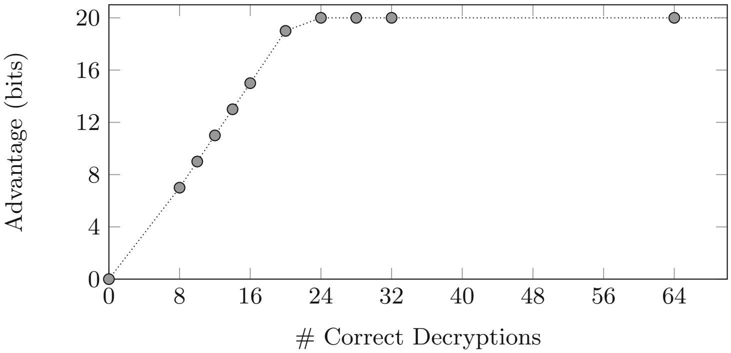

Fig. 5. Attack on Ketje. Advantage in bits when targeting $A_{\chi_2}[0, 0, 0]$ and guessing the associated 31 bits of the 200-bit equivalent key.

6 Conclusion

In this work, we present the first fault attacks targeting a broad range of nonce-based authenticated encryption schemes. While fault attacks on authenticated encryption have already been shown at Asiacrypt 2016 [15], this attack is mostly limited to schemes that additionally feature a final key addition and thus, is not directly applicable to most sponge-based or stream cipher-based constructions. We close this gap and show attacks based on SIFA [16], which are in principle applicable to most nonce-based authenticated encryption schemes that perform some sort of initialization where the nonce (or an other publicly known input) is mixed with the secret key. Since we only need to know whether a fault induction was ineffective or not, attacking the decryption function of authenticated encryption schemes gives us a perfect oracle. Our attack evaluation is focused on

Keyak and Ketje, however, we conjecture that our attack can also be adopted to other schemes like the CAESAR finalists ACORN, AEGIS, Ascon, MORUS, etc. in a rather straight-forward way.

SIFA is resistant to popular fault countermeasures like double-execution and infection-based countermeasures as shown in [16]. Even additional masking does not preclude this attack vector [14]. The key recovery is capable of dealing with an arbitrary amount of noise (however requiring more faulted decryptions) that might arise due to possibly imperfect fault inductions. The effort required to perform our attack is rather low. We neither require perfectly timed faults nor precise knowledge about the effect of the induced fault. In our fault setup we are able to collect enough material for key recovery within 5 min. The actual key recovery for Keyak and Ketje is easily parallelizable and takes about 30 min and 10 h, respectively. The hardware cost of the attack setup does not exceed 300$.

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

A Bits Involved in the Calculation for Ketje Jr

Input to	Bit positions
θ_1	ff bf 7f bf fb
	fe bf 7f bf fb
	fe bf 7f ff fb
	fe bf 7f bf fb
	fe ff 7f bf ff
χ_1	-1 81 81 8- -1
	-1 81 8- 8- -1
	-1 81 8- 8- -1
	-1 81 8- 8- -1
	-1 81 8- 8- -1
θ_2	-1 8- -- -- -1
	-- 8- -- -- -1
	-- 8- -- -- -1
	-- 8- -- -- -1
	-- 8- -- -- -1
χ_2	-1 -- -- -- --
	-- -- -- -- --
	-- -- -- -- --
	-- -- -- -- --
	-- -- -- -- --

Fig. 6. Bits involved in calculation of $A_{\chi_2}[0,0,0]$. Zeros are replaced by - to improve readability.

B Equations to Calculate $A_{\chi_2}[0, 0, 0]$

$$A_{\chi_2}[0,0,0] = A_{\theta_2}[0,0,0] \oplus \bigoplus_{y'=0}^{4} A_{\theta_2}[4,y',0] \oplus \bigoplus_{y'=0}^{4} A_{\theta_2}[1,y',63]$$

$$\begin{aligned}
A_{\theta_2}[0,0,0] &= A_{\chi_1}[0,0,0] \oplus ((A_{\chi_1}[1,0,0] \oplus 1) \cdot A_{\chi_1}[2,0,0]) \\
A_{\theta_2}[4,0,0] &= A_{\chi_1}[4,0,0] \oplus ((A_{\chi_1}[0,0,0] \oplus 1) \cdot A_{\chi_1}[1,0,0]) \\
A_{\theta_2}[4,1,0] &= A_{\chi_1}[4,1,0] \oplus ((A_{\chi_1}[0,1,0] \oplus 1) \cdot A_{\chi_1}[1,1,0]) \\
A_{\theta_2}[4,2,0] &= A_{\chi_1}[4,2,0] \oplus ((A_{\chi_1}[0,2,0] \oplus 1) \cdot A_{\chi_1}[1,2,0]) \\
A_{\theta_2}[4,3,0] &= A_{\chi_1}[4,3,0] \oplus ((A_{\chi_1}[0,3,0] \oplus 1) \cdot A_{\chi_1}[1,3,0]) \\
A_{\theta_2}[4,4,0] &= A_{\chi_1}[4,4,0] \oplus ((A_{\chi_1}[0,4,0] \oplus 1) \cdot A_{\chi_1}[1,4,0])
\end{aligned}$$

$$\begin{aligned}
A_{\theta_2}[1,0,63] &= A_{\chi_1}[1,0,63] \oplus ((A_{\chi_1}[2,0,63] \oplus 1) \cdot A_{\chi_1}[3,0,63]) \\
A_{\theta_2}[1,1,63] &= A_{\chi_1}[1,1,63] \oplus ((A_{\chi_1}[2,1,63] \oplus 1) \cdot A_{\chi_1}[3,1,63]) \\
A_{\theta_2}[1,2,63] &= A_{\chi_1}[1,2,63] \oplus ((A_{\chi_1}[2,2,63] \oplus 1) \cdot A_{\chi_1}[3,2,63]) \\
A_{\theta_2}[1,3,63] &= A_{\chi_1}[1,3,63] \oplus ((A_{\chi_1}[2,3,63] \oplus 1) \cdot A_{\chi_1}[3,3,63]) \\
A_{\theta_2}[1,4,63] &= A_{\chi_1}[1,4,63] \oplus ((A_{\chi_1}[2,4,63] \oplus 1) \cdot A_{\chi_1}[3,4,63])
\end{aligned}$$

$$\begin{aligned}
A_{\chi_1}[0,0,0] &= A_{\theta_1}[0,0,0] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',0] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',63] \\
A_{\chi_1}[1,0,0] &= A_{\theta_1}[1,1,20] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',20] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',19] \\
A_{\chi_1}[2,0,0] &= A_{\theta_1}[2,2,21] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',21] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',20] \\
A_{\chi_1}[4,0,0] &= A_{\theta_1}[4,4,50] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',50] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',49]
\end{aligned}$$

$$A_{\chi_1}[4,1,0] = A_{\theta_1}[2,4,3] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',3] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',2]$$

$$A_{\chi_1}[0,1,0] = A_{\theta_1}[3,0,36] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',36] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',35]$$

$$A_{\chi_1}[1,1,0] = A_{\theta_1}[4,1,44] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',44] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',43]$$

$$A_{\chi_1}[4,2,0] = A_{\theta_1}[0,4,46] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',46] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',45]$$

$$A_{\chi_1}[0,2,0] = A_{\theta_1}[1,0,63] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',63] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',62]$$

$$A_{\chi_1}[1,2,0] = A_{\theta_1}[2,1,58] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',58] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',57]$$

$$A_{\chi_1}[4,3,0] = A_{\theta_1}[3,4,8] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',8] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',7]$$

$$A_{\chi_1}[0,3,0] = A_{\theta_1}[4,0,37] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',37] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',36]$$

$$A_{\chi_1}[1,3,0] = A_{\theta_1}[0,1,28] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',28] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',27]$$

$$A_{\chi_1}[4,4,0] = A_{\theta_1}[1,4,62] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',62] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',61]$$

$$A_{\chi_1}[0,4,0] = A_{\theta_1}[2,0,2] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',2] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',1]$$

$$A_{\chi_1}[1,4,0] = A_{\theta_1}[3,1,9] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',9] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',8]$$

$$A_{\chi_1}[1,0,63] = A_{\theta_1}[1,1,19] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',19] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',18]$$

$$A_{\chi_1}[2,0,63] = A_{\theta_1}[2,2,20] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',20] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',19]$$

$$A_{\chi_1}[3,0,63] = A_{\theta_1}[3,3,42] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',42] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',41]$$

$$A_{\chi_1}[1,1,63] = A_{\theta_1}[4,1,43] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',43] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',42]$$

$$A_{\chi_1}[2,1,63] = A_{\theta_1}[0,2,60] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',60] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',59]$$

$$A_{\chi_1}[3,1,63] = A_{\theta_1}[1,3,18] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',18] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',17]$$

$$A_{\chi_1}[1,2,63] = A_{\theta_1}[2,1,57] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',57] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',56]$$

$$A_{\chi_1}[2,2,63] = A_{\theta_1}[3,2,38] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',38] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',37]$$

$$A_{\chi_1}[3,2,63] = A_{\theta_1}[4,3,55] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',55] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',54]$$

$$A_{\chi_1}[1,3,63] = A_{\theta_1}[0,1,27] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',27] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',26]$$

$$A_{\chi_1}[2,3,63] = A_{\theta_1}[1,2,53] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',53] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',52]$$

$$A_{\chi_1}[3,3,63] = A_{\theta_1}[2,3,48] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',48] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',47]$$

$$A_{\chi_1}[1,4,63] = A_{\theta_1}[3,1,8] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[2,y',8] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',7]$$

$$A_{\chi_1}[2,4,63] = A_{\theta_1}[4,2,24] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[3,y',24] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[0,y',23]$$

$$A_{\chi_1}[3,4,63] = A_{\theta_1}[0,3,22] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[4,y',22] \oplus \bigoplus_{y'=0}^{4} A_{\theta_1}[1,y',21]$$

References

1. Anceau, S., Bleuet, P., Clédière, J., Maingault, L., Rainard, J., Tucoulou, R.: Nanofocused X-ray beam to reprogram secure circuits. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 175–188. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_9
2. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. Proc. IEEE **94**(2), 370–382 (2006). https://doi.org/10.1109/JPROC.2005.862424
3. Bellare, M., Rogaway, P., Wagner, D.A.: EAX: a conventional authenticated-encryption mode. Cryptology ePrint Archive, Report 2003/069 (2003). http://eprint.iacr.org/2003/069
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak SHA-3 submission (Version 3.0) (2011). http://keccak.noekeon.org/Keccak-submission-3.pdf
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28496-0_19
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: Ketje v2. https://keccak.team/files/Ketjev2-doc2.0.pdf
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: Keyak v2. https://keccak.team/files/Keyakv2-doc2.2.pdf
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: Keccak code package. https://github.com/gvanas/KeccakCodePackage. Accessed 05 Dec 2017
9. 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). https://doi.org/10.1007/3-540-44598-6_8
10. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052259
11. 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). https://doi.org/10.1007/3-540-69053-0_4
12. CAESAR committee: CAESAR: Competition for authenticated encryption: Security, applicability, and robustness (2014). http://competitions.cr.yp.to/caesar.html
13. Clavier, C.: Secret external encodings do not prevent transient fault analysis. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 181–194. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_13

14. Dobraunig, C., Eichlseder, M., Gross, H., Mangard, S., Mendel, F., Primas, R.: Statistical ineffective fault attacks on masked AES with fault countermeasures. Cryptology ePrint Archive, Report 2018/357 (2018). https://eprint.iacr.org/2018/357
15. Dobraunig, C., Eichlseder, M., Korak, T., Lomné, V., Mendel, F.: Statistical fault attacks on nonce-based authenticated encryption schemes. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 369–395. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_14
16. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. In: IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2018, no. 3, pp. 547–572, August 2018. https://tches.iacr.org/index.php/TCHES/article/view/7286
17. Dobraunig, C., Eichlseder, M., Mendel, F.: Heuristic tool for linear cryptanalysis with applications to CAESAR candidates. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 490–509. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_20
18. Fuhr, T., Jaulmes, É., Lomné, V., Thillard, A.: Fault attacks on AES with faulty ciphertexts only. In: Fischer, W., Schmidt, J.M. (eds.) FDTC 2013, pp. 108–118. IEEE Computer Society (2013)
19. Maurine, P.: Techniques for EM fault injection: equipments and experimental results. In: Bertoni, G., Gierlichs, B. (eds.) FDTC 2012, pp. 3–4. IEEE Computer Society (2012)
20. 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). https://doi.org/10.1007/978-3-540-30556-9_27
21. McKay, K.A., Bassham, L., Turan, M.S., Mouha, N.: NISTIR 8114: report on lightweight cryptography (2017). https://doi.org/10.6028/NIST.IR.8114
22. Mennink, B., Reyhanitabar, R., Vizár, D.: Security of full-state keyed sponge and duplex: applications to authenticated encryption. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 465–489. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_19
23. National Institute of Standards and Technology: FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Information Processing Standards Publication 202, U.S. Department of Commerce, August 2015. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
24. National Institute of Standards and Technology: DRAFT submissionrequirements and evaluation criteria for the lightweight cryptographystandardization process (2018). https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/Draft-LWC-Submission-Requirements-April2018.pdf
25. 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). https://doi.org/10.1007/978-3-540-45238-6_7
26. Rogaway, P.: Authenticated-encryption with associated-data. In: CCS 2002, pp. 98–107. ACM (2002)
27. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) CCS 2001, pp. 196–205. ACM (2001)

28. Ronen, E., Shamir, A., Weingarten, A.O., O'Flynn, C.: IoT goes nuclear: creating a ZigBee chain reaction. In: SP 2017, pp. 195–212. IEEE Computer Society (2017)
29. 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, Cham (2016). https://doi.org/10.1007/978-3-319-31301-6_24
30. Saha, D., Chowdhury, D.R.: EnCounter: on breaking the nonce barrier in differential fault analysis with a case-study on PAEQ. In: Gierlichs, B., Poschmann, A. (eds.) CHES 2016. LNCS, vol. 9813, pp. 581–601. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_28
31. 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, Cham (2014). https://doi.org/10.1007/978-3-319-13039-2_12
32. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_2
33. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM) (2003)
34. Yen, S.M., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Trans. Comput. **49**(9), 967–970 (2000)