2018 IEEE European Symposium on Security and Privacy

Attacking Deterministic Signature Schemes Using Fault Attacks

Damian Poddebniak*, Juraj Somorovsky†, Sebastian Schinzel*,
Manfred Lochter‡ and Paul Rösler†
*Münster University of Applied Sciences
poddebniak@fh-muenster.de, schinzel@fh-muenster.de
†Chair for Network and Data Security, Ruhr University Bochum
juraj.somorovsky@rub.de, paul.roesler@rub.de
‡Federal Office for Information Security
manfred.lochter@bsi.bund.de

***Abstract*—Many digital signature schemes rely on random numbers that are unique and non-predictable per signature. Failures of random number generators may have catastrophic effects such as compromising private signature keys. In recent years, many widely-used cryptographic technologies adopted deterministic signature schemes because they are presumed to be safer to implement.**

In this paper, we analyze the security of deterministic ECDSA and EdDSA signature schemes and show that the elimination of random number generators in these schemes enables new kinds of fault attacks. We formalize these attacks and introduce practical attack scenarios against EdDSA using the Rowhammer fault attack. EdDSA is used in many widely used protocols such as TLS, SSH, and IPSec, and we show that these protocols are not vulnerable to our attack. We formalize the necessary requirements of protocols using these deterministic signature schemes to be vulnerable, and discuss mitigation strategies and their effect on fault attacks against deterministic signature schemes.

1. Introduction

The Digital Signature Standard (*DSS*) describes a family of cryptographic methods such as the Digital Signature Algorithm (*DSA*) for digitally signing content. The Elliptic Curve Digital Signature Algorithm (*ECDSA*) is a variant of DSA using elliptic curve cryptography. Both depend on a cryptographic random value r, which must never repeat under different messages. This value is also known as a digital signature's ephemeral key, session key, or secret nonce, underlying the fact that r also needs to be confidential. We call it *nonce* throughout the paper, despite the different standards having different names for it. This is because in this paper we focus on the "number used once" property of r and study the security consequences of nonce reuse in deterministic signature schemes.

RNG Failures. Using a given nonce only for one message is crucial; if a nonce is reused for two different messages, an attacker can trivially calculate the private key for generating signatures. This is due to the underlying primitive – an interactive zero-knowledge proof [1] – requiring distinct nonces in order to be secure. Thus, the developers of cryptographic implementations have to ensure that nonces are never reused. A common way to achieve this is by using a high entropy RNG to generate fresh nonces for each signature. A prominent example where the reuse of nonces led to a compromise of signature keys was the Sony Playstation 3 [2], where the hacker group *fail0verflow* showed that Sony was reusing the same nonce for every digitally signed game. The members then calculated the private key and create valid signatures for arbitrary files including pirated games or Linux applications. Another security breach that resulted out of insufficient entropy with the nonce of ECDSA signatures happened in a Bitcoin Android app [3]. As a result, attackers stole Bitcoins worth several thousand dollars.

Deterministic signature schemes. To cope with this well-known pitfall in implementing DSA and ECDSA, a nonce may be calculated deterministically, as initially proposed by Barwoord[1] and Wigley.[2] The idea is to calculate the nonce from the to-be-signed message M and the private signing key. The advantage here is that there is no need for generating fresh random numbers for each signature. Edwards curve Digital Signature Algorithm (EdDSA) follows a similar approach and uses the hash of the private key and the message M as a nonce. Thus, any change of M results in a new nonce. A deterministic variant of DSA and ECDSA was described by Pornin [4].

Fault attacks and the case of Rowhammer. Fault attacks are a well-known technique in cryptanalysis that induces errors during cryptographic computations. Traditionally, fault attacks require physical access to the hardware to induce faults, for example, by using electrical glitching with power disturbances, thermal fluctuations or emitting radiation to the memory chips. Rowhammer is a recently found attack technique that allows inducing bit-flips in DRAM memory chips on commodity computers without physical access to

1. https://groups.google.com/forum/#!msg/sci.crypt/SalLSLBBTe4/xtYNGDe6irIJ
2. https://groups.google.com/forum/#!msg/sci.crypt/3g8DnnEkv5A/a26mLrwfjiMJ

DOI 10.1109/EuroSP.2018.00031

the device. The idea is to very quickly and repeatedly read DRAM rows to increase internal cell leakage. When a cell leakage is raised to a level such that a cell is unable to maintain a charge for a specific time frame, it will lose its data. Because read access is sufficient and nearby memory regions are affected, Rowhammer enables an attacker to flip bits in memory areas where they should not have access to. This allows bypassing typical memory protection mechanisms.

Fault attacks on deterministic signatures. In this paper, we analyze the effects of fault attacks on deterministic cryptographic signature schemes. We analyze signature schemes under the assumption that an attacker can induce semi-targeted bit flips in different intermediate signature values. Our main observation is as follows: Deterministic signatures produce the nonce from a private key and the message M. Thus, the signing application needs to read M twice: once for producing the nonce and again for calculating the digital signature. If an attacker is able to change M to $M^{\lightning}$ just after the nonce was calculated but before the actual signing, then $M^{\lightning}$ is signed with a nonce for M. We show that it is possible to achieve such a situation by using bit-flips, induced by double-sided as well as single-sided Rowhammer attacks. The evaluation results show that this presents a new attack against EdDSA that allows the attacker to retrieve the private signature key. This stands in contrast to the current state of the art where EdDSA was found to be resilient to fault attacks [5]. Our work shows the importance of considering fault attacks on deterministic signature algorithms in general and specifically on EdDSA. This is underlined by a large number of recent cryptographic protocols implementing these signature algorithms [6], [7], [8], [9], [10].

Contributions. We make the following contributions:

- We describe a new fault attack against deterministic signature schemes that allows the attacker to retrieve the private signature key.
- We formally analyze the properties of cryptographic primitives, which are the reason for the vulnerability. We conclude that our attack is generally applicable to a well-known class of deterministic signatures derived via the Fiat-Shamir transform [11].
- We investigate real-world protocols and find one potentially vulnerable to our attacks: Online Certificate Status Protocol (OCSP).
- We evaluate the practicality of our attack and implement a realistic attack against EdDSA, utilizing double-sided and single-sided Rowhammer.
- We propose and discuss possible countermeasures and extensions to EdDSA in order to counter our attack.

2. Cryptographic Background

In this section, we briefly introduce Elliptic Curve Cryptography (ECC) and outline two signature algorithms, ECDSA (both the non-deterministic and the deterministic variant) and EdDSA.

Throughout this paper, we use the following notation: M is a plaintext message. H denotes a hash function. G denotes the base point of an elliptic curve E which is constructed over the finite field $\mathbb{F}_p$. f is a function that returns the x-coordinate of a point. $[d]P$ denotes a point multiplication of point P with a scalar d. If a hash over a value is computed, we assume a correct encoding as described in the relevant standard or literature [12]. $||$ denotes concatenation of bytes. Random sampling from a set or the return value of a probabilistic algorithm is denoted by $\xleftarrow{\$}$.

2.1. Elliptic Curve Cryptography

We are mainly interested in elliptic curves E over finite prime fields $\mathbb{F}_p$ with $p > 3$. Such curves can be described in the *short Weierstrass form*

$$E : y^2 = x^3 + A_W x + B_W.$$

Other representations of elliptic curves exist and are widely used in implementations; one being the Edwards form of elliptic curves.

The set of points on an elliptic curve carries a natural abelian group law which we will write additively. By $[d]P$ we denote the sum $P + \cdots + P$ (d times).

We fix a base point G of prime order q in $E(\mathbb{F}_p)$. Typically, cryptographic algorithms based on elliptic curves compute $[d]P$ for a secret scalar d and $P \in \langle G \rangle$. The security of the ECC algorithm relies on the presumed difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP) in $\langle G \rangle$. The best-known algorithms to tackle the ECDLP have a complexity of $O(\sqrt{q})$. Therefore q (and thus p) should be at least 256 bit primes.

In standard applications of ECC, the point multiplication described above is implemented in a randomized way to counter side-channel attacks. For example, one computes $[d + \lambda q]P = [d]P$ such that an attacker cannot recover d (e.g., by using an electromagnetic trace). This countermeasure is also called blinding.

2.2. ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a signature standard building upon elliptic curve cryptography.

Parameters. ECDSA's *domain parameters* are given by $(H, \mathbb{F}_p, E, q, G)$ with H being a hash function, E being an elliptic curve over the finite field $\mathbb{F}_p$, and G being a point in $E(\mathbb{F}_p)$ with prime order q. The public/private key pair is given by (A, a) with the private key $a \xleftarrow{\$} \mathbb{F}_p$ and the public key $A = [a]G$.

Signing. To sign a message M the signer generates a random number $r \xleftarrow{\$} \{1, \ldots, q-1\}$ and computes:

$$R = f([r]G) \mod q \tag{1}$$

$$s = (H(M) + aR)/r \mod q \tag{2}$$

The pair (R, s) is a signature for M, if both R and s are non-zero.

Verification. (R, s) is accepted as signature for M if

$$R = f([H(M)/s]G + [R/s]A). \tag{3}$$

Note that there are many sophisticated attacks on randomized signatures. An overview of attacks and countermeasures is given in [13].

2.3. Deterministic Signature Schemes

Securely implementing elliptic curve algorithms is often difficult due to many pitfalls and can result in critical security flaws [14], [15], [16]. Some of these pitfalls emerge from the used curve (i.e., incorrect point addition or missed point membership checks) and some due to difficulties in a signature scheme (i.e., lack of entropy in nonce generation). The fragility of ECDSA allows an attacker to learn the ECDSA private key if two different messages M, M' are signed using the same nonce r. *Deterministic signature schemes* were developed to avoid this pitfall by eliminating the need for random numbers during signature generation.

2.3.1. Deterministic ECDSA. Deterministic ECDSA [4] uses the same parameters and procedures as described in the previous section. The only difference is that instead of generating the nonce r at random, r is derived deterministically from the message M by using a nested HMAC construction with fixed key values (HMAC_DRBG). The full description of the algorithm is not needed to state the results of this paper and can be found in sec. 3.2 of [4].

The parameters and algorithms for generation of the pair (R, s) are the same as described in the previous section. For our paper, it is important to note that the signature generation results in equal signature outputs if the same message and private key are used.

Note that Pornin explicitly mentions that side-channel attacks are not taken into account, nor are fault attacks mentioned [4].

2.3.2. EdDSA. The *Edwards Digital Signature Algorithm (EdDSA)* is a digital signature scheme with a focus on simple implementation and high-performance [17]. The specifics of EdDSA include the choice of an *Edwards curve* [18] (which facilitates curve arithmetic), a deterministic nonce generation (to avoid forgery due to implementation flaws and eliminate the need for a reliable source of entropy), and avoidance of secret branch-conditions or lookup-indices (to prohibit side-channel attacks like cache- or timing-attacks) [17]. The name *Ed25519* is used to describe an instance of EdDSA with a specific set of parameters, specifically with a twisted Edwards curve birationally equivalent to Curve25519 [17].

Parameters. EdDSA, as initially proposed in [17], has the following parameters: $b \in \mathbb{N}$ with $b \geq 10$, a hash function H with $2b$-bit output, a prime power $p \equiv 1 \pmod 4$, an encoding of elements of the finite field $\mathbb{F}_p$ with $b-1$ bits, a non-square element $d \in \mathbb{F}_p$, a prime q with $2^{b-4} \leq q \leq 2^{b-3}$, and an element $G \neq (0, 1)$ such that $[q]G = (0, 1)$ [17]. The elliptic curve E is given as

$$E = \left\{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : -x^2 + y^2 = 1 + dx^2y^2\right\} \tag{4}$$

with the complete addition law being

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 + x_1x_2}{1 - dx_1x_2y_1y_2}\right) \tag{5}$$

An EdDSA secret key is a randomly generated string k with b bits. From the secret key obtain the hash $(h_0, h_1, \ldots, h_{2b-1}) = H(k)$ and calculate

$$a = 2^{b-2} + \sum_{3 \leq i < b-2} 2^i h_i \tag{6}$$

The public key is $A = [a]G$.

Signing. The signature of a message M is a pair (R, s) with:

$$r = H(h_b, \ldots, h_{2b-1}, M) \tag{7}$$

$$R = [r]G \tag{8}$$

$$s = (r + H(R, A, M)\, a) \mod q \tag{9}$$

Please note that for the simplicity we omit elliptic curve point encoding in our description.

Verification. Given the signature (R, s), the message M, and the public key A, check if

$$[s]G = R + [H(R, A, M)]A \tag{10}$$

holds. The signature is valid if the equation holds and no errors occurred during encoding/parsing of the values.

PureEdDSA and HashedEdDSA. One of the parameters of EdDSA is the pre-hash function *PH* that is applied to M before signing and whose output is then signed. If PH is the identity function (i.e., $PH(M) = M$), the signature algorithm signs the full M. This is called *PureEdDSA*. If PH uses a collision resistant hashing function like SHA-512, then the SHA-512 hash of M is signed instead of M directly. This is called *HashedEdDSA*. Bernstein et al. recommend PureEdDSA [19].

3. Fault Injection Background

The effects of faults on electronic systems have been studied for over 40 years. Since then, various forms of *fault injections* such as varying the voltage supply, casting high temperatures on hardware, using x-rays, etc. have been applied to real hardware [20]. Due to the need of physical access to a device, fault injections were mainly a threat to *tamper-proof hardware* like *smart cards* or *hardware security modules*. This has changed with the appearance of *Rowhammer*, a hardware fault affecting a computer's main memory [21]. In 2014 Kim et al. showed that malicious software can utilize *main memory hammering* to induce bit flips in nearby memory regions, bypassing the hardware's memory protection [21]. With Rowhammer, faults may be

injected remotely and strictly using software, making it a novel and very powerful form of fault injection.

While we focus in this paper on Rowhammer, we stress that our findings will work with other "faulting primitives".

3.1. Rowhammer

Main memory is composed of multiple *memory Integrated Circuits (ICs)* packed on a *Dual Inline Memory Module (DIMM)*. Although the form factor may differ, the memory technology used in each IC is *Dynamic Random Access Memory (DRAM)*. DRAM is used because each *cell* (i.e., each bit) is made up of a relative simple circuitry mainly containing a *transistor* and a *capacitor* (see Figure 1 (d)).

3.1.1. Accessing Data. DRAM cells are not directly accessible and each access must be served through a bank's *row buffer* (Figure 1 (c)) [21]. Data is accessed in three steps: (1) *Opening* a row, which transfers the current of the cells in one row into the bank's row buffer, (2) invoking read or write operations on the row buffer and (3) *closing* the row by transferring the row buffer's current back into the cells [21].

3.1.2. Refreshing Data. Due to internal *current leakage*, DRAM cells have a very limited *retention time* [21], [22]. This means that each cell will slowly leak its current and eventually lose its state when it is not periodically refreshed – hence the term "dynamic memory". DRAM memory controllers thus guarantee that each cell is refreshed at least once in a 64-millisecond time frame. This has two implications: (1) the memory controller must constantly refresh the current in each cell and (2) each cell must be guaranteed to keep its state for at least 64 ms in order to avoid data loss.

3.1.3. Disturbance Errors. As the proximity between each cell increases, *disturbance errors* become more likely. As a result, cells may leak their charge at an accelerated rate when inferring with other electrical components. When the cell leakage is raised to a level such that a cell can't keep its charge for 64 ms, it will lose its data [21]. This physical effect is the cause of the Rowhammer bug and probably known since 2012 [21], [23], [24], [25], [26], [27], [28], [29].

3.2. Hammering

Kim et al. demonstrated how bit flips can be triggered by using software. This is done by repeatedly activating two or more rows in a single bank while bypassing the CPU cache via cache-flush instructions [21]. There are two possible variants of those activations, namely single- and double-sided hammering. For both variants, an attacker must trigger row activation commands to one particular DRAM bank. For double sided hammering, the two selected rows must additionally enclose one victim row to intensify the

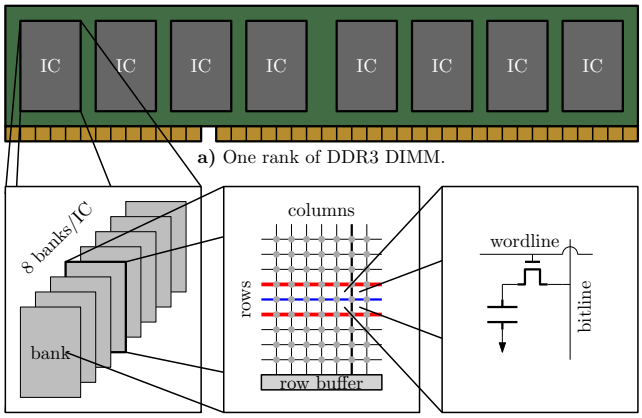

Figure 1. Simplified topology of DRAM organization.

leakage. Although, memory banks were reported to be vulnerable against both variants, double-sided hammering is more effective and often reported as the only way to trigger bit flips in reasonable time [30].

3.3. Memory Optimization Techniques

Rowhammer attackers face basically two challenges: (1) bypassing the CPU cache and (2) finding appropriate rows to hammer. The first challenge can be solved by utilizing special CPU instructions, like `clflush` and non-temporal instructions, or by crafting memory access patterns for fast cache eviction [21], [31], [32]. However, based on the attacker model, it is not immediately clear how the appropriate rows can be found.

With "Flip Feng Shui", Razavi et al. demonstrated how certain *memory optimizations* can be exploited for precise Rowhammer attacks [33]. A machine may deploy memory optimization strategies to reduce its memory footprint and increase system performance. Two features facilitating Rowhammer attacks are in particular *memory deduplication* (discussed in form of Linux' *Kernel Samepage Merging*) and *huge pages* (discussed in form of Linux' *Transparent Huge Pages*). Both technologies combined allow for controlled bit flips across virtual machine boundaries [33].

3.3.1. Kernel Samepage Merging. *Kernel Samepage Merging (KSM)* is a memory optimization technique designed to reduce the overall memory consumption of a system. KSM *deduplicates* memory by identifying and merging memory pages with the same content, such that multiple processes can transparently share the same physical memory. Since a page may be writable a *Copy-on-Write (CoW)* mechanism is introduced such that a page is unmerged prior to a write operation [34].

The idea behind KSM exploitation is rather simple: if an attacker knows the exact content of a memory page she wants to corrupt, she allocates a page with the same content and waits for the deduplication system to merge her

and the victim's page. This way, she is able to point her virtual addresses to the physical page of the victim with little effort. Obviously, a write operation will trigger the CoW-mechanism, such that changes will not propagate to a victim's VM. However, Rowhammer is a physical fault. Thus, the CoW mechanism is not triggered and the bit flip affects each application having a page with the same content.

3.3.2. Transparent Huge Pages. *Transparent Huge Pages (THP)* are a technology to automate the creation and management of huge pages. Typically, pages have a size of 4096 bytes and allocating a large amount of memory will take multiple pages which must all be tracked by the system and hardware. With huge pages – which are typically 2 MiB in size – the mapping overhead can be reduced. The algorithm does this in the background, identifying small pages and merging them into huge pages when possible.

Relationship to Rowhammer. Most importantly, THP allows for efficient hammering. A single DRAM row (as defined in the DDR3 spec.) is 8KiB in size. Given that there are, for example, 16 banks, 128 KiB of data fall into the same row index (but different bank). With huge pages in place, a single page spans across 512 "small pages" (2 MiB / 4 KiB) and hence covers 32 physical rows independently from the DRAM bank, allowing double-sided Rowhammering to become possible. Using huge pages for double-sided Rowhammering was first described by Gruss et al. [32].

Relationship to KSM. Currently, KSM works with 4 KiB pages only but will split a huge page into small pages for deduplication. It is noteworthy that the physical alignment during the split may not be destroyed and thus first allocating a large amount of memory, waiting for THP to merge the pages to huge pages, and then utilizing KSM is possible [33].

3.4. Templating

An attacker can search and "collect" pages vulnerable to bit flips, even with a particular offset. This was coined "templating" [33]. The idea is to search for bit flips in attacker-controlled memory, adjust the content of a vulnerable page, and wait for deduplication. This way, precise bit flips can be induced. For more details refer to [33].

4. Fault-attacking Deterministic Signatures

In the following, we describe how fault attacks on deterministic signature schemes work in general. Furthermore, we show concrete attacks against Deterministic ECDSA and EdDSA to confirm that the generic attacks work against real algorithms. We also discuss which steps and variables in signature algorithms can be compromised through fault attacks. Finally, we discuss possible countermeasures and show that the type of fault injection heavily influences the choice of effective countermeasures.

4.1. Nonce reuse in deterministic ECDSA

Fault attacks on deterministic signatures require only one faulty signature $(R,\ s^\not{})$ over M and one correct signature $(R,\ s)$ over M to recover the secret key a. Note that the attack described below only affects deterministic signature schemes and does not apply to randomized signatures such as ECDSA under correct usage randomness sources.

We assume a scenario, in which a signing process runs i times and in which the attacker can read the respective signature (R_i, s_i). The computation of $R_i := f([(K(a,M) + \lambda_i q]G) \bmod q$ with K being a pseudorandom function (PRF) is performed multiple times, using blinding values λ_i. Blinding is a typical countermeasure against side-channel attacks like power analysis and the result of this computation is independent from λ. We assume a blinded implementation in order to show that blinding does not prevent fault attacks against deterministic signatures.

First the attacker gets a correct value

$$s_0 = (aR_0 + H(M))K(a,M)^{-1} \bmod q.$$

Introducing an arbitrary fault in the computation of R leads to a faulty $R^\not{}$, from which $s^\not{}$ is computed using the *identical nonce* $K(a,M)$. From the signatures (R_0, s_0) and $(R^\not{}, s^\not{})$ we construct a system of two linear equations over $\mathbb{F}_q$:

$$\begin{aligned} K(a,M)s_0 &= aR_0 + H(M) \\ K(a,M)s^\not{} &= aR^\not{} + H(M) \end{aligned}$$

This system of two equations for two unknowns (output of PRF K and secret key a) can easily be solved. The point here is that the nonces are identical. For deterministic ECDSA, at least two types of faults are possible:

- The attacker modifies M after $K(a,M)$ has been computed; i.e. $s^\not{}$ is computed using the correct nonce $K(a,M)$, the correct value R_0, but an incorrect hash. In this case, an internal validation of the signature would be successful.
- The computation of $R^\not{} := f([K(a,M)+\lambda_1 q]G) \bmod q$ is attacked, i.e. $R^\not{}$ is incorrect.

A similar attack for the classic (i.e. non-deterministic) ECDSA is not possible, because the above equations would end up with three unknowns, which has no unique solution. Thus, the fault attack is only possible for deterministic nonces but not for random nonces as in classic ECDSA.

4.2. Nonce reuse in EdDSA

In contrast to classic ECDSA, EdDSA uses deterministic nonces by design. EdDSA calculates its nonces by hashing a long-term secret concatenated with M (see Equation 7), so that different messages will lead to different, hard-to-predict values of r [17].

In order to demonstrate the effects of a repeating nonce, we will assume for a moment that an attacker can produce two messages $M_1 \neq M_2$ that yield identical hash values

in the computation with $h_b, ..., h_{2b-1}$, and thus result in an identical nonce r:

$$H(h_b, ..., h_{2b-1}, M_1) = r = H(h_b, ..., h_{2b-1}, M_2). \quad (11)$$

To produce the EdDSA signature for M_1, M_2, we continue with

$$s = (r + H(R, A, M_1)\, a) \mod q \quad (12)$$

$$s' = (r + H(R, A, M_2)\, a) \mod q \quad (13)$$

with $s \neq s'$. It follows that

$$H(R, A, M_1)\, a - s = -r \mod q \quad (14)$$

$$H(R, A, M_2)\, a - s' = -r \mod q \quad (15)$$

and hence

$$\overbrace{H(R, A, M_1)}^{=:\widehat{H_1}}\, a - s = \overbrace{H(R, A, M_2)}^{=:\widehat{H_2}}\, a - s' \quad (16)$$

$$\Leftrightarrow \quad (\widehat{H_1} - \widehat{H_2})\, a = s - s' \quad (17)$$

$$\Leftrightarrow \quad a = \frac{s - s'}{\widehat{H_1} - \widehat{H_2}} \quad (18)$$

which yields the private key a.[3] Note that the values s, s', $\widehat{H_1}$, and $\widehat{H_2}$ are all known by an attacker. Contrary to [5], $h_b, \ldots, h_{2b-1}$ is *not* needed to create forged signatures for different messages.

Of course, crafting two messages $M_1 \neq M_2$ such that Equation 11 holds is infeasible for secure cryptographic hash functions such as SHA-512. Hence, the only realistic possibility to generate the same nonce twice is using the same two messages $M_1 = M_2$. This, however, yields two identical signature pairs – which reveals no further information.

We now observe that M is read twice during the signature process. First, to generate the nonce r and second, M is read again to calculate s. If an attacker is able to change M to $M^{\lightning}$ just after the generation of r, then r was calculated from M but is used to sign $M^{\lightning} \neq M$. Now assume that the attacker can perform the signing process twice: once with an unchanged M and once with the changed $M^{\lightning}$. If both resulting signatures use the same r, then the attacker has just forced the target to reuse a nonce for two different messages.

This attack gets practical when taking fault-injections into account, because M can be transformed into $M^{\lightning}$ by inducing bit flips in M. In other words, instead of crafting two messages whose hashes collide, we sign the same message M twice and induce a bit flip during one signing just after r was calculated.

When looking closely at Equation 12, we see that not only is M used to produce the hash to be signed, but also R and A. If an attacker can change either R, A, M, the above attack is possible, see Figure 2. However, note that depending on the scenario, M can be much larger than R or A and thus may be easiest for an attacker to inject faults into M.

3. As per definition, the b-bit string k is considered the private key. But a is the secret scalar for the public key $A = [a]G$ and knowing a allows for signature forgery as knowing k would do. Thus, the term secret key is used for both k and a.

5. Attack Strategies on EdDSA via Rowhammer

Traditionally, fault injection attacks required physical access to the targeted machine. With the publication of the Rowhammer attack, it became feasible to perform fault injections remotely, which is highly relevant for the attacks on deterministic signature schemes. This section describes prerequisites on both the protocol and the machine under attack, and introduces strategies for Rowhammering under different constraints. We concentrate on the analysis of EdDSA since this algorithm is considered for further standardization by NIST and FIPS 186, and it is currently being standardized in several well-used cryptographic protocols [6], [7], [8], [9], [10]. As discussed in the previous section, we see three obvious ways to provoke a nonce reuse in EdDSA: (1) faulting the scalar multiplication, (2) flipping bits in the public key A, and (3) flipping bits in the message M.

5.1. Attacker Scenario and Prerequisites

We consider a cloud scenario and an attacker whose virtual machine is co-located to a victim's virtual machine. It was shown in the past that this is feasible [35]. The victim is running a cryptographic application using EdDSA signatures. The attacker can execute Rowhammer attacks as described in section 3.4. The goal of our attacker is to recover the victim's private EdDSA key.

5.1.1. Cryptographic Protocol Prerequisites. The cryptographic scheme must meet the following prerequisites:

- Signatures can be observed by an attacker, i.e. the victim serves as a signature oracle.
- At least two EdDSA signature generations (Pure- or HashedEdDSA) can be triggered.
- It must be feasible for an attacker to trigger independent signature generations yielding the same result. In other words: No *uncontrollable randomness* is incorporated into the signature generation.
- The to-be-signed message M is known to an attacker, but it is not necessary to choose its content.

5.1.2. Prerequisites for Rowhammer. The prerequisites for executing a Rowhammer attack are as follows:

- Rowhammering must be feasible on the machine under attack.
- For the weak attacker scenario, the system needs to be vulnerable to single-sided Rowhammering and M needs to be relatively large compared to the overall available main memory.
- For the stronger cross-VM attack scenario, it is necessary for the host system to support KSM and THP.

Note that THP is a default-on feature in many Linux distributions [33] and thus most setups differ in the deployment of memory deduplication only, i.e. if a form of memory deduplication is active or not.

	Faulty R	**Faulty** A	**Faulty** M
Step 1	$r = H(h_b, \ldots, h_{2b-1}, M)$	$r = H(h_b, \ldots, h_{2b-1}, M)$	$r = H(h_b, \ldots, h_{2b-1}, M)$
Step 2	$R = [r]G$	$R = [r]G$	$R = [r]G$
Step 3	$s_R^{\lightning} = (r + H(R^{\lightning}, A, M)\, a) \mod q$	$s_A^{\lightning} = (r + H(R, A^{\lightning}, M)\, a) \mod q$	$s_M^{\lightning} = (r + H(R, A, M^{\lightning})\, a) \mod q$

Figure 2. Three fault injections provoking nonce reuse in EdDSA. For each fault injection three computational steps revealing the secret key are shown.

5.2. Attack Strategies

The success probability of the different Rowhammer attack strategies depends on the setup under attack and the attacker's choice of the faulted variable. For example, random single-sided hammering is unlikely to flip bits in small public keys A, but has a realistic success chance to flip bits in large messages M. Additionally, with the Flip-Feng-Shui [33] method using KSM and THP, it may even become feasible to flip bits in assembly instructions or the base point G to inject faults in the scalar multiplication – a scenario typically found in side-channel analysis of smartcards and tamper-proof hardware.

Example for weak attacker scenario. Consider the following configuration of the weak attacker scenario deploying single-sided hammering. The host is equipped with 4 GiB of main memory, the attacker-VM allocates 2 GiB, M is 1 GiB in size. Assuming a uniform distribution of M over the main memory, the probability that two randomly picked addresses are in the same bank depends on the DRAM configuration. We assume 16 banks in total, thus yielding a probability of $\frac{1}{16}$ for picking two rows in the same bank. Each of those rows will typically have 2 adjacent rows (corner cases excluded). The probability that at least one of those 4 adjacent rows in total contains a slice of the message is therefore $1 - (1 - \frac{1}{4})^4$. This yields roughly a probability of $\frac{1}{24}$ for randomly choosing an address pair which can *potentially* inject bit flips in a page of M. The probability is expected to be much lower in practice, because not each row is vulnerable to hammering. However, many hammering attempts are possible and the success rate highly depends on the quality of the deployed DRAM.

Constraints. Single-sided hammering has various constraints on its own:

- Implementations may limit the message size and provide a fallback from PureEdDSA to HashedEdDSA for bigger messages.
- The corrupted message $M^{\lightning}$ is required to reconstruct the private key and thus the exact bit flips in M must be known. Otherwise, a large message with multiple bit flips will become a combinatorial problem (i.e., with $\binom{\text{amount of bits}}{\text{amount of bit flips}}$ possible combinations). In fact, with large messages, more than one bit flip would become infeasible to test for. However, an attacker can observe the first change and test under the assumption that exactly one bit has flipped – which is often the case when conducting less aggressive hammering. Furthermore, optimizations can be applied, for example, when bit flips are known to flip from 1 to 0 only.
- Many machines are assumed to exhibit bit flips only via double-sided hammering in reasonable time [30]. However, we experienced bit flips in longer test periods and concluded that single-sided hammering is feasible on our testing setup. Typically, we were able to identify randomly induced bit flips within a few hours of hammering. Similar results were also reported on the Rowhammer mailinglist.[4]

5.2.1. Faulting R. This fault injection was previously discussed by Barenghi and Pelosi [5]. The authors state that "since the value of k [r using our notation – ed. note] depends on both msg and an unknown portion of the hash [$h_b, \ldots, h_{2b-1}$ using our notation – ed. note] of d [k using our notation – ed. note], the attacker will not be able to exploit it to successfully forge a signature for any message different from msg" [5]. Their conclusion was motivated by the fact that a forged signature *can be proven* not to be generated by the legitimate owner since the determinism of r will deduce the correct values for $h_b, \ldots, h_{2b-1}$.[5] However, we find that a signature check *will* pass. This is because an attacker in possession of the private key a can simply choose another $h'_b, \ldots, h'_{2b-1}$ instead of the original one and generate a valid signature for any M.

We see two ways to induce faults in R via bit flips: by corrupting the calculating code (i.e., by inducing bit flips into the corresponding assembler instructions) or by inducing faults into the base point G. The preferred way depends on the outcome of the templating phase (see section 3.4). The base point is expected to have at least 32 bytes and hence there are 32 exploitable offsets per page. The number of exploitable bit flips in the assembler code depends on the particular implementation.

5.2.2. Faulting A. Razavi et al. implemented a successful attack against *OpenSSH* by inducing faults in RSA public keys [33]. Their attack benefits from Linux' *page cache*, a consolidated cache used to accelerate file reads from disk. When a file is read for the first time, it is stored in the page cache, so that subsequent reads are served from main memory instead of disk. The whole main memory of a virtual machine is candidate for deduplication. Therefore, an attacker can (1) trigger a signature generation to put the public key file into page cache, (2) wait for KSM to deduplicate the page, (3) induce a bit flip in A and (4) trigger a second signature generation to obtain the required faulty $s^{\lightning}$. If a protocol meets the requirements discussed earlier,

4. https://groups.google.com/forum/#!forum/rowhammer-discuss
5. We thank the authors of [5] for clarification.

those four steps are sufficient to obtain the EdDSA private key.

Secret keys are likely protected in a way such that bit flips will cause key loading routines to fail. Public keys, on the other hand, may experience less protection as OpenSSH demonstrates: OpenSSH documentation recommends that the `authorized_keys` file is not accessible by others, but no integrity checks are used.

5.2.3. Faulting M. Compared to R and A, the message M can be a particularly good attack target due to the fact that M can be very large. When large messages are signed, the probability to inject faults into the message increases. Additionally – because whole pages can be filled with known data – deduplication becomes easier to trigger.

Faulting the message is especially useful for applications which read a message twice from the disk, as is the case for PureEdDSA implementations. Due to the page cache, the same properties as in the public key scenario may apply. Additionally, since the message may fill whole pages, each template becomes exploitable.

However, a fault must be induced in a specific time frame, specifically after step 1 and before step 3 in Figure 2. For small pages, the scalar multiplication is the most time-consuming operation. For large messages, the computation is dominated by the hash operations in the first and in the last step of signing. The time window, in which bit flips in M are exploitable, can thus be estimated as the time between two reads at the same offset in the message. To conclude, an approximate time window for hammering is at least 50% of the hashing time, plus the time needed for scalar multiplication.

It is noteworthy that bit flips may become persistent as long as a file is kept in page cache. Thus, if the time frame was missed by an attacker, subsequent messages will differ from the original by exactly the bit flips induced in the prior attempt. This leads to different messages but is not problematic, because each bit flip is registered and can easily be taken into account for future attempts. The exact behavior is highly dependent on the application-under-attack (i.e., if files are `mmap`ed or `read`, etc).

6. Applicability of the Attack to Other Signature Schemes

In this section, we introduce the background and explain the general applicability of our results regarding signatures in a form similar to EdDSA. Therefore, we first give the structural connection between interactive zero-knowledge proofs and signatures of this form and thereby the origin of such signatures. More precisely, we concentrate on Σ-protocols as a special class of interactive zero-knowledge proofs since they automatically provide an algorithm for the extraction of the secret key if nonces collide. Subsequently, we explain the impact of the issue caused by fault attacks relating to the set of signatures originated from Σ-protocols.

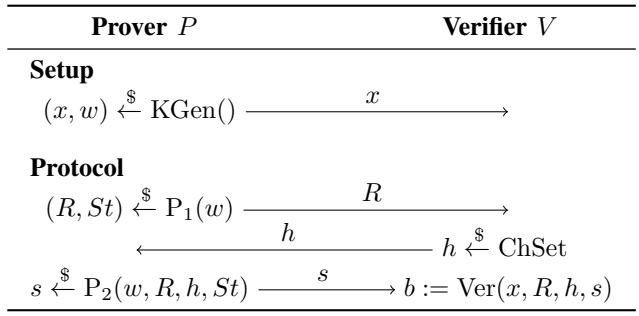

Figure 3. Generic Σ-protocol.

6.1. From Σ-Protocols to Signatures

Σ-protocols are a special form of interactive zero-knowledge proofs [1] between a prover P and a verifier V in which P proofs the knowledge of a witness w for a public element x such that $(x, w) \in \mathcal{R}$ holds for the binary relation $\mathcal{R}$ between a public element and its private counterpart. P proves the knowledge towards V without revealing any non-public knowledge to V. Consequently, V is not able to perform a proof of this knowledge after the execution of the protocol to another party. Σ-protocols are defined as $\Sigma = (\mathrm{KGen}, \mathrm{P}_1, \mathrm{P}_2, \mathrm{ChSet}, \mathrm{Ver})$ such that KGen generates a pair $(x, w) \in \mathcal{R}$ where the possession of the private parameter w is proven for the public parameter x. P_1 generates parameters for a session for which the proof is computed with P_2. The internal state St of the prover is handed over from P_1 to P_2. The verifier's challenge, which is also input to the algorithm P_2, is randomly chosen from a challenge set ChSet. Finally, the Ver algorithm of V outputs 1 if the proof is correct or 0 otherwise. Typically, the proof setting is parameterized by (x, w) but in order to show the connection between zero-knowledge proofs and the respective signatures, we add the setup phase to the description.

In the protocol description in Figure 3, P proves the possession of the witness w for the public element x in the session specified by R, where h is V's challenge. A zero-knowledge protocol of this form must satisfy three main conditions:

- *Completeness* if P possesses a witness w for x, then V is satisfied by the proof,
- *Soundness* if P does not know a witness w for x, then P is able to prove its possession to V only with negligible probability and
- *Zero-Knowledge* by executing the protocol, V learns nothing but the fact that P possesses a witness w for x.

Soundness is formally shown by fulfilling *Special Soundness*: for two correct transcripts (R, h, s), $(R, h', s'), h \neq h'$, there exists an algorithm $\mathrm{Ext}(x, R, h, h', s, s') \rightarrow w$. Essentially, this property implies that if a prover can correctly answer two different challenges of V for one session started with R, then P already knew w for x such that $(x, w) \in \mathcal{R}$. Although this

property is only mandatory for Σ-protocols, according to Cramer et al. this restriction is non-serious because "all known proofs of knowledge have this property" [36].

In this sense one can consider the three messages as: R is a unique session identifier, h is the unforeseeable challenge by the verifier, and s is the proof of knowledge in the session R for the challenge h. Now h is the protection for V that P cannot cheat and in the light of special soundness, R is the protection for P that nobody can retrieve her secret.

Fiat and Shamir proposed a generic transformation [11] to construct signature schemes (see Figure 4) from identification schemes, and thereby from zero-knowledge proofs and Σ-protocols. In order to convert the interactive protocol into a non-interactive proof, the unpredictable challenge from V is replaced by the output of a collision resistant hash function H. The resulting signature scheme $Sig = (\mathrm{Gen}, \mathrm{Sign}, \mathrm{Vfy})$ can be proved to be secure according to the definition of existential unforgeability under chosen message attacks (EUF-CMA) in the random oracle model (ROM).

The binary relation from the zero-knowledge proof is the relation between secret keys and public keys in the signature scheme.

In EdDSA, the computation of h also includes the signer's public key $h = \mathrm{H}(R, A, M)$. In contrast to the Fiat Shamir transformation where R is computed probabilistically, in EdDSA the algorithm P_1 is deterministically computed on the message and a secret value.

The proof of the transformation, however, assumes no collisions of R. Particularly Fiat and Shamir require that "P uses each R_i only once" during multiple executions of the proof.[6]

Since EdDSA not only computes the challenge from the verifier with a deterministic function but also the algorithm P_1, and the procedure P_2 is deterministic, the whole signing algorithm is deterministic. Therefore, collisions of R are an intrinsic side effect of the EdDSA construction. This effect is contrary to the original intent of R, which requires R to be randomly chosen and free of collisions.

6.2. General Applicability on Deterministic Signatures from Σ-Protocols

The issue presented in this paper applies to all signature schemes that are based on a Fiat Shamir transformed Σ-protocol (e.g. [11], [17], [37], [38], [39], [40]) and more generally zero-knowledge proof that fulfill *special soundness* (e.g. [41], [42], [43]). In fact, many post-quantum secure signature schemes employ Σ-protocols (e.g., [38]). However, as long as the computation of R is probabilistic, the extraction algorithm Ext from the special soundness condition cannot be used to derive the secret key. As soon as R is computed deterministically and h can be manipulated such that two tuples $(h, s), (h', s'), h \neq h'$ are gathered for one R, the secret key can be computed.

6. The R in Fiat and Shamir's construction consists of multiple values R_i and the proof originally is conducted between parties A and B.

Gen()	$(pk, sk) \xleftarrow{\$} \mathrm{KGen}()$
Sign(sk, M)	$(R, St) \xleftarrow{\$} \mathrm{P}_1(sk)$ $h := \mathrm{H}(R, M)$ $s \xleftarrow{\$} \mathrm{P}_2(sk, R, h, St)$ $\sigma := (R, s)$
Vfy(pk, M, σ)	$h := \mathrm{H}(R, M)$ $b := \mathrm{Ver}(pk, R, h, s)$

Figure 4. Signature scheme derived from the Fiat Schamir transform.

Since h is not computable from the signature (R, s), the fault attack has to be performed carefully (see section 5.2). If a similar determinism would be applied to a scheme like the Schnorr signature scheme [37] where h is part of the signature (h, s), the attack could be much easier. Instead of attacking only a few bits of the processed message, the input of the second call of H can arbitrarily be manipulated (i.e. any number of bits) in order to compute the secret key.

As a consequence, either the determinism of a scheme is effectively preserved or randomness is inevitable to derive a secure signature scheme from a zero-knowledge proof with *special soundness*.

7. Application of the Attack to Real-world Protocols

We are not aware of any current cryptographic standards adapting the deterministic ECDSA scheme. On the other hand, there exist several major cryptographic standards and implementations adapting EdDSA. EdDSA is being implemented in TLS [6], SSH [7], IPsec [8], X.509 infrastructures [9], and DNSSEC [10]. In the following, we analyze the major cryptographic standards using EdDSA with respect to the prerequisites described in section 5.1.

7.1. On the Impossibility of Attacking TLS, SSH, and IPSec

7.1.1. TLS. Version 1.3 of the Transport Layer Security (TLS) protocol [6] specifies the usage of PureEdDSA for securing the authenticity of server-generated (EC)DH parameters exchanged in the TLS handshake. A typical handshake with server authentication in TLS 1.3 works as follows. The client starts the handshake with the `ClientHello` messages. The `ClientHello` contains a list of cipher suites, client random and further cryptographic properties of the TLS connection. In addition, it contains a fresh (EC)DH key denoted as `ClientKeyShare`. The server responds with a list of TLS messages. `ServerHello` contains a fresh server random, `ServerKeyShare`, and further cryptographic properties. `Certificate` contains an X.509 certificate possibly with an EdDSA public key. The server signature is constructed over all previous messages using the PureEdDSA algorithm. Finally, both peers exchange `Finished` messages to confirm that they are

in possession of the secret key established based on the exchanged (EC)DH key shares.

Although TLS 1.3 uses PureEdDSA, from the perspective of our attacker, a practical attack on EdDSA is infeasible due to the EdDSA signature always being generated over fresh server-generated inputs: the server nonce and the server key share. Therefore, the attacker is not able to force the server to sign the same message multiple times. Previous TLS versions do not officially support EdDSA.

7.1.2. SSH. Recent implementations of the Secure Shell (SSH) transport layer protocol [7] support Ed25519 (e.g., OpenSSH since version 6.7). However, the SSH protocol exposes similar behavior as TLS 1.3 and makes the attack infeasible. SSH specifies two key exchange mechanisms: Diffie-Hellmann and RSA key exchange. In both cases, the signature input is derived by incorporating the server's random cookie, which is sent at the beginning of the protocol flow in the `SSH_MSG_KEXINIT` message.

7.1.3. IPsec. Internet Key Exchange Protocol Version 2 (IKEv2) [44], [45] standardizes the key exchange mechanisms for IPsec [46]. A recent Internet draft specifies the usage of EdDSA for IKEv2 [8]. In IKEv2 the server computes a signature over its initial message and the initiator nonce (Ni). The server initial message contains Header data (HDR), Security Association (SA), and two random values: the server nonce (Nr) and the fresh DH key share (KEr). Therefore, similarly to TLS and SSH, our Rowhammer attacks are not applicable.

7.2. Potential Dangers in OCSP

Online Certificate Status Protocol (OCSP) allows a client to query a certificate authority about the current status of a certificate [47]. For this purpose, the client sends a certificate identifier in its request. The server responds with a message containing the current certificate status (good, revoked, or unknown), the time when the response was generated, and a signature computed across a hash of the response. Both OCSP messages can contain extensions with client and server nonces.

OCSP fulfills the protocol prerequisites described in section 5.1.1 and establishes a valid scenario for our Rowhammer attack; an attacker may be able to force the OCSP responder to generate signatures over equal messages within a short time period without uncontrollable randomness. This is due to the existence of a client nonce extension which is reflected in the OCSP response. Since any nonce can be sent to the server, it is unlikely that equal nonces are cached. Furthermore, nonces are defined as `OCTET STRINGS` [47] and not restricted in size. Although we are not aware of EdDSA actively being used in OCSP, given the fast deployment of this signature algorithm in other standards, we can assume its future deployment as well. In that case, we recommend to use server nonce extensions in OCSP responses and refer to section 9.

8. Attacking Minisign

In this section, we analyze the feasibility of our attack in a realistic setting using *Minisign*[7] – a tool similar to *OpenBSD's signify* – and demonstrate two variants how EdDSA can be faulted in a cross-VM scenario. Minisign utilizes *libsodium*[8] (a fork of *NaCl*) for its cryptographic primitives. NaCl itself utilizes the "ref10" reference C-implementation of Ed25519. Minisign supports both variants of EdDSA, Pure and HashedEdDSA. HashedEdDSA is not used by default, but mandatory for messages above 1 GiB in size. However, if a message smaller than 1 GiB is chosen, Minisign defaults to PureEdDSA.

8.1. Hardware and Software Setup

Our test device (Arch Linux 4.12.6 x86_64) was equipped with 8 GiB of DRAM known to be vulnerable against Rowhammer and deployed two virtual machines (Ubuntu 16.04 LTS) via KVM/QEMU called the attacker- and the victim-VM. On this machine, we typically observed bit flips induced by single-sided hammering (using the probabilistic version of `rowhammer-test`[9]) within some hours of testing. With double-sided hammering, we typically observed bit flips within a few seconds of testing.

Attacker-VM. The attacker machine ran a hammering program which could be configured for random (single-sided) and deduplication-based (double-sided) hammering. In order to induce precise bit flips in memory pages, we reproduced the findings of [33] and implemented the templating phase to find vulnerable pages.

Victim-VM. The victim machine repeatedly signs a file and was able to listen for incoming signature requests. Each signature request was passed to Minisign and the resulting signature was sent to the attacker-VM. That is, so the victim-VM becomes a signature oracle.

8.2. Proof of Concept with Memory Deduplication

When memory deduplication is active, a similar attack to [33] can be executed. As stated earlier, a single bit flip in the public key can result in the private key becoming compromised; however, in Minisign, the public key is protected by a checksum.[10] Due to this checksum, we opted to fault the message, despite it not being an optimal target, as Minisign does not `mmap` the message, but rather creates an internal copy via `malloc`/`fread`. This means that KSM must deduplicate the internally created copy. Furthermore, since the message is likely to stay in the page cache, bit flips induced over time will accumulate. Being aware of this pitfall, we designed the hammering program to collect each observed bit flip for later analysis and eventually tested for each permutation. Our hammering program allocated

7. https://github.com/jedisct1/minisign
8. https://download.libsodium.org/doc/
9. https://github.com/google/rowhammer-test
10. See Secret key format at https://jedisct1.github.io/minisign/

memory pages (e.g., based on file input) and created a backup for each page. The backup pages were "blinded" by XORing with a constant value (or using a random 8 byte page prefix) so that KSM did not merge the to-be-hammered pages with the backup pages.

The attack was conducted as follows:

1) Directly after bootup allocate a large message in the attacker-VM that is co-located to the victim-VM.
2) Wait for THP in the victim-VM and for THP in the host-OS to merge 4 KiB pages into huge pages. The intuition here is that both merge operations will roughly result in physically continuous huge pages in the DRAM [33].
3) Allocate the same message a second time in the attacker-VM and wait for KSM to deduplicate both memory pages, in oder for them to be added to KSM's stable tree.
4) Trigger a signature generation with the message in the victim-VM. The obtained signature will be used as a reference.
5) Trigger further signature generations while hammering the deduplicated addresses.
6) If a signature deviates from the reference signature, a bit flip was triggered in the page cache or during signature generation. If it does not, go to step 5.
7) Try to extract the private key as discussed in section 4.2. If this is not successful, for example, if R differs from the reference signature, go to step 5.

8.3. Proof of Concept Without Memory Deduplication

Executing precise Rowhammer attacks in cloud-scenarios is difficult due to indirections in the memory system. Typically, an attacker with control over the virtual machine is able to access `/proc/<pid>/pagemap` which allows for virtual-to-physical address translation. However, the physical addresses obtained in a virtual machine are so called "guest-physical" addresses and its relationship to real physical addresses (i.e. "host-physical" addresses) is typically unknown to an attacker.

Nonetheless, one can opt for random (single-sided) hammering. By using random hammering, no information on virtual-to-physical mapping of addresses (how the operating system maps virtual pages to physical frames) or physical-to-DRAM mapping (how physical addresses are mapped to rank, bank, row, etc.) is needed. Furthermore, no "memory massaging" primitives as used by Razavi et al. [33] are needed.

We estimate the feasibility of an attack based on random hammering with two variables: the interleave of attacker- and victim-pages and the amount of vulnerable cells in main memory. We examined the memory of our test machine (via a patched version of KVM to translate guest-physical addresses to host-physical addresses) and evaluated how many attacker-VM rows are adjacent to victim-VM rows. We call these rows "neighboring rows". We measured the interleave of two 1 GiB messages allocated in the attacker- and victim-VM over 20 reboots of the host machine.

We measured up to 18.83% max neighboring rows on reboot and up to 18.55% max after idling for five minutes.

Analysis. In order to extract the private key, we developed an analysis program which takes a list of signatures and encountered bit flips (if available) as input. First, a hash map with R as a key and a list of s' is created. The reason is that we reconstruct the secret key only if two signatures with $R_1 = R_2$ and $s_1 \neq s_2$ were found. Afterwards, each bit flip is tested against candidate signature pairs. The test succeeds if the private key is found. This can be verified by calculating $A' = [a']G$ and checking if $A = A'$.

8.4. Analysis of Single-sided Hammering

We explored our test scenario and found that we can successfully inject cross-VM bit flips only by random hammering. During our tests, we were able to conduct the full attack in under four hours of random hammering and a 700 MiB large message. It is important to note, that these tests are highly specific to the hardware of the machine-under-attack and that the attack outcome may vastly differ among hardware.[11] To our knowledge, there are no public studies investigating the vulnerability of off-the-shelf hardware to single-sided hammering and we did no further evaluation of the outcome of the attack (i.e., the likelihood of a key becoming compromised within a specific time frame).

Nonetheless, our test hardware was vulnerable to random hammering, which shows that weaker attackers with no access to double-sided hammering primitives may be able to perform the presented attack. Furthermore, we expect that optimizations as discussed by the authors of [30], [32], [48] may be applied to make single-sided hammering more efficient.

9. Countermeasures

In the following we present several countermeasures and discuss why some of these countermeasures do not work.

9.1. Signature Validation

Our Rowhammer attack on EdDSA leads to the computation of invalid signatures. A typical countermeasure against such fault attacks is signature verification. This countermeasure can be successfully applied in the RSA signature generation process with the Chinese Remainder Theorem (CRT) as showed by Lenstra [49], [50]. Lenstra's attack exploits a random fault injected in a CRT multiplication step during signature generation. Since the signature verification does not include CRT multiplication, a simple signature verification step can detect a fault attack.

11. See the rowhammer-discuss mailinglist (https://groups.google.com/forum/#!forum/rowhammer-discuss) for reportedly failing and surviving machines.

However, this countermeasure does not help to prevent our attack on EdDSA. Imagine the attacker managed to induce a bit flip in the message M, which is located in the memory and used as an input to the signing function (see also Figure 2):

$$s^{\lightning} = (r + H(R, A, M^{\lightning})a) \mod l$$

In that case, the signature verification process would lead to a correct result, because the modified message $M^{\lightning}$ is used for the verification:

$$[s^{\lightning}]G = R + [H(R, A, M^{\lightning})]A$$

This interesting property of fault attacks on EdDSA has also been independently discussed by Tony Arcieri.[12]

9.2. Signature Generation

A potential countermeasure against our Rowhammer attack on EdDSA would be to generate the signature twice and compare the outputs of both signature generation functions. However, a more precise Rowhammer attacker might be able to introduce a bit flip on the same position in two or even more messages. Given the recent development in the Rowhammer attacks, this threat should not be overlooked.

Another important point that needs to be considered is the performance penalty that is introduced by generating signatures. Even though this penalty is not as high as for RSA signatures. Especially for large messages over 1 MiB, the signature generation process is slower than signature verification. See also Table 1.

9.3. Usage of HashedEdDSA

EdDSA can be used in two variations: PureEdDSA and HashedEdDSA. PureEdDSA uses message M directly as an input. HashedEdDSA hashes M with a hash function H before executing further operations. From a Rowhammer attacker perspective, the PureEdDSA algorithm offers more flexibility by executing the attack since she needs to flip a single bit in a message of arbitrary length. On the contrary, HashedEdDSA forces the attacker to induce a bit flip in the output of a few bytes (depending on the hash algorithm and the internal representation of the hash result). Therefore, HashedEdDSA would be a better choice to counter the Rowhammer attacks.

PureEdDSA is recommended because of possible hash collisions [19]. Although the attack on HashedEdDSA is more complicated than on PureEdDSA, we need to consider a more skilled Rowhammer attacker who can also induce precise bit flips in the hashed message input.

9.4. Checksum Over Input Values

Another countermeasure for EdDSA would be to introduce checksum computation over input values R, A and M, before and after signature generation. If the checksums are equal, no bit has been flipped. Otherwise, a fault attack can be assumed and the signature generation process must be interrupted.

Minisign implements this countermeasure on public keys. The checksums over public keys are verified during the signature generation. This made our Rowhammer attack with public keys infeasible.

12. https://github.com/jedisct1/libsodium/issues/170

TABLE 1. DURATION OF SIGNING AND VERIFICATION OPERATIONS DEPENDING ON THE MESSAGE SIZE PERFORMED WITH THE LIBSODIUM LIBRARY (METHODS `CRYPTO_SIGN_ED25519_VERIFY_DETACHED` AND `CRYPTO_SIGN_ED25519_DETACHED`). THE MEASUREMENT RESULTS ARE PROVIDED IN CLOCK CYCLES OBTAINED BY `RDTSC`.

Message (bytes)	Signing (cycles)	Verification (cycles)
10	232 155	550 380
100	207 795	461 272
1 000	230 442	415 431
10 000	682 785	546 288
100 000	3 022 662	1 718 436
1 000 000	28 205 265	14 153 310

9.5. Additional Randomness

The main argument for deterministic signatures is the need for strong randomness (e.g., following [51], [52]). One could instead combine a somewhat weaker random number generator with the deterministic procedure to generate "ephemeral" keys. This construction would have the same security properties as EdDSA, but the signatures over equal messages would always be different.

XEdDSA [53] implements this kind of countermeasure and appends 64 bytes of secure random data Z to the message M while computing $r = H(h_b, \ldots, h_{2b-1}, M, Z)$. This makes our fault attacks infeasible.

10. Related Work

10.1. Attacks on Cryptographic Algorithms

RNG failures. Invalid functionality of RNGs lead to several remarkable attacks on cryptographic implementations. In 2010, the hacker group fail0overflow retrieved Sony's ECDSA key that could allow them to sign any application for the Play Station 3 [2]. The reason was a reused ECDSA nonce. A vulnerability in a Java RNG lead to a further failure by the usage of ECDSA. Reuse of nonces allowed attackers to retrieve private keys from Bitcoin Android apps [3].

Vulnerabilities in RNGs do not only influence the security of ECDSA cryptosystems. For example, if the same nonce is used during the AES-GCM encryption process, the attacker can learn the authentication key and create arbitrary valid ciphertexts [54]. This attack is also called the *Forbidden attack*. Therefore, several cryptographers recently described AES-GCM as "fragile" [55], [56]. In 2016 Böck et al. showed that about 70 000 servers are potentially vulnerable to this attack [57].

Attacks on elliptic curves. In 2000 Biehl et al. presented an invalid curve attack that allows to extract EC private keys from applications that do not check whether the incoming ECDH share lies on a correct curve [58]. Practical application of this attack to TLS implementations was shown by Brumley et al. [59] and Jager et al. [16].

ECDSA implementations can enable further side-channels. For example, Brumley et al. showed that ECDSA key extraction from TLS servers is possible remotely by measuring timing differences [60], [61]. Genkin et al. showed that the ECDSA key extraction is possible even with magnetic side-channels [62].

Attacks on EdDSA. Very recently two publications pointed out problems in deterministic signature schemes. Romailler and Pelissier presented a practical fault attack on EdDSA [63]. They developed an attack scenario based on the original implementation by Bernstein [64] which was run on Arduino Nano. Samwel et al. presented a differential power analysis (DPA) attack on EdDSA which was able to leak private keys from about $4\,000$ traces. In particular, the attack targeted the nonce with the SHA-512 hash function [65] (see the computation of r in Equation 7). Our paper is a concurrent work and was created independently of these two results.

10.2. Rowhammer Attacks

Local attacks. Kim et al. assumed that they can "develop [...] a disturbance attack that [...] perhaps even hijacks control of the system" [21]. Their assumption was proven right by Seaborn et al. as they demonstrated the first kernel privilege escalation under GNU/Linux based on Rowhammer [30]. Govindavajhala et al. presented a similar approach, by utilizing a classic fault attack by overheating the memory with a light bulb [66]. One of the first cryptographic Rowhammer-based attacks was presented by Bhattacharya et al. [67]. The authors showed how to flip bits in an RSA secret exponent and eventually how to reconstruct it. With *Windows 8.1* and *Windows 10*, memory deduplication is used by default. Bosman et al. demonstrated a JavaScript-based exploit targeted to hijack Microsoft's Edge browser [68]. The attack involves a memory deduplication side-channel to disclose valuable data (i.e., high entropy byte-by-byte disclosure) and utilizes Rowhammer to gain arbitrary read and write capabilities in Microsoft's Edge browser.

Cross-VM attacks. Xiao et al. introduced a method to gain arbitrary access to a paravirtualized Xen host via Rowhammer [69]. Paravirtualization, in contrast to full virtualization solutions, is not transparent for the virtualized operating system and must be adapted to cooperate with the hypervisor through a specific *Hypercall-API*. They presented a *page table replacement attack* to replace page tables via Rowhammer bit flips. Razavi et al. introduced the first cross-VM Rowhammer attack [33]. They were able to *precisely* induce bit flips in deduplicated pages, break *OpenSSH's* public key authentication, and compromise Ubuntu's update mechanism.

Remote Attacks. Seaborn et al. were able to escape from Google's Native Client by utilizing the `clflush` instruction [30]. Google's Native Client validated the code prior to executing it, such that it conforms to a specific subset of x86. The authors were able to execute a bit flip in a previously validated code and escape the sandbox. After that, the first JavaScript-based Rowhammer attack was presented by Gruss et al. [32]. In order to execute bit flips the authors had to solve two key challenges: (1) find a way to bypass the CPU cache in JavaScript and (2) retrieve information on physical addresses in JavaScript. The first challenge was solved by using a novel cache eviction strategy. The second challenge was solved by the observation that operating systems tend to use huge pages when large typed arrays are used. Qiao et al. analyzed whether it is possible to trigger memory access patterns remotely from benign code [31], and performed a search of `clflush`- and non-temporal instructions. They found 7 packages containing `clflush` and 21 packages containing non-temporal instructions in the Debian source code repository, allowing them to induce bit flips via benign code located in the *Newlib* C library.

11. Conclusion

In this paper, we presented practical fault attacks on EdDSA and deterministic ECDSA signatures. Somewhat unexpectedly, we found EdDSA more susceptible to Rowhammer-based attacks than classical ECDSA. The fault attacks are not only applicable on smartcards but also on commodity hardware. With the newest developments in the area of Rowhammer, remote fault attacks are becoming a more prevalent threat that needs to be considered.

Although the presented attacks are only effective under special circumstances, they show a new fragile side of deterministic signature schemes and their general sensitivity to fault attacks. Our aim is to raise the awareness regarding recent fault attacks on these signature schemes, given their importance and fast deployment in practice.

We presented several countermeasures that could effectively mitigate our fault attacks and recommended to extend EdDSA to incorporate a long-term secret as well as *additional* per-signature entropy during nonce generation as already specified in XEdDSA. This hardens EdDSA against Rowhammer-based attacks and does not decrease its security in most scenarios, even with bad RNGs in place. This countermeasure should be implemented together with a checksum protecting EdDSA's public keys and messages if not otherwise deployed.

Acknowledgements

We thank Daniel Gruss for countless remarks which helped us to develop the Rowhammer attack. We also thank one of our USENIX Security reviewers who pointed out that OCSP might be vulnerable to our attacks.

References

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems (extended abstract)," in *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, R. Sedgewick, Ed. ACM, 1985, pp. 291–304. [Online]. Available: http://doi.acm.org/10.1145/22145.22178

[2] bushing, marcan, segher, and sven, "Console hacking 2010 – ps3 epic fail," in *27th Chaos Communication Congress*. Chaos Computer Club, 2010.

[3] P. Ducklin. (2013) Android random number flaw implicated in Bitcoin thefts. [Online]. Available: https://nakedsecurity.sophos.com/2013/08/12/android-random-number-flaw-implicated-in-bitcoin-thefts/

[4] T. Pornin, "RFC 6797 Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)," 2013.

[5] A. Barenghi and G. Pelosi, *A Note on Fault Attacks Against Deterministic Signature Schemes (Short Paper)*. Cham: Springer International Publishing, 2016, pp. 182–192. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-44524-3_11

[6] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," 2017, internet draft, version 19, https://github.com/tlswg/tls13-spec.

[7] T. Ylonen and C. Lonvick, "RFC 4253 The Secure Shell (SSH) Transport Layer Protocol," 2006.

[8] Y. Nir, "Using Edwards-curve Digital Signature Algorithm (EdDSA) in the Internet Key Exchange (IKEv2)," 2017, internet draft, https://tools.ietf.org/html/draft-ietf-ipsecme-eddsa-00.

[9] S. Josefsson and N. Mavrogiannopoulos, "Using EdDSA with Ed25519/Ed448 in the Internet X.509 Public Key Infrastructure," 2017, internet draft, version 4, https://tools.ietf.org/html/draft-josefsson-pkix-eddsa-04.

[10] O. Sury and R. Edmonds, "RFC 8080 Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC," 2017.

[11] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, ser. Lecture Notes in Computer Science, A. M. Odlyzko, Ed., vol. 263. Springer, 1986, pp. 186–194. [Online]. Available: http://dx.doi.org/10.1007/3-540-47721-7_12

[12] S. Josefsson and I. Liusvaara, "RFC 8032 Edwards-Curve Digital Signature Algorithm (EdDSA)," 2017.

[13] W. Killmann, T. Lange, M. Lochter, W. Thumser, G. Wicke, D. Feldhusen, M. Gebhardt, G. Illies, M. Kasper, R. Petri, and O. Stein, "Minimum Requirements for Evaluating Side-Channel Attack Resistance of Elliptic Curve Implementations," Bundesamt für Sicherheit in der Informationstechnik, 2013. [Online]. Available: http://www.bsi.bund.de/

[14] M. Schmid, "Ecdsa-application and implementation failures," 2015.

[15] S. Vaudenay, "The security of dsa and ecdsa," in *International Workshop on Public Key Cryptography*. Springer, 2003, pp. 309–323.

[16] T. Jager, J. Schwenk, and J. Somorovsky, "Practical invalid curve attacks on tls-ecdh," in *European Symposium on Research in Computer Security*. Springer, 2015, pp. 407–425.

[17] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 124–142.

[18] H. Edwards, "A normal form for elliptic curves," *Bulletin of the American Mathematical Society*, vol. 44, no. 3, pp. 393–422, 2007.

[19] D. J. Bernstein, S. Josefsson, T. Lange, P. Schwabe, and B.-Y. Yang, "Eddsa for more curves," 2015. [Online]. Available: http://ed25519.cr.yp.to/eddsa-20150704.pdf

[20] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[21] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 361–372, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2678373.2665726

[22] N. A. Ricci, "Rowhammering: a physical approach to gaining unauthorized access," *Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal*, vol. 3, no. 1, p. 5, 2016.

[23] J. Crawford, B. Morris, S. Mandava, and R. Ramanujan, "Techniques for probabilistic dynamic random access memory row repair," Sep. 18 2014, uS Patent App. 14/132,987. [Online]. Available: https://www.google.com/patents/US20140281206

[24] K. Bains, J. Halbert, C. Mozak, T. Schoenborn, and Z. Greenfield, "Row hammer refresh command," Jan. 2 2014, uS Patent App. 13/539,415. [Online]. Available: https://www.google.com/patents/US20140006703

[25] K. Bains and J. Halbert, "Distributed row hammer tracking," Mar. 29 2016, uS Patent 9,299,400. [Online]. Available: https://www.google.com/patents/US9299400

[26] K. Bains, J. Halbert, C. Mozak, T. Schoenborn, and Z. Greenfield, "Row hammer refresh command," Jan. 12 2016, uS Patent 9,236,110. [Online]. Available: https://www.google.com/patents/US9236110

[27] Z. Greenfield and T. LEVY, "Throttling support for row-hammer counters," Feb. 2 2016, uS Patent 9,251,885. [Online]. Available: https://www.google.com/patents/US9251885

[28] S. Mandava, B. Morris, S. Sah, R. Stevens, T. Rossin, M. Stefaniw, and J. Crawford, "Techniques for determining victim row addresses in a volatile memory," Feb. 23 2016, uS Patent 9,269,436. [Online]. Available: https://www.google.com/patents/US9269436

[29] J. Halbert and K. Bains, "Method, apparatus and system for responding to a row hammer event," Mar. 15 2016, uS Patent 9,286,964. [Online]. Available: https://www.google.com/patents/US9286964

[30] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," 2015. [Online]. Available: https://googleprojectzero.blogspot.de/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

[31] R. Qiao and M. Seaborn, "A new approach for rowhammer attacks," *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 161–166, 2016.

[32] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," *CoRR*, vol. abs/1507.06955, 2015. [Online]. Available: http://arxiv.org/abs/1507.06955

[33] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[34] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using ksm," in *Proceedings of the linux symposium*. Citeseer, 2009, pp. 19–28.

[35] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 199–212.

[36] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed., vol. 839. Springer, 1994, pp. 174–187. [Online]. Available: http://dx.doi.org/10.1007/3-540-48658-5_19

[37] C. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, ser. Lecture Notes in Computer Science, G. Brassard, Ed., vol. 435. Springer, 1989, pp. 239–252. [Online]. Available: http://dx.doi.org/10.1007/0-387-34805-0_22

[38] Y. Yoo, R. Azarderakhsh, A. Jalali, D. Jao, and V. Soukharev, "A post-quantum digital signature scheme based on supersingular isogenies," in *Financial Cryptography*, 2017.

[39] C. F. Kerry, "Digital signature standard (dss)," *National Institute of Standards and Technology*, 2013.

[40] ANSI, *ANSI X9.62, Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, November 2005, american National Standards Institute, X9-Financial Services.

[41] P. Véron, "Improved identification schemes based on error-correcting codes," *Appl. Algebra Eng. Commun. Comput.*, vol. 8, no. 1, pp. 57–69, 1996. [Online]. Available: http://dx.doi.org/10.1007/s002000050053

[42] J. Stern, "A new identification scheme based on syndrome decoding," in *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, ser. Lecture Notes in Computer Science, D. R. Stinson, Ed., vol. 773. Springer, 1993, pp. 13–21. [Online]. Available: http://dx.doi.org/10.1007/3-540-48329-2_2

[43] L. D. Feo, D. Jao, and J. Plût, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," *J. Mathematical Cryptology*, vol. 8, no. 3, pp. 209–247, 2014. [Online]. Available: http://dx.doi.org/10.1515/jmc-2012-0015

[44] C. Kaufman, "RFC 4306 Internet Key Exchange (IKEv2) Protocol," 2005.

[45] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "RFC 7296 Internet Key Exchange Protocol Version 2 (IKEv2)," 2014.

[46] S. Frankel and S. Krishnan, "RFC 6071 IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," 2011.

[47] S. Santesson, R. Ankney, M. Myers, A. Malpani, S. Galperin, and D. C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," RFC 6960, Jun. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6960.txt

[48] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 1675–1689. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978406

[49] A. K. Lenstra, "Memo on rsa signature generation in the presence of faults," Oct. 1996, manuscript.

[50] F. Weimer, "Factoring RSA Keys With TLS Perfect Forward Secrecy," Sep. 2015. [Online]. Available: https://people.redhat.com/~fweimer/rsa-crt-leaks.pdf

[51] Bundesamt für Sicherheit in der Informationstechnik, "AIS 20: Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren," 2013. [Online]. Available: http://www.bsi.bund.de/

[52] ——, "AIS 31: Funktionalitätsklassen und Evaluationsmethodologie für physikalische Zufallszahlengeneratoren," 2013. [Online]. Available: http://www.bsi.bund.de/

[53] T. Perrin, "The xeddsa and vxeddsa signature schemes," *Specification. Oct*, 2016. [Online]. Available: https://signal.org/docs/specifications/xeddsa/xeddsa.pdf

[54] A. Joux, "Authentication failures in NIST version of GCM," 2007, http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/GCM/Joux_comments.pdf.

[55] K. G. Paterson, "Countering cryptographic subversion," 2015, https://hyperelliptic.org/PSC/slides/paterson-PSC.pdf [Accessed 2016-05-13].

[56] S. Gueron and V. Krasnov, "The fragility of aes-gcm authentication algorithm," in *Information Technology: New Generations (ITNG), 2014 11th International Conference on*, April 2014, pp. 333–337, preprint available at https://eprint.iacr.org/2013/157.pdf.

[57] H. Böck, A. Zauner, S. Devlin, J. Somorovsky, and P. Jovanovic, "Nonce-disrespecting adversaries: Practical forgery attacks on gcm in tls," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*. Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/bock

[58] I. Biehl, B. Meyer, and V. Müller, "Differential fault attacks on elliptic curve cryptosystems," in *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '00. London, UK, UK: Springer-Verlag, 2000, pp. 131–146. [Online]. Available: http://dl.acm.org/citation.cfm?id=646765.704124

[59] B. B. Brumley, M. Barbosa, D. Page, and F. Vercauteren, "Practical realisation and elimination of an ecc-related software bug attack," in *Topics in Cryptology – CT-RSA 2012*, ser. Lecture Notes in Computer Science, O. Dunkelman, Ed. Springer Berlin Heidelberg, 2012, vol. 7178, pp. 171–186. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-27954-6_11

[60] D. Brumley and D. Boneh, "Remote timing attacks are practical," in *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, ser. SSYM'03. USENIX Association, Jun. 2003.

[61] B. Brumley and N. Tuveri, "Remote Timing Attacks Are Still Practical," in *Computer Security - ESORICS 2011*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Sep. 2011, vol. 6879.

[62] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, "Ecdsa key extraction from mobile devices via nonintrusive physical side channels," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 1626–1638. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978353

[63] S. Pelissier and Y. Romailler, "Practical fault attack against the ed25519 and eddsa signature schemes," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Sep. 2017.

[64] D. J. Bernstein, "Eddsa software," 2017, https://ed25519.cr.yp.to/python/ed25519.py.

[65] N. Samwel, L. Batina, G. Bertoni, J. Daemen, and R. Susella, "Breaking ed25519 in wolfssl," Cryptology ePrint Archive, Report 2017/985, 2017, http://eprint.iacr.org/2017/985.

[66] S. Govindavajhala and A. W. Appel, "Using memory errors to attack a virtual machine," in *Security and Privacy, 2003. Proceedings. 2003 Symposium on*. IEEE, 2003, pp. 154–165.

[67] S. B. D. Mukhopadhyay, "Curious case of rowhammer: Flipping secret exponent bits using timing analysis," Cryptology ePrint Archive, Report 2016/618, 2016. [Online]. Available: http://eprint.iacr.org/2016/618

[68] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup est machina: Memory deduplication as an advanced exploitation vector," in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 987–1004. [Online]. Available: https://doi.org/10.1109/SP.2016.63

[69] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 19–35. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/xiao