Lattice-based Fault Attacks on Deterministic Signature Schemes of ECDSA and EdDSA

No Author Given

No Institute Given

Abstract. The deterministic ECDSA and EdDSA signature schemes have found plenty of applications since their publication and standardization. Their theoretical security can be guaranteed under certain well-designed models, while their practical risks from the flaw of random number generators can be mitigated since no randomness is required by the algorithms anymore. But the situation is not completely optimistic, since it has been gradually found that delicately designed fault attacks can threaten the practical security of the schemes.

We present a lattice-based fault analysis method to the deterministic ECDSA and EdDSA algorithms. The underlying fault injection model is a special case of the random fault model in [14]. By noticing the algebraic structures of the deterministic algorithms, we show that, when providing with some valid faulty signatures and an associated correct signature of the same input message, some instances of lattice problems can be constructed to recover the signing key. This makes the allowed faulty bits close to the size of the signing key, and obviously bigger than that of the existing differential fault attacks. Moreover, the lattice-based approach supports much more alternative targets of fault injection when comparing with the existing approaches, which further improves its applicability.

Experiments are performed to validate the effectiveness of the key recovery method. It is demonstrated that, for 256-bit deterministic ECDSA/EdDSA, the signing key can be recovered efficiently with significant probability even if the targets are affected by 250/247 faulty bits. This is, however, impractical for the existing faulty pattern enumerating approaches.

Keywords: Side channel attack, Fault attack, Lattice-based attack, Deterministic ECDSA, EdDSA

1 Introduction

As a fundamental building block of modern cryptography, digital signature has been widely used in practice. For its efficiency and standardization in FIPS 186 and ANSI X9.62, ECDSA has found various applications since its publication. In spite of the fact that the theoretical security of ECDSA has not been prove finally, it is still believed to be secure and connected with some hard problems in mathematics. However, side channel attacks on various implementations of ECDSA have been continuously discovered during the last decades. Some of the attacks, for example, are induced by the deficiency of the ephemeral random numbers (denoted *nonce* hereinafter) required by the scheme. If the nonce is biased or has a few bits leaked, Bleichenbacher's attack [8] and lots of lattice-based approaches [16,23] can be employed to extract the private key by BDD [19]. This is realistic and has been demonstrated several times in real products [6,13,15,2]. Hence, an intuition on improving security is to remove the randomness requirement from the algorithms. This gave birth to a study on *deterministic signature schemes*. For example, in recent years deterministic ECDSA and EdDSA were introduced and standardized in RFC 6979 and RFC 8032 respectively. They have received plenty of attention in the research of applied cryptography, especially in the realization of cryptographical libraries of OpenSSH, Tor, TLS and even in the rising applications like Apple AirPlay and Blockchain. Specifically, the deterministic version of ECDSA derives the nonce just from the private key and the input message by means of cryptographic hash or HMAC primitive. In this way, no randomness is required on the implementation platform, and it seems the threat from physical attacks are mitigated.

But the situation is not improved, since new flaws in deterministic signature algorithms have been gradually identified when considering differential fault attacks (DFA) [5,28,29,30]. DFAs have been proved to be valid for different types of cryptographic schemes [7,9] in the literature. Generally, a DFA adversary manages to disturb the signature generation procedure (by means of voltage clitches, laser or electro-magnetic injection and so on [17]) to make the platform output faulty results, and exploits them to do key recovery. To have a better view about the capability and limitation of existing attacks on deterministic signature schemes, the following intends to review the known results.

Firstly, it is shown in [5] that if a fault is injected during the calculation of scalar multiplication of deterministic ECDSA or EdDSA, and results a known faulty signature (r', s'), then with the correct signature (r, s) from the same signing key d and input message m, the key d can be recovered by solving some linear equations. It is noted that, though no limitation on the number of faulty bits, the approach is limited by the possible locations (or rather *targets*) of fault injection. As a relaxation, another approach was introduced in [5] by assuming that only limited bits of the target (e.g., the nonce k) would be randomly affected by each fault. For simplicity, the faulty value is denoted by $k' = k + \varepsilon 2^l$ (with limited ε and known l). Then by constructing a differential distinguisher, the private key d in deterministic ECDSA can be recovered efficiently by enumerating ε. Both attacks have been improved later, especially by those in [28,29,30], where different fault injection methods and targets are exploited and experimented on specific hardware platforms. A recent extension was presented in [1], where several fault injection targets have been identified and analyzed.

From a common point of view, the signing key of deterministic schemes can theoretically be recovered by adjusting fault injection actions and enumerating the possible faults. The efficiency of the existing attacks is obviously constrained by the enumeration complexity, or specifically it is feasible only if the fault injection is controlled to affect very limited bits of the targets. Another limitation of the existing attacks lies in the selectability of targets that can be used for fault injection. Generally, the more targets would mean the more selectable attack paths, and thus more difficult to resist the attack. The targets that were considered in the existing attacks are very constrained. For example, the first attack in [5] only supports one possible target (i.e., the scalar multiplication kG in signatures), and although some more targets were considered later in [1], it is still far away from covering all the possible attack paths.

A promising solution is to develop lattice-based approaches. It is noticed that lattice-based attacks were used to analyze plain (EC)DSA and qDSA, in which the main purpose of fault attack is to obtain and exploit some leaked information of the random nonce k, such as those in [11,21,26,31,36]. Targets of fault injection in those approaches are usually the nonce itself or the scalar multiplication with a nonce as scalar. For those attacks to be effective, nonce in the plain signature is supposed to be random numbers. Hence it is generally thought that deterministic ECDSA is immune to them because of deterministic nonce generation. However, this conception has been disproved by [10], where a lattice-based attack to compromise deterministic signatures was presented. That attack is specific to lattice-based cryptography, and the lattice constructed for the attack is also specific to the signature scheme. Although it casts a new light on the study of lattice-based fault attacks on more deterministic signature schemes, it is still not known whether the method is effective to deterministic ECDSA or EdDSA with different algebraic structures.

In this paper, we show lattice-based fault attacks can be applied to deterministic ECDSA and EdDSA schemes. We consider the attacks in a random fault model where a continuous bits block of fault targets is disturbed randomly. It can be equivalent to the bit-wise random fault model in [14]. Under this model, a corresponding lattice-based key recovery method is proposed. Essentially, by virtue of the special algebraic structures of the signature generation algorithms, the method reduces the key recovery problem to the approximate shortest/closest problems in some lattice, with the instances of the approximate problems being constructed from the collected faulty signatures. Since the approximate problems can be solved within some scale, the signing key can be recovered subsequently (provided that the faulty signatures are valid as per some criteria).

In comparison, the advantage of our lattice-based method over the existing approaches [1,5,28,29] makes it more practical. This is summarized as follows.

- The proposed method allows more choices of target for fault injection. A target of fault injection is denoted by the notation of the interest intermediate and the timing of using it in computation. Since a general representation method of fault is adopted to remove the discrepancies of various targets, a number of possible targets are allowed by our attacks, which relatively covers more possibilities than the existing approaches. See Section 3.1 for detail.
- The proposed method can tolerate more faulty bits. The proposed lattice method is not to enumerate all the faulty patterns, but rather to solve the instances of approximate lattice problems. This makes the tolerable bits can be close to the size of the signing key. For instance, the case of faulty

bits up to 250(for 256-bit deterministic ECDSA)/247(for ed25519) has been validated in experiments efficiently. As discussed above, this is infeasible for the existing approaches. See Section 5 for detail.

The remainder of this paper is organized as follows: Section 2 describes the specification of deterministic ECDSA and EdDSA, and gives some results about lattices. Section 3 introduces the fault model. Section 4 illustrates two representative lattice-based attacks based on the described model. Section 5 describes the experimental facets of the validity of the lattice-based key recovery method. The discussion about the corresponding countermeasures is given in Section 6. More attacks with other fault targets are introduced in Appendix A.

2 Preliminaries

2.1 Notations

We denote by $\mathbb{F}_q$ the finite field of prime order q, $\mathbb{R}$ the field of real number, and $\mathbb{Z}_n$ the additive group of integer modulo n. Bold lowercase letters such as $\boldsymbol{v}$ are denoted as vectors, while bold uppercase letters such as $\mathbf{M}$ are denoted as matrix. The norm of vector $\boldsymbol{v} = (v_1, \ldots, v_N) \in \mathbb{R}^N$ is denoted by $\|\boldsymbol{v}\| = \sqrt{\sum_i^N v_i^2}$, while the multiplication of $\boldsymbol{v}$ and $\mathbf{M}$ is denoted by $\boldsymbol{v}\mathbf{M}$.

2.2 The deterministic signature algorithms

We recap the deterministic signature generation algorithms below by abstracting from some less important details in the specifications of RFC 6979 and RFC 8032 respectively. As shown in Algorithms 1 and 2, the analysis focuses on Step 6 of Algorithm 1 and Step 4 of Algorithm 2 during the signature generations, where the order n is a prime. Moreover, in EdDSA signature, the two b-bit subkeys d_0 and d_1 are derived by the hash function $H(d) = (h_0, h_1, ..., h_{2b-1})$, where d is the private key, $d_0 = 2^{b-2} + \sum_{i=3}^{b-3} 2^i h_i$ and $d_1 = (h_b, ..., h_{2b-1})$. The public key P satisfies $P = d_0 G$. The hash functions employed in deterministic ECDSA are generally SHA-1 and SHA-2(e.g., SHA-256 and SHA-512), which all belong to the structure of message digest. For EdDSA, the default hash function(i.e., $H(.)$) is SHA-512. In addition, there still exist other hash functions belonging to the sponge structure, such as SHAKE256(SHA-3) for Ed448. For the sake of simplicity, we just consider the compression function of SHA-2. As shown in Figure 1, input IV and a group of message, execute L-round compression and output the final result of compression plus IV as the hash value or the next group of IV.

2.3 Some Problems in Lattice

Since the proposed attacks on deterministic signature schemes are related to the construction and computation of some problems in lattice, we give a basic introduction on the relevant conceptions and results.

Algorithm 1 Signature generation of deterministic ECDSA

```
Input: The definition of a specific elliptic curve $\mathrm{E}(\mathbb{F}_q)$, a base point $G$ of the curve
    with order $n$, message $m$, private key $d$.
Output: Signature pair $(r, s)$.
1: $e = H(m)$, where $H$ is a cryptographic hash function;
2: Generate $k = F(d, e)$ such that $k \in [1, n-1]$, where $F(d, e)$ denotes the
   HMAC_DRBG function with $d$ as the key;
3: $Q(x_1, y_1) = kG$;
4: $r = x_1 \bmod n$;
5: if $r = 0$ then goto step 2;
6: $s = k^{-1}(e + dr) \bmod n$;
7: if $s = 0$ then goto step 2;
8: return $(r, s)$
```

Algorithm 2 Signature generation of EdDSA

```
Input: The definition of a specific elliptic curve $\mathrm{E}(\mathbb{F}_q)$, a base point $G$ of the curve
    with order $n$, message $m$, private key $(d_0, d_1)$, and public key $P(P = d_0G)$.
Output: Signature pair $(R, s)$.
1: $k = H(d_1, m) \bmod n$, where $H$ is SHA-512 by default;
2: $R(x_1, y_1) = kG$;
3: $r = H(R, P, m) \bmod n$;
4: $s = k + rd_0 \bmod n$;
5: return $(R, s)$
```

In a nutshell, a *lattice* is a discrete subgroup of $\mathbb{R}^m$, generally represented as a spanned vector space of linearly independent row vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_N \in \mathbb{R}^m$ of matrix $\mathbf{M} \in \mathbb{R}^{N \times m}$, in the form of

$$\mathcal{L} = \mathcal{L}(\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_N) = \{\boldsymbol{z} = \sum_{i=1}^{N} x_i \cdot \boldsymbol{b}_i | x_i \in \mathbb{Z}\}. \tag{1}$$

The vectors $\boldsymbol{b}_i$s are called a basis of $\mathcal{L}$, and N is the dimension of $\mathcal{L}$. If $m = N$, then $\mathcal{L}$ is full rank. Moreover, if $\boldsymbol{b}_i$ belongs to $\mathbb{Z}^m$ for any $i = 1, ..., N$, $\mathcal{L}$ is called an integer lattice. In this way, it is straightforward to find that for every $\boldsymbol{z} \in \mathcal{L}$, there must exist $\boldsymbol{x} = \{x_1, ..., x_N\} \in \mathbb{Z}^N$ such that $\boldsymbol{z} = \boldsymbol{x}\mathbf{M}$.

In lattice, a few well-known problems have been studied, such as the *shortest vector problem*(SVP) and *closest vector problem*(CVP), which are believed to be hard in computation theoretically.

SVP: given a basis $\boldsymbol{b}_i$s of $\mathcal{L}$, find a nonzero vector $\boldsymbol{v} \in \mathcal{L}$ such that

$$\|\boldsymbol{v}\| = \lambda_1(\mathcal{L}), \tag{2}$$

where $\lambda_1(\mathcal{L})$ means the length of the shortest vector in $\mathcal{L}$.

CVP: given a basis $\boldsymbol{b}_i$s of $\mathcal{L}$ and a target vector $\boldsymbol{u} \in \mathbb{R}^m$, find a nonzero vector $\boldsymbol{v} \in \mathcal{L}$ such that

$$\|\boldsymbol{v} - \boldsymbol{u}\| = \lambda(\mathcal{L}, \boldsymbol{u}), \tag{3}$$

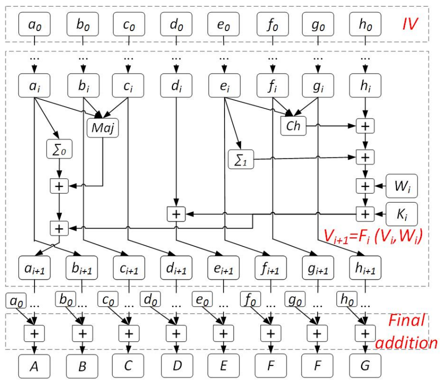

Fig. 1. Compression function of SHA2

where $\lambda\left(\mathcal{L},\boldsymbol{u}\right)$ is the closest distance from vector $\boldsymbol{u}$ to lattice $\mathcal{L}$.

Generally, the best algorithms for solving SVP and CVP are LLL algorithm [18] or BKZ algorithm [32,33,34] to find their approximate solutions, i.e., solve approximate SVP and CVP. For an N-dimensional approximate SVP, a short lattice vector can be output when the approximate factor is large enough. The approximate factor of the LLL algorithm is given from Lemma 1.

Lemma 1. *[20,18] Given an integer basis B of N-dimensional lattice $\mathcal{L}$, there exists a polynomial time algorithm to find a nonzero lattice vector $\boldsymbol{x}$ satisfying*

$$\|\boldsymbol{x}\| \leq (2/\sqrt{3})^N \lambda_1\left(\mathcal{L}\right).$$

Hence, the exact SVP and CVP can be approximated within an exponetial factor in polynomial time.

For random lattices with dimension N, Gaussian heuristic [22] expected the shortest length could be defined to be

$$\sigma(\mathcal{L}) = \sqrt{\frac{N}{2\pi e}}\mathrm{vol}(\mathcal{L})^{1/N},$$

where vol denotes the volume or determinate of $\mathcal{L}$.

Actually, the exact shortest vector of N-dimensional random lattices is much easier to be found along with the increment of the gap between the shortest length and $\sigma(\mathcal{L})$. If it is much shorter than $\sigma(\mathcal{L})$, it shall be founded in polynomial time by using LLL and related algorithms. Heuristically, as introduced in [27], assuming the lattice $\mathcal{L}$ behaves like random, if there exists a lattice vector whose distance from the target is much shorter than $\sigma(\mathcal{L})$, this lattice vector is expected to be the closest vector from the target. Accordingly, this special instance of CVP usually could be solved by Babai algorithm[4] or embedding-based SVP[25].

3 Adversarial model

In regard to fault attacks on signature schemes, the adversary is allowed to query and at the same time disturb the signing procedure to collect the correct or faulty signatures (in the *fault injection phase*), then employs the collected signatures to recover the private key (in the *key recovery phase*). The difference between various fault attacks lies in the approaches used for both fault injection and key recovery. The following describes the adversarial model for these two phases.

3.1 Fault injection model

During the fault injection phase, we assume the adversary is capable of inducing *transient* faults to some specific intermediates in computation. That is, during the invocation of signature generation, faults can be injected to the data when it is transmitted over the physical circuit (such as buses), or stored in the memory cells or CPU registers. Then, after the invocation, the computation device will restore to a normal state and the faults will not be passed on to the next invocation. In this way, the computation may be temporarily tampered to produce available faulty results for the adversary.

The fault in this paper can be regarded as a bit-wise random fault as defined in [14]. In detail, a fault induced to a specific intermediate $v \in \mathbb{Z}_n$ can be formalized as an addition with a (bounded) random value $\varepsilon \in \mathbb{Z}$ in the form of $v + \varepsilon 2^l \bmod n$, where $-2^w < \varepsilon < 2^w$, $l \geq 0$ is the known value and $l + w$ is less than the key bit length of signature. That means, there are w bits of v(starting from l-th bit) to be disturbed randomly. It is noted that we do not use modular 2 addition as in [14](e.g., $v \oplus X2^l \bmod n$ and X is a w-bit random number) but rather the equivalent group addition in $\mathbb{Z}_n$ to represent the effect of a fault to an intermediate.

To facilitate the description, the specific intermediates which may suffer from faults are called (potential) *targets* of fault injection in this paper. All the potential targets that can be exploited by the proposed attacks are listed in Table 1. It is noted that a target is determined by two factors, i.e., *the notation of the variant* (corresponding to the intermediate), and *the timing for fault injection.* For example, the two items "k before the calculation of scalar multiplication kG" and "k during the calculation of s" are recognized as two different targets in this paper. In comparison, though some of the identified targets in Table 1 have also been considered in [1], not all of them can be exploited to do key recovery in their method, especially when the target is affected by lots of faulty bits.

On the other hand, different targets may be equivalent if considering the final effect of fault injection. For example, the targets on hash function in deterministic ECDSA: "registers before outputting the hash values $F(d, e)$", "last modular additions before outputting the hash values $F(d, e)$" and "hash value $F(d, e)$ during the reduction of k" are equivalent to the target "k before the calculation of kG", since fault injection to the four targets will produce a same type of faulty k. Therefore, we define 'k before the calculation of kG" as the *representative* target of the four targets, and indicate it in **bold** type in the table. Similarly,

Table 1. The fault targets and solved problem in our attacks on deterministic ECDSA and EdDSA.

Algorithm	Target of fault injection	Related problem
Deterministic ECDSA	r during the calculation of s	SVP
	k^{-1} during the calculation of s	SVP
	k during the calculation of s	SVP
	d during the calculation of s	SVP
	e during the calculation of s	SVP
	–Registers before outputting hash value $H(m)$	
	–Last modular additions before outputting $H(m)$	
	rd during the calculation of s	SVP
	$e + rd$ during the calculation of s	SVP
	k before the calculation of kG	CVP
	–Registers before outputting hash value $F(d, e)$	
	–Last modular additions before outputting $F(d, e)$	
	–Hash value $F(d, e)$ during the reduction of k	
EdDSA	**r during the calculation of s**	SVP
	–Registers before outputting hash value $H(R, P, m)$	
	–Last modular additions before outputting $H(R, P, m)$	
	–Hash value $H(R, P, m)$ during the reduction of r	
	k before the calculation of kG	CVP
	–Registers before outputting hash value $H(d_1, m)$	
	–Last modular additions before outputting $H(d_1, m)$	
	–Hash value $H(d_1, m)$ during the reduction of k	

other representative targets are also indicated in Table 1 in the same way. Note that, all the hash functions in the targets refer to SHA-2 hash function.

In each of the proposed attack, the adversary is required to pre-determine at most one target and then fix the choice throughout the signature queries. Note that we don't consider the possibility that more than one target is chosen in a query, since the key recovery model doesn't support this case. Hence, there is no guarantee that the key can be recovered successfully. A set of faulty signatures are called *valid* if they are computed with the same message as input and the same equivalent target for fault injection.

It is noted that, since the paper aims to examine the conception that some deterministic signature schemes may be threatened by lattice-based fault attacks, we don't consider the so-called instruction skipping attacks (where the execution flow is disturbed such that some instructions are skipped without being executed) and persistent faults (i.e., permanently modifying data in the memory), though the model may be somehow extended to cover these cases.

3.2 Key recovery by solving approximate problems in lattice

When enough valid faulty results are collected, the adversary manages to recover the signing key. This section is devoted to describe the fundamental idea behind the attacks, the instantiation is left to be described in Section 4.

Intuitively, the proposed attacks in this paper exploit some special algebraic structures of the signature generation algorithm of deterministic ECDSA and EdDSA, which are discovered by the following observations.

a)**Representation of faults.** Firstly, due to the special structure of the deterministic ECDSA and EdDSA, when gathering a correct signature and $N-1$ valid faulty results for a common message, the adversary can construct one of the following two relations(corresponding to SVP and CVP) for the random faulty values $\{\varepsilon_i\}_{i=1}^{N-1} \in \mathbb{Z}$(corresponding to the faulty signatures):

$$\varepsilon_i = A_i D + h_i n, \tag{4}$$

$$\varepsilon_i = A_i D + h_i n - B_i, \tag{5}$$

with $-2^w < \varepsilon_i < 2^w < n$, where A_i, B_i, w, n are known values (with n being the order of base point G), and D, ε_i, h_i are unknown values.

In detail, D is a function of the private key, the input message and some known variables. Then it is important to notice that when the input message is known, D is reversible and subsequently the key can be recovered. This is true when the input message is not affected by the injected faults, and by the fact that the input message is chosen and known to the adversary before the attack. Thus the goal of the proposed attacks is translated to recover D.

b)**Key recovery using lattice.** Based on the above observation, we can construct a lattice $\mathcal{L}$ with a basis being the row vectors of a matrix $\mathbf{M}$ as

$$\mathbf{M} = \begin{pmatrix} n & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & n & 0 \\ A_1 & \cdots & A_{N-1} & 2^w/n \end{pmatrix}.$$

It is noted that, under the random models of faults injection, $\mathcal{L}$ behave like a random lattice. Then, a target vector $\boldsymbol{v} \in \mathcal{L}$ can be constructed from the coordinate vector $\boldsymbol{x} = (h_1, \ldots, h_{N-1}, D) \in \mathbb{Z}^N$ as

$$\boldsymbol{v} = \boldsymbol{x}\mathbf{M} = (A_1 D + h_1 n, \ldots, A_{N-1} D + h_{N-1} n, D2^w/n).$$

The given volume of $\mathcal{L}$ meets $\mathrm{vol}(\mathcal{L}) = \det(\mathbf{M}) = n^{N-2}2^w$. Under the condition of $|\varepsilon_i| < 2^w$, supposing $f = \lceil \log n \rceil$, $w < f - \log\sqrt{2\pi e}$ and $N \gg 1 + \frac{f+\log\sqrt{2\pi e}}{f-w-\log\sqrt{2\pi e}}$, one of the following relations will hold:

(i) when the faulty value is represented by equation (4), we have

$$\|\boldsymbol{v}\| < \sqrt{N}2^w \ll \sqrt{\frac{N}{2\pi e}}\mathrm{vol}(\mathcal{L})^{\frac{1}{N}}; \tag{6}$$

(ii) when the faulty value is represented by equation (5), then for vector $\boldsymbol{u} = (B_1, \ldots, B_{N-1}, 0) \in \mathbb{Z}^N \notin \mathcal{L}$, we have

$$\|\boldsymbol{v} - \boldsymbol{u}\| < \sqrt{N}2^w \ll \sqrt{\frac{N}{2\pi e}}\mathrm{vol}(\mathcal{L})^{\frac{1}{N}}. \tag{7}$$

Then, heuristically we expect that the vector $\boldsymbol{v}$ in inequalities (6) is the shortest vector in $\mathcal{L}$ and the $\boldsymbol{v}$ in inequalities (7) is the closest vector to $\boldsymbol{u}$ in $\mathcal{L}$. By the discussion in Section 2.3, when N is bounded, vector $\boldsymbol{v}$ can be found efficiently by solving the SVP or CVP with LLL or other related algorithm, and then the value of D can be recovered, which immediately leaks the private key d in deterministic ECDSA or d_0 in EdDSA. To have a complete view about the proposed attacks, Table 1 relates the targets with the relevant problems in lattice.

4 Concrete lattice-based fault attacks on deterministic ECDSA and EdDSA algorithms

In this section, we instantiate the idea of the attacks discussed in Section 3. The key point is to show that equations (4) and (5) can be constructed when concrete targets are selected. Then, the lattice-based approach described in Section 3.2 can be followed to do key recovery. Since most of the attacks presented in this paper are of similar structure in description, to simplify presentation, only two representative attacks are described in this section, while other attacks, with targets shown in Table 1, are gathered in Appendix A.

4.1 Fault attacks with target r during the calculation of s

Suppose the adversary decides to inject a fault against r before using it to calculate s. Then after getting a correct signature for a message m (chosen by the adversary in advance), the adversary manages to get $N-1$ valid faulty signatures with the same message m as input, and r as the target of fault injection.

4.1.1 Attacks on deterministic ECDSA

Step 1: inject fault to r during the calculation of s

During the calculation of s, if injected with a fault, r can be represented as $r_i = r + \varepsilon_i 2^{l_i}$ for $i = 1, ..., N-1$, where ε_i is a random number satisfying $-2^w < \varepsilon_i < 2^w < n$ (by the random fault model) and the known value l_i satisfies $l_i \geq 0$ and $l_i + w \leq f$. The correct signature (r, s_0) and $N-1$ faulty results (r_i, s_i) for the same input message m can be represented as

$$\begin{cases} s_0 = k^{-1}\left(e + rd\right) \bmod n \\ s_i = k^{-1}\left(e + (r + \varepsilon_i 2^{l_i})d\right) \bmod n \text{ (for } i = 1, ..., N-1). \end{cases} \tag{8}$$

Step 2: recover the private key d by solving SVP

After reduction, equation (8) can be transformed as

$$\varepsilon_i = (s_i - s_0)\, 2^{-l_i} d^{-1} k \bmod n. \tag{9}$$

Let $A_i = (s_i - s_0)2^{-l_i} \bmod n$ and $D = d^{-1}k \bmod n$. There must exist $h_i \in \mathbb{Z}$ for $i = 1, ..., N-1$ such that

$$\varepsilon_i = A_i D + h_i n, \tag{10}$$

where D is a fixed value due to the same input message m for all the signature queries.

It is clear that equation (10) is exactly equation (4). Then following the strategy described in Section 3.2, D can be recovered by solving SVP and subsequently the private key d can be recovered by virtue of the equation

$$d = (Ds_0 - r)^{-1} e \bmod n.$$

4.1.2 Attacks on EdDSA

Before we proceed, it should be noted that the existing DFAs against EdDSA [1,5,28,29,30] do not recover the private key d, but rather recover the subkeys d_0 or d_1. This is still a real risk to the security of EdDSA since knowing a partial key d_0 or d_1 suffices to forge signatures [29].

Just like in the case of deterministic ECDSA, if the target r during the calculation of s is chosen, the correct and faulty signatures can be expressed as

$$\begin{cases} s_0 = k + rd_0 \mod n \\ s_i = k + (r + \varepsilon_i 2^{l_i}) d_0 \mod n \, (i = 1, ..., N-1). \end{cases} \tag{11}$$

After reduction, there must exist $h_i \in \mathbb{Z}$ for $i = 1, ..., N-1$ such that equation (11) can be transformed as

$$\varepsilon_i = A_i D + h_i n, \tag{12}$$

where $A_i = (s_i - s_0) 2^{-l_i} \bmod n$, and $D = d_0^{-1} \bmod n$.

Equation (12) is exactly equation (4). Analogously, by applying the general strategy described in Section 3.2, D can be found by solving CVP and subsequently the signing key d_0 can be obtained.

4.2 Fault attacks with target k before the calculation of kG

Suppose the adversary decides to inject a fault to k before using it to calculate kG. Then after getting a correct signature for a message m (chosen by the adversary also), the adversary can manage to get $N-1$ valid faulty signatures with the same message m as input, and k as the target.

4.2.1 Attacks on deterministic ECDSA

Step 1: inject fault to k before the calculation of kG

When k is injected with a fault, it has $k_i = k + \varepsilon_i 2^{l_i}$ for $i = 1, ..., N-1$, where ε_i satisfying $-2^w < \varepsilon_i < 2^w$ is a random number, $l_i \geq 0$ and $l_i + w \leq f$. The correct signature (r_0, s_0) and $N-1$ faulty ones (r_i, s_i) for the same message m can be represented as

$$\begin{cases} k = {s_0}^{-1} (e + r_0 d) \bmod n \\ k + \varepsilon_i 2^{l_i} = {s_i}^{-1} (e + r_i d) \bmod n \, (i = 1, ..., N-1). \end{cases} \tag{13}$$

Step 2: recover the private key d by solving CVP

After reduction, equation (13) can be transformed as

$$\varepsilon_i = \left(s_i{}^{-1}r_i - s_0{}^{-1}r_0\right)2^{-l_i}d - \left(s_0{}^{-1} - s_i{}^{-1}\right)2^{-l_i}e \bmod n. \tag{14}$$

Let $A_i = \left(s_i{}^{-1}r_i - s_0{}^{-1}r_0\right)2^{-l_i} \bmod n$,$B_i = (s_0{}^{-1} - s_i{}^{-1})2^{-l_i}e \bmod n$ and $D = d \bmod n$. Then there must exist $h_i \in \mathbb{Z}$ for $i = 1, ..., N-1$ such that

$$\varepsilon_i = A_iD + h_in - B_i. \tag{15}$$

Equation (15) is exactly equation (5). Analogously, by applying the general strategy described in Section 3.2, D, i.e., the private key d can be obtained in polynomial time in N.

4.2.2 Attacks on EdDSA

Just like in the case of deterministic ECDSA, if the target k before the calculation of kG is chosen, the correct and faulty signatures can be expressed as

$$\begin{cases} s_0 = k + r_0d_0 \bmod n \\ s_i = k + \varepsilon_i 2^{l_i} + r_id_0 \bmod n (i = 1, ..., N-1). \end{cases} \tag{16}$$

After reduction, there must exist $h_i \in \mathbb{Z}$ for $i = 1, ..., N-1$ such that equation (16) can be transformed as

$$\varepsilon_i = A_iD + h_in - B_i, \tag{17}$$

where $A_i = (r_0 - r_i)2^{-l_i} \bmod n$, $D = d_0 \bmod n$ and $B_i = (s_0 - s_i)2^{-l_i} \bmod n$.

Equation (17) is exactly equation (5). Analogously, by applying the strategy described in Section 3.2, d_0 can be obtained in polynomial time in N.

5 Experiment and complexity discussion

The validity of the proposed attacks lies in two aspects, namely, the validity of fault injection and the validity of key recovery. Section 3 presents the conditions and allowed adversarial actions for fault injection, and it is reasonable to believe that suitable faults can be induced during the signature generation process since our adversarial model is not completely new compared to the models in [14,28,29,30]. Thus, we do not conduct concrete experiments to demonstrate the applicability of fault injection. On the other hand, experiments are performed to check the validity of lattice-based key recovery algorithms. This is helpful to understand the relations between the allowed faulty bits(w), the required number of faulty signatures (N), and the success rate (γ) of the presented key recovery.

The experiments are conducted in a computer with 2.4GHz CPU, 8GB memory and Windows7 OS. The BKZ algorithm with block size of 20 implemented in NTL library [35] is employed to solve SVP/CVP. The experimental results for 256-bit deterministic ECDSA(based on NIST P-256) and EdDSA(based on curve25519, i.e., Ed25519) under some specific elliptic curve parameterized, are listed in Table 2 and Table 3 respectively.

Table 2. Success rate in attacking 256-bit deterministic ECDSA ($f = 256$)

target of fault injection	$w = 250$		$w = 245$		$w = 192$		$w = 160$		$w = 128$	
	N	γ	N	γ	N	γ	N	γ	N	γ
r during the calculation of s	80	100%	29	100%	6	96%	4	85%	3	70%
k^{-1} during the calculation of s	80	100%	29	100%	6	96%	4	89%	3	65%
k during the calculation of s	80	100%	29	100%	6	97%	4	87%	3	82%
$e, rd, e + rd$ during the calculation of s	80	100%	27	100%	6	97%	4	87%	3	67%
d during the calculation of s	80	100%	26	100%	6	95%	4	85%	3	67%
k before the calculation of kG	80	74%	30	100%	6	100%	4	100%	3	55%

Table 3. Success rate in attacking 256-bit EdDSA($f = 253$)

target of fault injection	$w = 247$		$w = 245$		$w = 192$		$w = 160$		$w = 128$	
	N	γ	N	γ	N	γ	N	γ	N	γ
r during the calculation of s	80	100%	45	98%	6	97%	4	84%	3	23%
k before the calculation of kG	110	13%	29	100%	6	100%	4	100%	3	12%

Before proceeding to describe the experiment results some points need to be clarified. First, to simplify the experiments, we only conduct key recovery experiments for representative targets (defined in Section 3.1). Similarly, due to the similarity of key recovery for targets $e, rd, e + rd$ during the calculation of s, we just conduct key recovery experiments with the target e.

Then, for each experiment of key recovery, we use a pseudo-random generator to generate the input message m and $N - 1$ groups of w-bit random numbers β_is. For simplicity, the chosen target v is set to be $v \oplus \beta_i$ for $i = 1, \ldots, N - 1$, which is equivalent to $v + \varepsilon_i \bmod n$ (where $l_i = 0$ and $-2^w < \varepsilon_i < 2^w$). For $l_i \neq 0$, the simulations are similar. Then the simulated faulty signatures are used to do key recovery. If the signing key can be recovered finally, the experiment is marked *successful*, otherwise *failed*. A such-designed experiment could fail because the approximate shortest (or closest) vector derived by LLL algorithm could be not the shortest one if the selected N is not big enough, or the constructed lattice basis is not nice due to the oversize w and so on. For simplicity, we record the success rate of the experiments as $\gamma = \frac{\text{number of successful experiments}}{\text{total number of experiments}}$.

Third, for each selected fault target (corresponding to each row of Table 2 and Table 3), we illustrate the validity of attacks in five groups, each of them corresponding to a specific value of parameter (w, N). Note that when n is fixed, the range of w and N can be determined from the relations $w < f - \log\sqrt{2\pi e}$ and $N \gg 1 + \frac{f + \log\sqrt{2\pi e}}{f - w - \log\sqrt{2\pi e}}$ respectively. Hence, when $f = \lceil \log n \rceil = 256$(or $f = 253$ in Ed25519), the tolerant bound of w can be up to 253(or 250) in theory. Then, for each pair (w, N), a number of experiments are conducted to validate the effectiveness of key recovery.

Regarding the experiment number of each case, when $w \leq 245$, we conduct 1000 experiments to derive each success rate γ; when $w = 250$ or 247, only 100 experiments is conducted since our experiment platform cannot afford the significant computational cost of BKZ algorithm. The maximal w in our experiments is considered as 250(or 247), which is slightly less than the tolerable bound(i.e., 253 or 250) in theory. It is hopeful that if some other improved lattice reduction algorithms, such as BKZ 2.0 [12] with some optimum parameters, are utilized in the experiments, the theoretical bounds (i.e., 253 and 250) could be achieved. Moreover, the needed N is approximate to $1 + \frac{f+\log\sqrt{2\pi e}}{f-w-\log\sqrt{2\pi e}}$ in experiments. In addition, the success rate γ is tightly related to the parameters w and N. When w is set to be closed to 245, 30 and 45 valid faulty signatures suffice to recover the key with absolute success rate for deterministic ECDSA and Ed25519 respectively. However, when w is significantly less than the bound, a few valid faulty signatures suffice to recover the key with significant success rate. For example, when $w = 128$, 1 correct signature and 2 valid faulty signatures suffice to recover the key with success rate over 12% in experimental time $2 \sim 3$ms. As a comparison, it is impractical for the existing DFAs [1,5,28,29] to break the deterministic signature when $w \geq 64$, since exponential complexity $O(2^w)$ is required to enumerate the faulty patterns. In addition, w will not be known in practice. Thus a conservative way is to set w as the (practical) maximum tolerable bound such that the key recovery can succeed.

Table 4. Comparison of attack complexity on 256-bit deterministic ECDSA or EdDSA

Scheme / Item	Our attacks	Previous DFAs [1,5,28,29]
tolerable bound of faulty bits (in w)	249	≈ 64
asymptotic time complexity	$O(N^5(N + \log A)\log A)$	$O(2^w)$
time cost in experiments ($w = 128$)	$2 \sim 3$ ms ($N = 3$)	$\approx 2^{128}$ basic operations

To have a more complete view about the computational complexity of the proposed key recovery algorithms, we compare them with the existing attacks in Table 4. In our experiments, the block size of BKZ algorithm is set as 20, and thus the LLL-based reduction with asymptotic complexity $O(N^5(N + \log A)\log A)$ [24] consumes the main time, where A is the maximum length in the original lattice vectors. When N is chosen as a polynomial of (f, w), the computational complexity is thus polynomial in $\log n$ and w, which is obviously less than the exponential complexity required by the existing approaches [1,5,28,29].

As a conclusion, our approach has obvious advantages over the mentioned existing approaches in terms of the tolerance of faulty bits (characterized by w) and time complexity, which also means the proposed attacks are of higher applicability when comparing with those approaches.

6 Countermeasures

In this section, we discuss the effectiveness of some possible countermeasures.

-Randomization. As introduced above, the proposed attacks take advantage of the fact that k is determined by the input message and the private key, and remains unchanged during the process of signature queries. Intuitively the condition can be removed by reintroducing randomness to the derivation of k. This is the exact idea of hedged signature schemes, where the input message, secret key and a nonce are input to generate the per-signature randomness k. The security of hedged signature schemes against fault attacks has recently been proved under some limited models [3]. This strategy can theoretically defeat our attacks but it remains unclear whether it can be used to resist all fault attacks.

-Data integrity protection. Integrity protection is a natural choice for fault attacks resistance. It is a fact that the security of data transmission and storage can be consolidated by adopting error detection (or correction) code in the circuit level. However, limited by the computing power and cost factors, it is usually impossible to adopt strong integrity protection in the smart card like products. Thus the usually implemented Parity check and Cyclic Redundancy Code will leave rooms for fault injection. Namely, though they can be used to resist our attacks to some extent, more considerations are required to validate the real effectiveness of the mechanism. In addition, though the strategy that checking whether the input and output points are on the original elliptic curve can be used to resist the attacks in [1], our attacks are still effective in this case.

-Signature verification before outputting. Note the signature result of the two targeted deterministic algorithms is the form of (r, s). If r is tampered but s remains untainted, verifying the signature before outputting cannot detect the fault. This means the attack selecting k before the calculation of kG as target can survive, but other proposed attacks can be prevented.

-Consistency check of repeated computations. In this strategy, the signature calculation on an input message is repeated for two or more times, and the signature result will be output only when all the computation results are consistent. This can be effective to resist all the proposed attacks since there is no guarantee that the fault induced each time will be the same under the random fault model. But this countermeasure may not be efficient, since in this case two scalar multiplications have to be computed, which is unaffordable for some devices (such as IoT devices) whose computing power is very limited.

-Infective computation. This strategy is graceful in that the adversary in this case cannot distinguish whether the faulty signature is valid or not, thus the key recovery can be defeated. We propose two infective countermeasures to resist the proposed attacks.

(i) For EdDSA, the *last modulo-*2^t *additions* in the hash function $H(d_1, m)$ generating k are calculated twice to obtain two identical nonces k_1 and k_2, and *the last modulo-*2^t *additions* in the hash function $H(R, P, m)$ generating r are calculated twice to obtain two identical r_1 and r_2; moreover, an infective factor β is introduced, which has the same bit length with k, and is regenerated per signature. Then compute $s = (1 + \beta)(k_1 + d_0 r_1) - \beta(k_2 + d_0 r_2) \bmod n$.

(ii)For deterministic ECDSA, the *last modulo-*2^t *additions* in the hash function $F(d, e)$ generating k are calculated twice to obtain two identical nonces k_1 and k_2. The *last modulo-*2^t *additions* in the hash function $H(m)$ generating e are calculated twice to obtain two identical e_1 and e_2. The *reduction* $r = x_1$

mod n generating r is calculated twice to obtain two identical r_1 and r_2. The private key d defined as d_1 and d_2 is invoked twice during the calculation of s, respectively. Hence, $s = (1+\beta){k_1}^{-1}(e_1 + d_1 r_1) - \beta {k_2}^{-1}(e_2 + d_2 r_2) \bmod n$.

7 Conclusion

We present a new fault analysis method to deterministic ECDSA and EdDSA algorithms. The fault injection model is a random fault model as in [14]. In the new model, the resulted intermediate of fault injection can be characterized as an addition of the original intermediate with a random value of left-shifted l bits. The range of the random value is determined by and close to the size of the signing key. This makes the method much more practical than the existing pattern enumerating approaches [1,5,28,29] in terms of tolerance of faulty bits.

The advantage is guaranteed by the lattice-based key recovery method. By noticing the algebraic structures of the deterministic algorithms, we show that, when providing with some valid faulty signatures and an associated correct signature of the same input message, some instances of approximate lattice problems can be constructed to recover the signing key. Moreover, the lattice-based approach supports much more alternative targets of fault injection than the existing approaches, which further improves the applicability of the approach.

Experiments are performed to validate the effectiveness of the key recovery method. It is demonstrated that, for 256-bit deterministic ECDSA and EdDSA, the signing key can be recovered efficiently with high probability even if the intermediates are affected by 250 and 247 faulty bits respectively. This is, however, impractical for the existing faulty pattern enumerating approaches to achieve the same objective.

Further Work. For signature schemes with different algebraic structure in generating s, such as $s = (1+d)^{-1}(k - rd) \mod n$ in SM2 signature generation, it is worth studying whether there are more fault injection targets in them. Moreover, if the nonce or the generated per-signature random number is misused in the hedged signatures, the effectiveness of our approaches deserves further analysis.

References

1. C. Ambrose, J. W. Bos, B. Fay, M. Joye, M. Lochter, and B. Murray. Differential attacks on deterministic signatures. In *Cryptographers Track at the RSA Conference*, pages 339–353. Springer, 2018.
2. D. F. Aranha, F. R. Novaes, A. Takahashi, M. Tibouchi, and Y. Yarom. Ladderleak: Breaking ECDSA with less than one bit of nonce leakage. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 225–242, 2020.
3. D. F. Aranha, C. Orlandi, A. Takahashi, and G. Zaverucha. Security of Hedged Fiat–Shamir Signatures under Fault Attacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 644–674. Springer, 2020.

4. L. Babai. On Lovászlattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.
5. A. Barenghi and P. Gerardo. A note on fault attacks against deterministic signature schemes. In *International Workshop on Security*, pages 182–192, 2016.
6. N. Benger, J. Van de Pol, N. P. Smart, and Y. Yarom. ooh aah... just a little bit: A small amount of side channel can go a long way. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 75–92. Springer, 2014.
7. I. Biehl, B. Meyer, and V. Müller. Differential fault attacks on elliptic curve cryptosystems. In *Annual International Cryptology Conference*, pages 131–146. Springer, 2000.
8. D. Bleichenbacher. On the generation of one-time keys in DL signature schemes. In *Presentation at IEEE P1363 working group meeting*, page 81, 2000.
9. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In *International conference on the theory and applications of cryptographic techniques*, pages 37–51. Springer, 1997.
10. L. G. Bruinderink and P. Pessl. Differential fault attacks on deterministic lattice signatures. In *IACR Transactions on Cryptographic Hardware and Embedded Systems (2018)*, pages 21–43, 2018.
11. W. Cao, J. Feng, H. Chen, S. Zhu, W. Wu, X. Han, and X. Zheng. Two lattice-based differential fault attacks against ECDSA with wNAF algorithm. In *International Conference on Information Security and Cryptology*, pages 297–313, 2015.
12. Y. Chen and P. Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In D. H. Lee and X. Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, 2011.
13. E. De Mulder, M. Hutter, M. E. Marson, and P. Pearson. Using Bleichenbachers solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA: extended version. *Journal of cryptographic engineering*, 4(1):33–45, 2014.
14. M. Fischlin and F. Günther. Modeling memory faults in signature and authenticated encryption schemes. In *Cryptographers Track at the RSA Conference*, pages 56–84. Springer, 2020.
15. 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*, pages 1626–1638, 2016.
16. N. A. Howgrave-Graham and N. P. Smart. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, 23(3):283–290, 2001.
17. D. Karaklajić, J. M. Schmidt, and I. Verbauwhede. Hardware designer's guide to fault attacks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(12):2295–2306, 2013.
18. A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
19. M. Liu and P. Q. Nguyen. Solving BDD by Enumeration: An Update. In *Topics in Cryptology – CT-RSA 2013*, pages 293–309, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
20. D. Micciancio and S. Goldwasser. *Complexity of lattice problems: a cryptographic perspective*, volume 671. Springer, 2002.
21. D. Naccache, P. Q. Nguyên, M. Tunstall, and C. Whelan. Experimenting with Faults, Lattices and the DSA. In *International Workshop on Public Key Cryptography*, pages 16–28. Springer, 2005.
22. P. Q. Nguyen. *Hermite's Constant and Lattice Algorithms*, pages 19–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

23. P. Q. Nguyen and I. E. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Designs, codes and cryptography*, 30(2):201–217, 2003.
24. P. Q. Nguyên and D. Stehlé. Floating-point LLL revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 215–233. Springer, 2005.
25. P. Q. Nguyen and J. Stern. Lattice reduction in cryptology: An update. *Lecture Notes in Computer Science*, 1838:85–112, 2000.
26. P. Q. Nguyen and M. Tibouchi. Lattice-based fault attacks on signatures. In *Fault Analysis in Cryptography*, pages 201–220. Springer, 2012.
27. P. Q. Nguyen and M. Tibouchi. Lattice-based fault attacks on signatures. In *Fault Analysis in Cryptography*, pages 201–220. Springer, 2012.
28. D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler. Attacking deterministic signature schemes using fault attacks. In *IEEE European Symposium on Security and Privacy (Euro S&P)*, pages 338–352. IEEE, 2018.
29. Y. Romailler and S. Pelissier. Practical Fault Attack against the Ed25519 and EdDSA Signature Schemes. In *Workshop on Fault Diagnosis and Tolerance in Cryptography(FDTC)*, pages 17–24, 2017.
30. N. Samwel and L. Batina. Practical Fault Injection on Deterministic Signatures: The Case of EdDSA. In *International Conference on Cryptology in Africa*, pages 306–321, 2018.
31. J. M. Schmidt and M. Medwed. A fault attack on ECDSA. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 93–99. IEEE, 2009.
32. C. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2):201 – 224, 1987.
33. C.-P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994.
34. C.-P. Schnorr and H. H. Hörner. Attacking the chor-rivest cryptosystem by improved lattice reduction. In *Advances in Cryptology Eurocrypt 1995*, pages 1–12. Springer, 1995.
35. V. Shoup. Number Theory C++ Library (NTL) version 9.6.4. http://www.shoup.net/ntl/, 2016.
36. A. Takahashi, M. Tibouchi, and M. Abe. New Bleichenbacher records: Fault attacks on qDSA signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 331–371, 2018.

A Appendix

This appendix will introduce the attacks with the remaining targets listed in Table 1 to deterministic ECDSA and EdDSA, including the attacks with targets k, k^{-1}, e, rd, $e+rd$ and d during the calculation of s and the attacks taking the hash functions generating k, e and r as fault targets.

A.1 Fault attacks with target k during the calculation of s to deterministic ECDSA

Suppose the adversary decides to inject fault to k before using it during the calculation of s. Then after getting a correct signature for a message m (chosen by the adversary in advance), the adversary can try to get $N-1$ valid faulty signatures with the same message m as input, and k as the target.

Step 1: inject fault to k during the calculation of s

When k is injected with a fault, it has $k_i = k + \varepsilon_i 2^{l_i}$ for $i = 1, ..., N-1$, where ε_i satisfying $-2^w < \varepsilon_i < 2^w < n$ is a random number and l_is are the given values satisfying $l_i \geq 0$ and $l_i + w \leq f$. The correct signature (r, s_0) and $N-1$ faulty ones (r, s_i) for the same input message m can be represented as

$$\begin{cases} k = {s_0}^{-1}(e + rd) \bmod n \\ k + \varepsilon_i 2^{l_i} = {s_i}^{-1}(e + rd) \bmod n (i = 1, ..., N-1) \end{cases}. \tag{18}$$

Step 2: recover the private key d by solving SVP

After reduction, equation (18) can be transformed as

$$\varepsilon_i = \left({s_i}^{-1} - {s_0}^{-1}\right) 2^{-l_i} (e + rd) \bmod n. \tag{19}$$

Let $A_i = ({s_i}^{-1} - {s_0}^{-1}) 2^{-l_i} \bmod n$, $D = e + rd \bmod n$. There must exist $h_i \in \mathbb{Z}$ for $i = 1, ..., N-1$ such that

$$\varepsilon_i = A_i D + h_i n, \tag{20}$$

where D is a fixed value due to the same input message m for all the signature queries.

Equation (20) is exactly equation (4). Then following the general strategy described in Section 3.2, D can be found by solving SVP and subsequently the private key d can be recovered by virtue of the equation

$$d = r^{-1}(D - e) \bmod n.$$

A.2 Fault attacks with target k^{-1} during the calculation of s to deterministic ECDSA

Suppose the adversary decides to inject fault to k^{-1} (after being generated by modular inversion of k) before using it during the calculation of s. Then after getting a correct signature for a message m, the adversary can try to get $N-1$ valid faulty signatures with the same message m as input, and k^{-1} as the target.

Step 1: inject fault to $k^{-1} \bmod n$ during the calculation of s

When $k^{-1} \bmod n$ derived by k is injected with a fault, it has $k_i^{-1} = k^{-1} + \varepsilon_i 2^{l_i}$ for $i = 1, ..., N-1$, where ε_i satisfying $-2^w < \varepsilon_i < 2^w$ is a random number and l_is are the given values satisfying $l_i \geq 0$ and $l_i + w \leq f$. The correct signature (r, s_0) and $N-1$ faulty ones (r, s_i) for the same input message m can be represented as

$$\begin{cases} s_0 = k^{-1}(e + rd) \bmod n \\ s_i = \left(k^{-1} + \varepsilon_i 2^{l_i}\right)(e + rd) \bmod n (i = 1, ..., N-1). \end{cases} \tag{21}$$

Step 2: recover the private key d by solving SVP

After reduction, equation (21) can be transformed as

$$\varepsilon_i = (e + rd)^{-1} (s_i - s_0) 2^{-l_i} \bmod n. \tag{22}$$

Let $A_i = (s_i - s_0)2^{-l_i} \bmod n$, $D = (e + rd)^{-1} \bmod n$. There must exist $h_i \in \mathbb{Z}$ for $i = 1, ..., N-1$ such that

$$\varepsilon_i = A_i D + h_i n, \tag{23}$$

where D is a fixed value due to the same input message m for all the signature queries.

Equation (23) is exactly equation (4). Then following the general strategy described in Section 3.2, D can be found by solving SVP and subsequently the private key d can be recovered by virtue of the equation

$$d = r^{-1}(D^{-1} - e) \bmod n.$$

A.3 Fault attacks with target d during the calculation of s to deterministic ECDSA

Suppose the adversary decides to inject fault to d before using it during the calculation of s. Then after getting a correct signature for a message m, the adversary can try to get $N - 1$ valid faulty signatures with the same message m as input, and d as the target.

Step 1: inject fault to d during the calculation of s

When d is injected with a fault, it has $d_i = d + \varepsilon_i 2^{l_i}$ for $i = 1, ..., N-1$, where ε_i satisfying $-2^w < \varepsilon_i < 2^w$ is a random number and l_is are the given values satisfying $l_i \geq 0$ and $l_i + w \leq f$. The correct signature (r, s_0) and $N-1$ faulty ones (r, s_i) for the same input message m can be represented as

$$\begin{cases} s_0 = k^{-1}\left(e + rd\right) \bmod n \\ s_i = k^{-1}\left(e + r(d + \varepsilon_i 2^{l_i})\right) \bmod n (i = 1, ..., N-1). \end{cases} \tag{24}$$

Step 2: recover the private key d by solving SVP

After reduction, equation (24) can be transformed as

$$\varepsilon_i = \left(s_i - s_0\right) 2^{-l_i} r^{-1} k \bmod n. \tag{25}$$

Let $A_i = \left(s_i - s_0\right) r^{-1} 2^{-l_i} \bmod n$, $D = k \bmod n$, there must exist $h_i \in \mathbb{Z}$ for $i = 1, ..., N-1$ such that

$$\varepsilon_i = A_i D + h_i n, \tag{26}$$

where D is a fixed value due to the same input message m for all the signature queries.

Equation (26) is exactly equation (4). Then following the general strategy described in Section 3.2, D can be found by solving SVP and subsequently the private key d can be recovered by virtue of the equation

$$d = r^{-1}\left(D s_0 - e\right) \bmod n.$$

A.4 Fault attacks with targets e, rd and $e + rd$ during the calculation of s to deterministic ECDSA

If the targets e, rd and $e + rd$ targets are disturbed by fault injection, a same model of key recovery can be constructed. Therefore, for simplicity, we define mv as any one of the three targets, that is, mv could be e, rd or $e+rd$. Suppose the adversary decides to inject fault to mv before using it during the calculation of s. Then after getting a correct signature for a message m, the adversary can try to get $N - 1$ valid faulty signatures with the same message m as input, and mv as the target.

Step 1: inject fault to mv during the calculation of s

When mv is injected with a fault, it has $mv_i = mv + \varepsilon_i 2^{l_i}$ for $i = 1, ..., N-1$, where ε_i satisfying $-2^w < \varepsilon_i < 2^w$ is a random number and l_is are the given values satisfying $l_i \geq 0$ and $l_i + w \leq f$. The correct signature (r, s_0) and $N - 1$ faulty ones (r, s_i) for the same input message m can be represented as

$$\begin{cases} s_0 = k^{-1}\left(e + rd\right) \bmod n \\ s_i = k^{-1}\left(e + rd + \varepsilon_i 2^{l_i}\right) \bmod n (i = 1, ..., N-1). \end{cases} \tag{27}$$

Step 2: recover the private key d by solving SVP

After reduction, equation (27) can be transformed as

$$\varepsilon_i = \left(s_i - s_0\right) 2^{-l_i} k \bmod n. \tag{28}$$

Let $A_i = \left(s_i - s_0\right) 2^{-l_i} \bmod n$, $D = k \bmod n$, there must exist $h_i \in \mathbb{Z}$ for $i = 1, ..., N-1$ such that

$$\varepsilon_i = A_i D + h_i n, \tag{29}$$

where D is a fixed value due to the same input message m for all the signature queries.

Equation (29) is exactly equation (4). Then following the general strategy described in Section 3.2, D can be found by solving SVP. Naturally, as mentioned above, the private key d can be recovered by virtue of D.

A.5 Fault attacks with the targets during the calculation of k to deterministic ECDSA and EdDSA

As described in Section 4.2, if injecting a fault into "k before the calculation of kG" to obtain some valid k_is satisfying $k_i = k + \varepsilon_i 2^{l_i} (-2^w < \varepsilon_i < 2^w, w < f - \log \sqrt{2\pi e})$, then equation (5) can be constructed to recover the private key in deterministic ECDSA or EdDSA. Then, besides the target "k before the calculation of kG", we found some other fault targets during the calculation of k also can generate valid faulty k_is, including "registers before outputting hash value", "last modular additions before outputting hash value" and "hash value during the reduction of k".

The following section will introduce the three targets and the fault injection models whose *final purpose* is to generate some valid faulty signatures satisfying

$k_i = k + \varepsilon_i 2^{l_i}(-2^w < \varepsilon_i < 2^w)$, and $w < f - \log\sqrt{2\pi e}$. For simplicity, we just consider the case when $l_i = 0$ for $i = 1, ..., N-1$(i.e., the w least significant bits of k are disturbed) and the hash function is SHA-2, to which the other cases are similar.

A.5.1 Hash Function Generating *k*

Although different methods are employed for generating k in deterministic ECDSA and EdDSA(for example, HMAC_DRBG_SHA256 $F(d, e)$ is utilized in deterministic ECDSA and hash algorithm SHA512 $H(m, d_1)$ is utilized in EdDSA by default), they all belong to the family of SHA2 and have the similar modular additions before outputting the hash value to generate k. As shown in Figures 2 and 3, after all the L-round calculations of Hash function, the modular additions(mod2^t, t is bit length of register) of the registers $(a_{L-1}, ..., , g_{L-1}) \in [0, 2^t)$ and $(a_0, ..., , h_0) \in [0, 2^t)$ are calculated and the results are assigned to the registers $(a_L, ..., , h_L) \in [0, 2^t)$ as the hash values. Hence, once a fault is injected into these registers, the calculation of addition or the hash values during the following reduction, k will be affected by the faulty values.

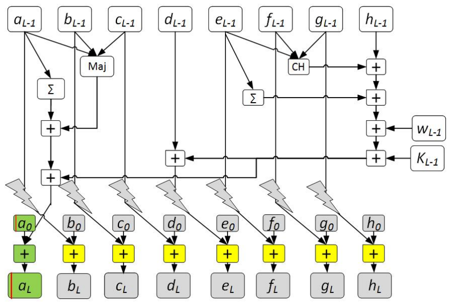

Fig. 2. Fault targets in the hash function of determinsic ECDSA

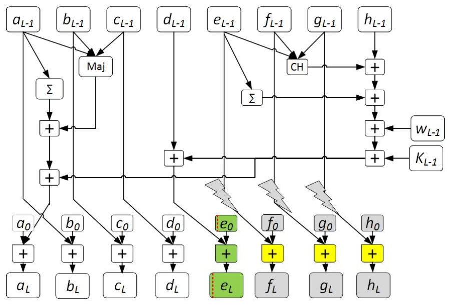

Fig. 3. Fault targets in the hash function of EdDSA

Table 5 gives an overview of all the fault targets before outputting the hash values and during the reduction of k.

A.5.2 Fault Attacks with Target: Registers before Outputting Hash Value

Attacks on deterministic ECDSA

For the HMAC_DRBG_SHA256 during signature generation of deterministic ECDSA, the final output registers $(a_L, ..., , h_L)$ can be reduced into a big number $k = T2^{8t} + a_L 2^{7t} + ... + g_L 2^t + h_L \bmod n$, where T is the concatenation of the previous u-times HMAC values(i.e., $T = HMAC_0||HMAC_1||...||HMAC_{u-1}$) which is equal to 0 in 256-bit deterministic ECDSA, and t is the bit length of register.

Table 5. The targets of fault injection during the calculation of k.

target	algorithm	concrete location of fault injection
registers before outputting hash values	deterministic ECDSA	(partial bits of a_0, a_L), $(b_0, ..., h_0)$ $(b_L, ..., h_L)$, $(a_{L-1}, ..., g_{L-1})$
	EdDSA	(partial bits of e_0, e_L), (f_0, g_0, h_0) (f_L, g_L, h_L), $(e_{L-1}, f_{L-1}, g_{L-1})$
modular additions before outputting hash values	deterministic ECDSA	all the modulo-2^t additions
	EdDSA	modulo-2^t additions in the right half
hash value during the reduction of k	deterministic ECDSA	the value of $F(d, e)$
	EdDSA	the value of $H(d_1, m)$

As shown in Figure 2, assuming that all or arbitrary one of the registers $(a_0, ..., h_0)$, $(a_{L-1}, ..., g_{L-1})$ before the last additions and $(a_L, ..., h_L)$ before outputting the hash value are injected with a fault, the consequent k can be represented as $k_i = T2^{8t} + (a_L 2^{7t} + b_L 2^{6t} + ... + g_L 2^t + h_L + \varepsilon_i) \bmod n$ for $i = 1, ..., N-1$, with a random faulty value ε_i satisfying $-2^w < \varepsilon_i < 2^w$ and $w < f - \log\sqrt{2\pi e} \leq 8t - \log\sqrt{2\pi e}$ ($8t = f$ for 256-bit deterministic ECDSA). That is, k_i which is derived from the faulty hash value and is to participate in the next calculation of kG, is equals to $k + \varepsilon_i \bmod n$.

Similar to the key recovery with target "k before the calculation of kG", equation (5) can be constructed. Then following the general strategy described in Section 3.2, the private key d can be recovered by solving the instance of CVP in lattice.

Note that in 256-bit deterministic ECDSA, to make sure $w < 8t - \log\sqrt{2\pi e}(8t = f)$, the register h_{L-1} can not be viewed as target, and as listed in Table 5, at least $\log\sqrt{2\pi e}$ most significant bits of the registers a_0 and a_L can not be disturbed when a fault is injected into them. Except this, all the fault injection against the other registers are arbitrary and uncontrolled.

Attacks on EdDSA

In the hash algorithm SHA512 $H(m, d_1)$ of EdDSA, the final output 512-bit registers $(a_L, ..., h_L)$ as the hash value must be reduced into the nonce $k = a_L 2^{7t} + ... + g_L 2^t + h_L \mod n$, where t is the bit length of register and is equals to 64 in SHA512. For 256-bit EdDSA, e.g., Ed25519, the modular reduction will reduce the 512-bit hash value into a 253-bit nonce k. Hence, in order to obtain valid faulty signatures, fault injection here will take the four registers in the right half as the targets.

As shown in Figure 3, when all or arbitrary one of the registers $(e_0, ..., h_0)$, $(e_{L-1}, ..., g_{L-1})$ before the last additions and $(e_L, ..., h_L)$ before outputting hash value is injected with a fault, the consequent k can be represented as $k_i = a_L 2^{7t} + ... + d_L 2^{4t} + (e_L 2^{3t} + ... + h_L + \varepsilon_i) \mod n$ for $i = 1, ..., N-1$, with a random faulty value ε_i satisfying $-2^w < \varepsilon_i < 2^w$ and $w < f - \log\sqrt{2\pi e} \leq 4t - \log\sqrt{2\pi e}$ ($4t \approx f$ for 256-bit EdDSA). That is, $k_i = k + \varepsilon_i \mod n$. Similar to the key recovery with target "k before the calculation of kG", equation (5) can be constructed.

Then according to the general strategy described in Section 3.2, the private key d can be recovered by solving the instance of CVP in lattice.

Note that in 256-bit EdDSA, to make sure $w < 4t - \log \sqrt{2\pi e}(4t \approx f)$, only the right half of the registers are viewed as targets, and as listed in Table 5, at least $\lceil \log \sqrt{2\pi e} \rceil$ most significant bits of the registers e_0 and e_l can not be disturbed when a fault is injected into them. Except this, the fault injection to the remaining three registers is arbitrary and uncontrolled. In addition, if the registers in the left half are disturbed, then $k_i = k + \varepsilon_i 2^{4t} \mod n$. Similarly, we also can construct a similar CVP in lattice to recover the private key.

A.5.3 Fault Attacks with Target: Last Modular Additions before Outputting Hash Value

As described in Sections A.5.1 and A.5.2, if the last modulo-2^t additions as targets of fault injection are injected by a fault to lead to the final hash values $a_L, ..., h_L$ faulty, then the nonce k reduced by the hash value has w disturbed bits, by which equation (5) can be constructed to recover the private key d.

For 256-bit deterministic ECDSA, as shown in Figure 2, all or arbitrary one of the last modulo-2^t additions can be injected with a fault. Moreover, it is noted that fault injection against the left first addition must make at least $\log \sqrt{2\pi e}$ high significant bits of a_L undisturbed.

Similarly, for 256-bit EdDSA, as shown in Figure 3, all or arbitrary one of the last modulo-2^t additions in the right half can be injected with a fault. Moreover, fault injection against the first addition in the right half must make at least $\lceil \log \sqrt{2\pi e} \rceil$ most significant bits of e_L undisturbed. In addition, similarly, if the additions in the left half are disturbed, we also can construct a similar CVP in lattice.

A.5.4 Fault Attacks with Target: Hash Value during the Reduction of *k*

After calculating the last modular additions in the hash function, the final registers are combined into a big number $E(E = F(d, e)$ in deterministic ECDSA or $E = H(d_1, m)$ in EdDSA), and E is needed to be reduced into nonce k, That is, $k = E \bmod n$. Assuming that a fault is injected into E during the reduction, the reduction $k = E \bmod n$ is changed into $k_i = E + \varepsilon_i 2^{l_i} \mod n$. Hence, as long as ε_i satisfying $-2^w < \varepsilon_i < 2^w$ is a random number and $w < f - \log \sqrt{2\pi e}$, equation (5) can be constructed. Thereby, the private key can be recovered by solving an instance of CVP in lattice.

To sum up, the three targets during the calculation of k for fault attacks are equivalent to the *representative* target "k before the calculation of kG", and thereby equation (5) can be constructed to recover the private key d in deterministic ECDSA and d_0 in EdDSA.

A.6 Fault attacks with targets during the calculation of e to deterministic ECDSA

As introduced in Appendix A.4, if injecting a fault into e during the calculation of s to obtain some valid e_is satisfying $e_i = e + \varepsilon_i 2^{l_i}$ ($-2^w < \varepsilon_i < 2^w$, $w < f - \log\sqrt{2\pi e}$ and $l_i + w \leq f$), then equation (4) can be constructed to recover the private key in deterministic ECDSA.

Similarly, besides directly injecting fault into the target "e during the calculation of s", there still exist two other fault targets during the calculation of e which can generate some valid faulty e_is for key recovery, including "registers before outputting the hash values $H(m)$" and "last modular additions before outputting the hash values $H(m)$". The models of fault injection with these two targets are similar to the ones introduced in Appendix A.5.2 and A.5.3, and thereby equation (4) which is similar to the one with target "e during the calculation of s", can be constructed to recover the private key in deterministic ECDSA.

A.7 Fault attacks with targets during the calculation of r to EdDSA

As introduced in Section 4.1.2, if injecting a fault into r during the calculation of s to obtain some valid r_is satisfying $r_i = r + \varepsilon_i 2^{l_i}$ ($-2^w < \varepsilon_i < 2^w$, $w < f - \log\sqrt{2\pi e}$ and $l_i + w \leq f$), equation (4) can be constructed to recover the private key in EdDSA.

Similarly, besides directly injecting fault into the target "r during the calculation of s", there still exist another two fault targets during the calculation of r which can generate some valid faulty r_is for key recovery, including "registers before outputting hash value $H(R, P, m)$", "last modular additions before outputting hash value $H(R, P, m)$" and "hash value $H(R, P, m)$ during the reduction of r". The models of fault injection with these three targets are similar to the ones in Appendix A.5.2, A.5.3 and A.5.4, and thereby equation (4) which is similar to the one of target "r during the calculation of s", can be constructed to recover the private key in EdDSA.