To appear in N. Smart, Ed., *Topics in Cryptology – CT-RSA 2018*, vol. 10808 of *Lecture Notes in Computer Science*, Springer, 2018.

Differential Attacks on Deterministic Signatures

Christopher Ambrose[1], Joppe W. Bos[2], Björn Fay[1], Marc Joye[3], Manfred Lochter[4], and Bruce Murray[1]

[1] NXP Semiconductors, Hamburg, Germany
[2] NXP Semiconductors, Leuven, Belgium
[3] NXP Semiconductors, San Jose, CA, USA
[4] Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn, Germany

Abstract. Deterministic signature schemes are becoming more popular, as illustrated by the deterministic variant of ECDSA and the popular EdDSA scheme, since eliminating the need for high-quality randomness might have some advantages in certain use-cases. In this paper we outline a range of differential fault attacks and a differential power analysis attack against such deterministic schemes. This shows, contrary to some earlier works, that such signature schemes are *not* naturally protected against such advanced attacks. We discuss different countermeasures and propose to include entropy for low-cost protection against these attacks in scenarios where these attack vectors are a real threat: this does not require to change the key generation or the verification methods and results in a signature scheme which offers high performance and security for a wide range of use-cases.

Keywords: Public-key algorithms; Elliptic curve cryptography; Digital signatures; Implementation attacks and defenses; Hardware security.

1 Introduction

The computation of cryptographically secure digital signatures is one of the cornerstones in public-key cryptography. This widely used cryptographic primitive is standardized in the digital signature standard [32]. The popular version of the digital signature scheme which uses elliptic curves is denoted ECDSA and is a variant of the classic signature system introduced by ElGamal [19]. This scheme (as we recall in Section 2) requires to compute a random number used only once (denoted nonce) when signing a message.

Since it might be non-trivial to obtain a good pool of entropy in practice (cf. [30,26]) and due to some noticeable failures [16] people started to deploy *deterministic* signature schemes where such randomness is not required. One such proposal modifies the existing ECDSA algorithm [38] while another popular digital signature approach uses recent developments in the field of elliptic curve cryptography: this approach is called EdDSA [8] and uses a new curve model [18,10] for performance considerations. To illustrate, it is shown that the performance of using Curve25519 [6] (which is used in the EdDSA proposal) is over twice as fast

compared to state-of-the-art implementation of NIST P-256 [25] as proposed in the digital signature standard at a comparable security level. See also [31].

The main advantage of these new deterministic digital signature proposals is clear: they don't need a good entropy pool during signing. However, when such schemes are standardized this means they need to be supported in other use-cases and settings which might have a different security model. Examples of such use-cases include (hardware) implementations as used in smart cards and for the Internet-of-Things (IoT). In these settings the adversary might own (or have access to) the target device and use meta-information when executing the cryptographic implementation. Besides such passive *side-channel attacks* (cf. [28,29]) one also has to guard the implementation against active attacks such as fault-injection attacks [14,12] and use the potentially corrupted output to obtain information about the secret key used.

Although this security model, where techniques such as faults and advanced side-channel attacks are considered, is often overlooked by the cryptographic software community (since they often do not directly apply) this is a very relevant area for industry dealing with cryptographic hardware implementations and embedded devices. The impact of this security model is expected to grow significantly in the next few years: to illustrate, the current forecasts expect 8.4 billion connected "things" in use worldwide in 2017 and will reach 20.4 billion by 2020 [23]. If one wants to secure such devices then these need to perform, among others, cryptographically secure digital signatures. For IoT devices which deal with sensitive (e.g., medical or privacy related) information then such a higher level of security protection against active and passive attacks might become a requirement.

There is an active research community which deals with such side-channel attacks and a broad amount of cryptanalytic work related to fault and side-channel attacks on ECDSA as we recall in Section 2. Surprisingly, there is not much work related to deterministic signatures. As far as we are aware the only published result related to cryptographic faults and deterministic signatures is [4]. It is demonstrated how with the help of a single correct-fault signature pair the secret key can be extracted from deterministic version of DSA and ECDSA while they conclude that the "*EdDSA algorithm shows structural resistance against such attacks.*"

It should be noted that recently a side-channel attack was pointed in [24] against Curve25519 when no validation of input points is performed as recommended in the original paper. Another recent result confirms the possibility of Rowhammer attacks on deterministic signatures. In [37] a fault attack on EdDSA is described: the attack is performed in a cloud scenario, and assumes an attacker whose virtual machine is co-located with the victim's virtual machine. The results of [37] were already announced in comments on FIPS 186-4 [33].

After this paper appeared online and independently of this work, the authors of [39] also published a differential fault attack against the deterministic signature scheme EdDSA. The presented attack is the same as the one we present in Section 3.6. It should be noted that the countermeasure described in [39] is not

sufficient since one could still succeed and extract the secret key by using the other differential attacks outlined in Section 3. Another independent work [40] shows that electromagnetic leakage in the message schedule of the hash computation in the deterministic signature scheme EdDSA can be used to derive the secret key. This is the same attack as the one we describe in Section 3.9.

Our contributions. In this work we study the impact of fault and side-channel attacks on deterministic digital signature schemes in more details. More specifically, we use the popular scheme EdDSA [8][5] as a use-case and illustrate *nine* different attacks on this scheme (but also show how these apply similarly to the deterministic ECDSA algorithm) in Section 3. This contradicts the conclusions from [4] where structural resistance against such attacks is claimed. We apply (single) faults in a different manner (compared to [4]) which results in a family of fault attacks against these new types of deterministic signature schemes.

In Section 4 we discuss practical countermeasures against these new fault attacks. However, these new safe-guards come at the price of a significant performance impact which significantly reduces the benefits when using such new digital signature approaches. We also propose a countermeasure which is not fully compliant with the current specification of the signature. The idea is to add some random noise to the input of the hash computation on platforms where such fault attacks are relevant. The verification method of the signature scheme remains unchanged but the signature scheme is no longer deterministic (in the sense that two messages always generate the same signature). We hope that this proposal can serve as additional input to the ongoing discussion and preparations for a new digital signature standard.

2 Preliminaries

The main idea behind fault attacks is to introduce a fault during the execution of the cryptographic algorithm and hope that this incorrect behavior leaks information about the secret key used. Examples related to elliptic curve cryptography include introducing a fault in one of the coefficients of the elliptic curve equation such that computations are performed on a different (weak) curve or using a different base point [17,21]. Another possibility is a sign change attack where the sign change of intermediate points can be used to recover the secret scalar factor [13,41,3].

Another type of fault attack is known as differential fault attack (DFA) where the idea is to use the difference between a faulty and a correct result to determine information about the secret key used (see [11] for the application of DFA to the elliptic curve scalar multiplication). This is the type of attack we are concerned with in this paper. The interested reader is referred to [27] and the surveys [20, Section 4] and [15] for more references and related work.

[5] See for example the "Things that use Ed25519" webpage https://ianix.com/pub/ed25519-deployment.html

Algorithm 1 ECDSA signature generation of a message m with the secret key d. The signature related parameters are as recalled in Section 2.1.

1: **function** ECDSA_SIGN(m, d)
2: $e = \mathcal{H}(m)$
3: **repeat**
4: **repeat**
5: Select $u \in [1, n-1]$ uniform random
6: $(x, y) = uG \in E_b(\mathbb{F}_p)$
7: $r = x \bmod n$
8: **until** $r \neq 0$
9: $s = u^{-1}(e + dr) \bmod n$
10: **until** $s \neq 0$
11: **return** (r, s)

Algorithm 2 Deterministic ECDSA signature generation of a message m with the secret key d. The signature related parameters are as recalled in Section 2.1.

1: **function** DETECDSA_SIGN(m, d)
2: $e = \mathcal{H}(m)$
3: **repeat**
4: **repeat**
5: $u =$ GENERATEU(d, e) using HMAC based DRNG (stateful)
6: $(x, y) = uG \in E_b(\mathbb{F}_p)$
7: $r = x \bmod n$
8: **until** $r \neq 0$
9: $s = u^{-1}(e + dr) \bmod n$
10: **until** $s \neq 0$
11: **return** (r, s)

We consider two types of fault: either an uncontrolled or a controlled fault during some target operation. With a controlled fault we mean the ability to inject a fault in a target memory range. For instance, flipping a bit in a byte, word or any range. These types of attacks are more difficult and expensive but still realistic (cf. [2]).

2.1 (Deterministic) ECDSA

In the digital signature standard [32] the randomized version of ECDSA is outlined together with some pseudo-random curves of prime order n. These curves are defined in their $a = -3$ short Weierstrass form $E_b : y^2 = x^3 - 3x + b$. These curves are defined over prime field $\mathbb{F}_p$ where $p > 3$. A generator $G \in E_b(\mathbb{F}_p)$ of order n is specified. The private key is a uniform random non-zero residue $d \in \mathbb{Z}_n$, in the range $[1, n-1]$, which defines the public key point $Q = dG$. The exact algorithm is outlined in Algorithm 1 where $\mathcal{H}$ is a cryptographic hash function. If we refer to ECDSA we mean this version which uses randomized nonces as selected in Line 5 in Algorithm 1.

A deterministic variant of ECDSA is described in an Internet Engineering Task Force (IETF) request for comments (RFC) [38]. The keys used are the same as in the randomized version of ECDSA and signatures remain valid with ECDSA: hence, no change to the verification is needed. The only change is how the nonce u is generated; in the deterministic variant this is done by an HMAC based DRNG derived from the HMAC_DRBG pseudorandom number generator, described in [34] and Annex D of [1]. This ensures that given the same message and secret key the same value u is generated.

We note that this RFC [38] explicitly acknowledges side-channel attacks as a serious threat and states that the implementer should "use defensive measures to avoid leaking the private key through a side channel" without stating how this should be done. Active attacks such as fault attacks are not addressed or considered.

2.2 EdDSA

The Edwards-curve Digital Signature Algorithm (EdDSA) is a variant of a Schnorr signature system [42] and specifies a *deterministic* digital signature algorithm using Edwards curves [18,10]. A generalized description of EdDSA takes the following eleven parameters [9]. One needs an odd prime (power) q which is used to define the finite field $\mathbb{F}_q$. Two elements $a, d \in \mathbb{F}_q$ which define the twisted Edwards curve $E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$ with an element $B \in E_{a,d}(\mathbb{F}_q)$ different from the neutral element. An integer c and odd prime ℓ which define the cardinality of the curve ($2^c\ell = \#E_{a,d}$), an integer n which determines the scalar size, an encoding of the finite field elements, and a "prehash" function $\mathcal{H}_1$. Moreover, an integer parameter b is chosen such that $2^{b-1} > q$. This determines the size of the signature ($2b$ bits) and the length of the output of a cryptographic hash function $\mathcal{H}_2$ ($2b$ bits). How to properly choose these parameters is outside the scope of this document. It should be noted that besides the encoding of finite field elements (which we denote with ENC_{INT}) one also encodes elliptic curve points (in order to reduce the number of bytes required to represent elliptic curve points) which we denote with $\text{ENC}_{\text{POINT}}$.

An EdDSA secret key is a b-bit value k while the public key is the b-bit $\text{ENC}_{\text{POINT}}(A)$. The elliptic curve point is defined as $A = sB \in E_{a,d}(\mathbb{F}_q)$, the scalar $s = 2^n + \sum_{c \le i < n} 2^i h_i$ where the h_i are in turn obtained from the output of the hashed secret key as $\mathcal{H}_2(k) = (h_0, h_1, \ldots, h_{2b-1})$.

The deterministic signature generation procedure is outlined in Algorithm 3.

3 Attacks against Deterministic Signature Schemes

In this section we describe several differential fault attacks and one side-channel attack on the deterministic signature scheme EdDSA. It should be noted that these attacks are not EdDSA specific but apply to any deterministic signature scheme (following the same design approach). The main difference between the deterministic and randomized signature schemes is how the nonce is generated.

Algorithm 3 EdDSA signature generation of a message m with the secret key k. The signature related parameters are as recalled in Section 2.2.

function EdDSA_sign((m, k))
  $m' = \mathcal{H}_1(m)$
  Retrieve or compute $(h_b, \ldots, h_{2b-1})$ from $\mathcal{H}_2(k) = (h_0, h_1, \ldots, h_{2b-1})$
  $r = \mathcal{H}_2(h_b, \ldots, h_{2b-1}, m') \bmod \ell$
  $R = rB \in E_{a,d}(\mathbb{F}_q)$
  $t = \mathcal{H}_2(\text{EncPoint}(R), \text{EncPoint}(A), m')$
  $S = (r + ts) \bmod \ell$
  return $(\text{EncPoint}(R), \text{EncInt}(S))$

While this is done using a (truly) random number generator when using a randomized version this is typically a function of the input message and the private key for deterministic schemes. This immediately highlights the problems between the typical hardware and software platforms: randomness is difficult to get and expensive from a performance point of view in software while not a problem in hardware (with some notable exceptions [7]). While computing a function on the secret key can be done trivially in software this needs very careful and expensive countermeasures in the security model used in cryptographic hardware implementations.

There are sophisticated lattice attacks on signature schemes which only require that the attacker is able to recover some bits of the ephemeral key for a certain number of signatures [35]. Typically one tries to recover the three least significant bits of the ephemeral key for, say, 300 signatures and one is then able to compute the victim's secret key.

Our high-level idea when performing a fault attack against EdDSA is to introduce a *single* fault at some point in the computation. Depending on the attack scenario this could be an uncontrolled fault somewhere during the computation or a controlled fault introduced in a pre-determined range (e.g., multiple bits, byte, or word). This fault alters the output of the signature generation procedure and allows an attacker to solve a (simple) system of equations and extract the secret key. We also present a passive attack where based on the power or electromagnetic information a side-channel attack might be mounted on the hash-function used in the deterministic signature scheme.

An overview of the points of attack, the type of attack and the number of faults needed to extract the secret key against EdDSA is given in Table 1. Similar attacks can be mounted on deterministic ECDSA as listed in Table 2. These attacks are outlined in more detail in the next subsections.

3.1 DFA on Base Point B During Import

At some stage in the cryptographic implementation the generator or base point B, which is public and given in the EdDSA signature definition, is loaded in order to perform the elliptic curve scalar multiplication with the deterministic nonce.

Table 1. Overview of the proposed attacks against EdDSA which result in extracting the private key s.

where	attack	type	number of faults
Import point B	fault	uncontrolled	≥ 1
Import point A	fault	controlled	≥ 1
Hash computation of r	fault	controlled	≥ 1
Hash computation of r with fixed (unknown) output	fault	uncontrolled	≥ 1
Scalar multiplication rB	fault	uncontrolled	≥ 1
Hash computation of t	fault	controlled	≥ 1
Hash computation of t with fixed (unknown) output	fault	controlled	≥ 2
Computation of S	fault	controlled	≥ 1
Hash computation of r	DPA/DEMA	–	–

Table 2. Overview of the proposed attacks against deterministic ECDSA which result in extracting the private key d.

where	attack	type	number of faults
Import point G	fault	uncontrolled	≥ 1
Hash computation of u	fault	controlled	≥ 1
Hash computation of u with fixed (unknown) output	fault	uncontrolled	≥ 1
Scalar multiplication uG	fault	uncontrolled	≥ 1
Computation of s	fault	controlled	≥ 1
Generation of u	DPA/DEMA	–	–

If a fault is introduced in this generator (potentially resulting in a value which is not a valid point on the curve anymore) then one could obtain a valid signature value (R, S) and an invalid one (R', S') for the same input message which represent

$$(R, S) = (rB, r + ts \bmod \ell)$$
$$(R', S') = (rB', r + t's \bmod \ell)$$

where $t' = \mathcal{H}_2(\textsc{Enc}_{\textsc{Point}}(R'), \textsc{Enc}_{\textsc{Point}}(A), m')$. All input values for the hash computation of t and t' are either known (A and m') or output by the algorithm (R and R'). Hence, both t and t' are known as well. This means the adversary can compute the secret key s from

$$S - S' \equiv s(t - t') \bmod \ell$$

where all other values are known.

3.2 DFA on Public Key A During Import

The idea here is similar to the one described in Section 3.1 but requires additional effort. The point of attack is the public key A during the import in the digital signature computation. If one can introduce a controlled fault in A in a restricted range, say ranging from bits i to j (where $0 \leq i \leq j \leq \lfloor \log_2(q) \rfloor$) then one could generate two signatures (R, S) and (R, S'), one with the original public key A and one with another (modified) public key A' for the same input message which represent

$$(R, S) = (rB, r + ts \bmod \ell)$$
$$(R, S') = (rB, r + t's \bmod \ell)$$

where $t' = \mathcal{H}_2(\text{ENC}_{\text{POINT}}(R), \text{ENC}_{\text{POINT}}(A'), m')$. If the number of bits $j - i + 1$ in the range where we introduced a fault is small enough (can be computationally enumerated) then the adversary can try all possible values for A' and hence t'. Hence, the adversary can compute the candidate secret key s from

$$S - S' \equiv s(t - t') \bmod \ell$$

and check if the right A' was used by verifying if $A = sB$. If so, the secret key has been successfully extracted.

3.3 DFA on Hash Computation of r

The point of attack is the *hash computation* of the nonce value r. Similar as in Section 3.2 the assumption is that the adversary can introduce a fault in the hash function computation which modifies only a limited number of bits in the digest value. More specifically, we assume the introduced fault $\hat{e}$ results in a nonce $r' = r + \hat{e}$. Hence, if one manages to generate two signatures (R, S) and (R', S'), one with the original scalar r and one with such scalar r', for the same input message then we have the following equations

$$(R, S) = (rB, r + ts \bmod \ell)$$
$$(R', S') = (r'B, r' + t's \bmod \ell)$$

with $t' = \mathcal{H}_2(\text{ENC}_{\text{POINT}}(R'), \text{ENC}_{\text{POINT}}(A), m')$. If the introduced error $\hat{e}$ in r' is limited then one could exhaustively try all possibilities for $\hat{e}$ and hence $R' = R + \hat{e}B$ and $t' = \mathcal{H}_2(\text{ENC}_{\text{POINT}}(R'), \text{ENC}_{\text{POINT}}(A), m')$. This results (again) in a simple system of equations which can be solved and checked if the right $\hat{e}$ was used (by checking $A = sB$). If so, the secret key has been extracted successfully.

3.4 DFA on Hash Computation of r with Fixed Output

The fault attack described here is a variation of the one described in Section 3.3. The point of attack is still the deterministic nonce r but now we assume an adversary can introduce one or more faults which result in the same value of r'

which could be *unknown* to the adversary. One can think of multiple scenarios to achieve this in practice: examples include skipping the call of the hash function, during loading of the hash input, update of the hash state or copy of the hash result. Once this has been achieved the adversary has the equations

$$(R', S_1) = (r'B, r' + t_1 s \bmod \ell)$$
$$(R', S_2) = (r'B, r' + t_2 s \bmod \ell)$$

with $t_2 = \mathcal{H}_2(\text{ENC}_{\text{POINT}}(R'), \text{ENC}_{\text{POINT}}(A), m_2')$. Again one can compute the secret key from $S_1 - S_2$ since all other values except s are known.

3.5 DFA on Scalar Multiplication

Another possible point for a fault attack is the elliptic curve scalar multiplication rB. If the adversary could introduce an uncontrolled fault during this computation then it could generate two signatures with the same input

$$(R, S) = (rB, r + ts \bmod \ell)$$
$$(R', S') = (R', r + t's \bmod \ell)$$

with $t' = \mathcal{H}_2(\text{ENC}_{\text{POINT}}(R'), \text{ENC}_{\text{POINT}}(A), m')$ and some $R' \neq R$. Please note that we have the correct r in the equation of S' instead of r' since the fault was introduced in the scalar multiplication and not in the value of r. Again the secret key can be extracted from $S - S'$ since all values are known in this equation except s.

3.6 DFA on Hash Computation of t

One could also introduce a controlled fault in the computation of the value t. If one can introduce this fault such that the faulty t' differs in a restricted range, say ranging from bit i to j (where $0 \leq i \leq j \leq \lfloor \log_2(q) \rfloor$) then one could generate two signatures (R, S) and (R, S') as follows

$$\begin{aligned}(R, S) &= (rB, r + ts \bmod \ell) \\ (R, S') &= (rB, r + t's \bmod \ell)\end{aligned} .$$

Hence, the adversary can compute the candidate secret key s from $S - S' \equiv s(t - t') \bmod \ell$ and check if the right t' was used by verifying if $A = sB$. If so, the secret key has been successfully extracted.

3.7 DFA on Hash Computation of t with Fixed Output

In the same vein as in Section 3.4 one could introduce two controlled faults to generate digital signatures (R_1, S_1) and (R_2, S_2) for two different messages m_1 and m_2, both with an unknown but fixed value t'. Such faults could be introduced in multiple places: for example, skipping the call of the hash function, during

loading of the hash input, update of the hash state or copy of the hash result. Next, generate the original two signatures (R_3, S_3) and (R_4, S_4) for the same messages m_1 and m_2. Then one obtains the following four equations

$$\begin{aligned} S_1 &= r_1 + t's \\ S_2 &= r_2 + t's \\ S_3 &= r_1 + t_1 s \\ S_4 &= r_2 + t_2 s \end{aligned} .$$

Given this information one can compute

$$S_3 - S_4 - (S_1 - S_2) = (r_1 - r_2) + (t_1 - t_2)s - (r_1 - r_2) = (t_1 - t_2)s$$

and the secret key s can be extracted.

3.8 DFA on Computation of S

If the adversary manages to generate two signatures (R, S) and (R, S'), one with the correct computation of S and one with faulty computation of S, then the secret key can be extracted. The faulty value S' is obtained by skipping one of the elementary arithmetic operations in $S = r + ts$. Hence, depending on the fault one obtains

$$\begin{aligned} S' &= ts \\ S' &= r + t \\ S' &= r + s \end{aligned} .$$

Depending on the case the adversary can compute $S - S' = r$, $S - S' = t(s-1)$ or $S - S' = (t-1)s$, respectively. In all three cases one can compute r or s (and then s or r).

3.9 DPA/DEMA on $h_b, \ldots, h_{2b-1}$ during Hash Computation of r

Instead of using an active attack, such as inserting fault(s), one could mount a passive attack based on either power consumption (such as the differential power analysis (DPA) attack [29]) or electromagnetic usage (such as differential electromagnetic analysis (DEMA) attacks [22]). In order for such an attack to be successful one needs to target a point in the algorithm where computation is performed on the secret together with some known data such that a differential attack can be mounted.

The main idea for such a passive attack is to target the computation of the message digest $\mathcal{H}_2(h_b, \ldots, h_{2b-1}, m')$ where both secret key derived material and user provided data are used as input. Whether such a passive attack is feasible depends on the exact choice of the hash function and the value of b. In EdDSA the hash function $\mathcal{H}$ used is SHA-512 and $b = 256$, hence the input to the hash function is processed in chunks of 128 bytes (or 1024 bits). Since 256 bits of secret-key derived material is used as input this means the first 1024-bit chunk

processed by the SHA-512 contains both secret key and user controlled input. Hence, a DPA/DEMA attack in the usual way should be possible and the bits $h_b, \ldots, h_{2b-1}$ can be extracted one at-a-time. This seems indeed feasible since a similar approach on HMAC based on SHA-256 is presented in [5].

4 Countermeasures for EdDSA

In this section we describe two different sets of countermeasures against the attacks we presented in Section 3. The first countermeasure does fully comply to the EdDSA specification while the second one does not: however, this last set of countermeasures does generate valid EdDSA signatures. This can only be distinguished from fully compliant signatures by the signer or by seeing the same message with two different signatures.

4.1 Fully Compliant Countermeasures

For some of the proposed attacks it might be sufficient to check if the targeted elliptic curve points are valid by checking the curve equation: this is true for the attacks from Section 3.1, Section 3.2, and Section 3.5. However, in order to protect against the other fault attacks (from Section 3.3, Section 3.4, and Section 3.6 to Section 3.8) it seems double computation and a comparison of results seems the only practical countermeasure. In order to protect against the side-channel attack from Section 3.9 one needs to harden the hash computation, which will result in much slower hash computation. A guesstimate, based on our experience implementing such countermeasures, of the practical impact of these countermeasures on the performance is around two to three times slower.

4.2 Not Fully Compliant Countermeasures

In this section we outline effective countermeasures against our proposed attacks. These techniques do not fully comply with the way how the deterministic signature algorithm states one needs to generate the nonces (see Algorithm 3). However, the proposed techniques are significantly faster compared to the compliant countermeasures considered in Section 4.1.

A much *simpler* countermeasure, which randomized the signature algorithm, is to include some noise in the computation of $r = \mathcal{H}_2(h_b, \ldots, h_{2b-1}, m') \bmod \ell$. By adding some uniform random noise one ensures some variable unknown data is introduced in the various equations from Section 3. One way of achieving this is by splitting the input to the hash function into three hash input blocks:

1. random noise (and/or counter),
2. secret input $(h_b, \ldots, h_{2b-1})$ and
3. prehashed message m'.

The amount of random noise depends on the needs and the targeted security level: hence, for SHA-512 noise with 256 bits of entropy is needed, but also less might be sufficient to actually protect against collisions for practical real-world attacks. If there is no random source available but non-volatile memory, one could also use an unknown counter. This of course leads to a stateful signing operation, but still stateless verification. This way the first input block to the hash function generates "unique" unknown (random) data to combine with the secret second block. The second block only consists of unknown secret data $(h_b, \ldots, h_{2b-1})$ and known public fixed (padding zeros) data. After processing the second block the hash state has full entropy coming from the secret data (but same for all signatures) and is different for each signature including some entropy coming from the noise. That means no easy collisions can be obtained and one cannot predict values easily. This is exactly what the "XEdDSA" approach [36], which enables use of a single key pair format for both elliptic curve Diffie-Hellman and signatures, is proposing.

This countermeasure protects against DFA and DPA/DEMA attacks but of course one is still vulnerable against SPA and template attacks on the hash computation of r. One possible solution to this would be to simply use a fully random nonce r provided by an RNG. For this solution an RNG of sufficient "good" quality is required, which might not be available on all platforms. However, in settings where such a level of security is often mandatory, e.g. on smart cards, one typically has access to a high-quality-RNG on board. This makes it significantly easier and faster to use the RNG instead of doing any hash computation.

Hence, the best solution (in terms of performance) would actually be an adaptive solution depending on the availability of a high-quality-RNG. Such a solution would offer high-security guarantees on platforms where active and passive attacks can be expected (and where often acquiring good entropy is not a problem) while it offers the same performance and security advantages in the pure software setting for the deterministic schemes as used today.

5 Conclusions and Future Work

We have presented a number of active attacks and one passive side-channel attack against deterministic signature schemes. This highlights that removing randomness from the equation does necessarily eliminate all attack vectors. Countermeasures which need to comply with the current specification of, for instance, EdDSA seem to have a significant performance impact: the resulting protected schemes seem to have no real performance benefits over the current standardized (randomized) ECDSA algorithm. However, if one is willing to slightly deviate from the specification and introduce high-quality randomness on platforms where this is possible then relatively cheap countermeasures can be constructed without affecting either the key generation and signature verification procedures.

In this work we only looked at "simple" single differential fault attacks. Future work include more advanced attacks (active and passive attacks) as well

as introducing multiple faults. Of course it would be very interesting to study other more advanced countermeasures which either do comply directly with the current deterministic signature specification or can be computed more efficiently.

We hope this work serves as valuable input when the community and the various standardization bodies start to define new cryptographic digital signature algorithms. In our opinion such a hybrid scheme (where the user can choose to include randomness or not) is a valuable addition to achieve a higher level of security.

Acknowledgments. We would like to thank Laurie Genelle for her comments on an earlier version of this paper.

References

1. American National Standards Institute: Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), ANSI X9.62-2005 (Nov 2005)
2. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. Proceedings of the IEEE 100(11), 3056–3076 (Nov 2012)
3. Barenghi, A., Bertoni, G.M., Breveglieri, L., Pelosi, G., Sanfilippo, S., Susella, R.: A fault-based secret key retrieval method for ECDSA: Analysis and countermeasure. ACM Journal on Emerging Technologies in Computing Systems 13(1), 8:1–8:26 (2016)
4. Barenghi, A., Pelosi, G.: A note on fault attacks against deterministic signature schemes. In: Ogawa, K., Yoshioka, K. (eds.) IWSEC 16. LNCS, vol. 9836, pp. 182–192. Springer, Heidelberg (Sep 2016)
5. Belaïd, S., Bettale, L., Dottax, E., Genelle, L., Rondepierre, F.: Differential power analysis of HMAC SHA-2 in the Hamming weight model. In: Samarati, P. (ed.) SECRYPT. pp. 230–241. IEEE (2013)
6. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (Apr 2006)
7. Bernstein, D.J., Chang, Y.A., Cheng, C.M., Chou, L.P., Heninger, N., Lange, T., van Someren, N.: Factoring RSA keys from certified smart cards: Coppersmith in the wild. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 341–360. Springer, Heidelberg (Dec 2013)
8. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg (Sep / Oct 2011)
9. Bernstein, D.J., Josefsson, S., Lange, T., Schwabe, P., Yang, B.Y.: EdDSA for more curves. Cryptology ePrint Archive, Report 2015/677 (2015), http://eprint.iacr.org/2015/677
10. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 29–50. Springer, Heidelberg (Dec 2007)
11. 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 (Aug 2000)

12. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (Aug 1997)
13. Blömer, J., Otto, M., Seifert, J.P.: Sign change fault attacks on elliptic curve cryptosystems. In: Breveglieri, L., et al. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 36–52. Springer, Heidelberg (2006)
14. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. Journal of Cryptology 14(2), 101–119 (2001)
15. BSI: Minimum requirements for evaluating side-channel attack resistance of elliptic curve implementations (2016), http://www.bsi.bund.de/
16. "Bushing", Cantero, H., Boessenkool, S., Peter, S.: PS3 epic fail (2010), http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf
17. Ciet, M., Joye, M.: Elliptic curve cryptosystems in the presence of permanent and transient faults. Designs, Codes and Cryptography 36(1), 33–43 (2005)
18. Edwards, H.M.: A normal form for elliptic curves. Bulletin of the American Mathematical Society 44, 393–422 (Jul 2007)
19. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31, 469–472 (1985)
20. Fan, J., Verbauwhede, I.: An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In: Naccache, D. (ed.) Cryptography and Security: From Theory to Applications. LNCS, vol. 6805, pp. 265–282. Springer, Heidelberg (2012)
21. Fouque, P.A., Lercier, R., Réal, D., Valette, F.: Fault attack on elliptic curve Montgomery ladder implementation. In: FDTC 2008. pp. 92–98. IEEE Computer Society (2008)
22. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (May 2001)
23. Gartner: Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. http://www.gartner.com/newsroom/id/3598917 (Feb 2017)
24. Genkin, D., Valenta, L., Yarom, Y.: May the fourth be with you: A microarchitectural side channel attack on several real-world applications of Curve25519. In: ACM CCS 2017. pp. 845–858. ACM Press (2017), also available as ePrint 2017/806, http://eprint.iacr.org/2017/806
25. Gueron, S., Krasnov, V.: Fast prime field elliptic-curve cryptography with 256-bit primes. Journal of Cryptographic Engineering 5(2), 141–151 (2015)
26. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your Ps and Qs: Detection of widespread weak keys in network devices. In: USENIX Security Symposium. pp. 205–220. USENIX, Bellevue, WA (2012)
27. Joye, M., Tunstall, M. (eds.): Fault Analysis in Cryptography. Information Security and Cryptography, Springer, Heidelberg (2012)
28. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (Aug 1996)
29. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (Aug 1999)
30. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Public keys. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 626–642. Springer, Heidelberg (Aug 2012)

31. M'Raïhi, D., Naccache, D., Pointcheval, D., Vaudenay, S.: Computational alternatives to random number generators. In: Tavares, S.E., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 72–80. Springer, Heidelberg (Aug 1999)
32. National Institute of Standards and Technology: Digital Signature Standard (DSS) (2013), Federal Information Processing Standards Publication 186-4, http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
33. National Institute of Standards and Technology: Public Comments Received on FIPS 186-4: Digital Signature Standard (DSS) (2015), http://csrc.nist.gov/groups/ST/toolkit/documents/Comments-received-FIPS-186-4-Dec2015.pdf
34. National Institute of Standards and Technology: SP 800-90A Rev. 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Jun 2015), http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
35. Nguyen, P.Q., Shparlinski, I.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. Designs, Codes and Cryptography 30(2), 201–217 (2003)
36. Perrin, T.: The XEdDSA and VXEdDSA signature schemes. Unpublished manuscript (Oct 2016), revision 1, available at https://signal.org/docs/specifications/xeddsa/
37. Poddebniak, D., Schinzel, S., Somorovsky, J., Lochter, M., Roesler, P.: Attacking deterministic signature schemes using fault attacks. In: EuroS&P 2018. IEEE Computer Society (to appear)
38. Pornin, T.: Deterministic usage of the digital signature algorithm (DSA) and elliptic curve digital signature algorithm (ECDSA). RFC 6979, https://tools.ietf.org/html/rfc6979 (2013)
39. Romailler, Y., Pelissier, S.: Practical fault attack against the Ed25519 and EdDSA signature schemes. In: FDTC 2017. pp. 17–24. IEEE Computer Society (2017)
40. Samwel, N., Batina, L., Bertoni, G., Daemen, J., Susella, R.: Breaking Ed25519 in WolfSSL. Cryptology ePrint Archive, Report 2017/985 (2017), http://eprint.iacr.org/2017/985
41. Schmidt, J.M., Medwed, M.: A fault attack on ECDSA. In: Breveglieri, L., et al. (eds.) FDTC 2009. pp. 93–99. IEEE Computer Society (2009)
42. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology 4(3), 161–174 (1991)