A Differential Fault Attack against Deterministic Falcon Signatures

Sven Bauer and Fabrizio De Santis

Siemens AG, Technology, Munich, Germany
{svenbauer,fabrizio.desantis}@siemens.com

Abstract. We describe a fault attack against the deterministic variant of the Falcon signature scheme. It is the first fault attack that exploits specific properties of deterministic Falcon. The attack works under a very liberal and realistic single fault random model. The main idea is to inject a fault into the pseudo-random generator of the pre-image trapdoor sampler, generate different signatures for the same input, find reasonably short lattice vectors this way, and finally use lattice reduction techniques to obtain the private key. We investigate the relationship between fault location, the number of faults, computational effort for a possibly remaining exhaustive search step and success probability.

Keywords: Fault attack · Post-quantum cryptography · Digital signature schemes · Lattice-based cryptography · Falcon

1 Introduction

In July 2022, the U.S. Department of Commerce's National Institute for Standard and Technology (NIST) announced the first four winners of the Post-Quantum Cryptography (PQC) standardization project after a six-year long competition and three rounds of evaluation [NIS22]. The goal of the NIST PQC standardization project is to standardize public-key post-quantum algorithms for public-key encryption (PKE), key encapsulation mechanisms (KEM), and digital signatures, as currently specified in the NIST SP800-56A/B and FIPS 186 documents. The first drafts are expected to be released in 2023. NIST has selected only CRYSTALS-Kyber [SAB$^+$22] for PKE/KEM, while three algorithms CRYSTALS-Dilithium [LDK$^+$22], Falcon [PFH$^+$22] and SPHINCS+ [HBD$^+$22] were selected for digital signatures. The security of CRYSTALS-Kyber, CRYSTALS-Dilithium, and Falcon is based on hardness problems on structured lattices, while the security of SPHINCS+ is based on the security of classical hash functions, e.g., SHA-2, SHA-3. In general, cryptographic algorithms mainly based on the hardness of structured lattice problems offer comparatively small keys and ciphertexts as well as high computational performance.

A provable approach to design post-quantum digital signature schemes was first described by Gentry, Peikert, and Vaikuntanathan [GPV07] and it is based on the hash-and-sign paradigm. This approach is referred to as the GPV framework. Falcon instantiates the GPV framework [GPV07] over NTRU lattices to achieve an efficient post-quantum digital signature scheme with very short signature size. In particular, NIST recommends Falcon for applications that require smaller signatures than CRYSTALS-Dilithium. The current version of Falcon [PFH$^+$22] submitted to the NIST standardization project is non-deterministic, i.e., signing the same message twice results in different signatures with a very high probability. As pointed out in [GPV07], any instance of the GPV framework must never output two different signatures for the same message hash. Falcon gets around this issue by randomizing the hash of the message that is to be signed. So signing the same message more than once will almost certainly lead to different message hash values and hence different signatures. However, a deterministic version of FALCON is desirable for some use-cases, cf. [LPa17], and may also be considered for future standardization. Deterministic FALCON de-randomizes both the message hashing and the trapdoor sampler to support a fully deterministic signing mode as specified in [LPa17]. However, it is very well known that deterministic digital signature schemes are prone to Differential Fault Analysis (DFA) (see, for example, [PSS+17, BP18]). Generally speaking, a DFA attacker can alter the signature generation process by physical means, e.g., by injecting voltage glitches, laser, or electro-magnetic radiation. In this way, an adversary is able to collect pairs of correct and faulty outputs, i.e., digital signatures, and recover the secret private key used for signature generation.

Our DFA on deterministic FALCON signatures targets the pseudo-random generator in the pre-image trapdoor sampler, essentially reintroducing randomness, thus forcing the trapdoor sampler to produce different signatures for the same input, hence violating a central security assumption of FALCON.

1.1 Previous work

Fault Attacks against Deterministic ECDSA/EdDSA Signatures A first DFA against deterministic ECDSA and EdDSA has been proposed in [BP16]. The paper presents two attacks exploiting faults injected during the calculation of the scalar multiplication and nonce during the digital signature generation routine. The adversary can recover the secret key by solving linear equations obtained from correct and faulty pairs of signatures. The attacks have subsequently been improved in [RP17, ABF+18, PSS+18, SB18], where different fault injection targets are exploited and an evaluation of real hardware platforms is performed. A further fault attack against deterministic ECDSA/EdDSA using lattice-based techniques to recover the secret key has recently been proposed in [CSC+22].

Fault Attacks against Lattice-based Signatures The work of [BBK16, EFGT16] investigated fault attacks on non-deterministic lattice-based signature schemes, such as BLISS [DDLL13], GLP [GLP12], PASSSign [LZA18], and ring-TESLA [ABB+16]. In particular, the work of [EFGT16] presented a fault attack exploiting early loop abort faults against discrete Gaussian sampling in the secret trapdoor lattice of the GPV-based hash-and-sign signature scheme of [DLP14]. The attack requires $m + 2$ faulty signatures using loop aborts after m iterations for a full key recovery. The possibility of lattice-based deterministic signatures was already hinted in [LDK+17, BAA+17]. However, the applicability of differential fault attacks against deterministic versions of CRYSTALS-Dilithium and qTESLA were investigated in a nonce-reuse scenario using a single random faults model [BP18].

Fault Attacks against Falcon Signatures In [MHS+19], the authors propose two types of fault attack against FALCON. First, they adapt an attack from [EFGT16] to FALCON. In this attack, an injected fault causes the trapdoor sampler to stop prematurely. The component s_2 of the resulting faulty signatures is then just a vector in a low-dimensional lattice generated by the secret key component F. Hence, lattice reduction can be applied on a small number of faulty signatures to find F. From F, the other secret key components can be found using the public key and the NTRU equation that defines the relationship between the secret key components. The second attack assumes the attacker can set large parts of the output of the trapdoor sampler to zero, resulting in the same type of faulty signatures as the first attack. The processing of the faulty signatures to obtain the private key is hence the same.

1.2 Contributions

To the best of our knowledge, we describe the first fault attack on deterministic Falcon that exploits specific algorithmic properties. The attack works under a very generic and realistic fault model using random faults injected during the execution of the pre-image trapdoor sampler. We investigate the relationship between fault location, the number of faults, computational effort for a possibly remaining exhaustive search step and success probability. We provide simulations results both on a PC and ARM Cortex-M4 processor to validate our claims.

1.3 Outline

In Section 2, we define the notation and provide some background information on lattice-based cryptography and the Falcon digital signature scheme. In Section 3, we give a brief algorithmic description of Falcon. The proposed fault attacks are described in Section 4. A practical evaluation of the attack is provided in Section 5. Countermeasures to protect against the proposed fault attack are described in Section 6. Conclusion and outlook are provided in Section 7.

2 Preliminaries

In this section, we briefly introduce the necessary background on lattice-based cryptography and the Falcon digital signature scheme.

Notation We denote row vectors by bold lowercase letters and matrices by bold uppercase letters. The inner product of two vectors $\mathbf{x}, \mathbf{y}$ is denoted by $\langle \mathbf{x}, \mathbf{y} \rangle$ and the Euclidean norm of a vector $\mathbf{x}$ is denoted by $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$. Let $\phi(x) = x^n + 1$ for $n = 2^\kappa$ a power of two and let $\mathcal{R} = \mathbb{Z}[x]/(\phi)$. The ring $\mathcal{R}$ is a truncated polynomial ring that inherits its ring structure from $\mathbb{Z}[x]$. The elements $f \in \mathcal{R}$ are represented as polynomials, e.g., $f(x) = \sum_{i=0}^{n-1} f_i x^i$, or vectors, e.g., $\mathbf{f} = (f_0, \ldots, f_{n-1})$. Note that $xf(x)$ corresponds to a negacyclic rotation of the corresponding coefficient vector, i.e. viewed as a vector $xf(x)$ is $(-f_{n-1}, f_0, f_1, \ldots, f_{n-2})$.

Any polynomial $g \in \mathcal{R}$ induces a linear map $M_g : \mathcal{R} \to \mathcal{R}$ given by polynomial multiplication, $M_g(f) = fg$. Hence, we can also view polynomials as linear maps or matrices.

For any polynomial f, we denote its Fourier transform by $\hat{f}$ and by $\hat{\mathbf{f}}$ the corresponding coefficient vector. Let $q \in \mathbb{N}^\star$. We denote the lattice generated by the basis $\mathbf{B} = (\mathbf{b}_0, ..., \mathbf{b}_{m-1}) \in \mathbb{Z}^{m \times n}$ by $\Lambda(\mathbf{B}) = \left\{ \sum_{i=0}^{m-1} x_i \mathbf{b_i}, x_i \in \mathbb{Z} \right\}$ and the corresponding orthogonal lattice given by $\Lambda^\perp(\mathbf{B}) = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{x}\mathbf{B}^* = 0\}$.

The q-ary lattice is denoted by $\Lambda_q(\mathbf{B}) = \{\mathbf{y} \in \mathbb{Z}^n : \mathbf{y} = \mathbf{x}\mathbf{B} \bmod q, \mathbf{x} \in \mathbb{Z}^n\}$ and the corresponding orthogonal q-ary lattice is denoted by $\Lambda^\perp(\mathbf{B})_q = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{x}\mathbf{B}^* = 0 \bmod q\}$. Let $\sigma \in \mathbb{R}$ with $\sigma > 0$. For any $\mathbf{c} \in \mathbb{R}^n$, the Gaussian function on $\mathbb{R}^n$ with center at $\mathbf{c}$ and standard deviation σ is denoted by $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = \exp(-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2})$. The discrete Gaussian distribution over Λ of center $\mathbf{c}$ and standard deviation σ is denoted by $D_{\Lambda,\sigma,\mathbf{c}}(\mathbf{z}) = \rho_{\sigma,\mathbf{c}}(\mathbf{z}) / (\sum_{\mathbf{x} \in \mathbf{\Lambda}} \rho_{\sigma,\mathbf{c}}(\mathbf{x}))$ for all $\mathbf{z} \in \Lambda$.

SIS problem Let $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ be an $m \times n$ matrix with entries in $\mathbb{Z}_q$ that consists of m uniformly random vectors $\mathbf{a_i} \in \mathbb{Z}_q^n$: $\mathbf{A} = [\mathbf{a_1} | \cdots | \mathbf{a_m}]$. The Short Integer Solution (SIS) problem asks to find a non-zero vector $\mathbf{x} \in \mathbb{Z}^m$ such that: $\|\mathbf{x}\| \leq \beta$ for some $\beta \in \mathbb{R}$ and $\mathbf{x}\mathbf{A} = \mathbf{0} \in \mathbb{Z}_q^n$. In order to guarantee that such a non-trivial, short solution exists, it is required that $\beta \geq \sqrt{n \log q}$, and $m \geq n \log q$. The R-SIS problem is a special case of the SIS problem, where the matrix $\mathbf{A}$ is restricted to negacyclic blocks $\mathbf{A} = [\mathsf{rot}(\mathbf{a_1}) | \cdots | \mathsf{rot}(\mathbf{a_m})]$

GPV Framework The GPV framework is a way to construct lattice-based signatures based on the hash-and-sign paradigm. At the high-level, the GPV framework works as follows: The public key is a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, where $n > m$, and the private key is a matrix $\mathbf{B} \in \mathbb{Z}_q^{m\times m}$. Public and private key are linked by the relation $\Lambda_q(\mathbf{A})^\perp = \Lambda_q(\mathbf{B})$. A signature for a given a message $\mathtt{m} \in \{0,1\}^*$ is a short vector $\mathbf{s} \in \mathbb{Z}_q^m$ such that $\mathbf{s}\mathbf{A}^* = \mathcal{H}(\mathtt{m})$, where $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q^n$ is a hash function. The function which finds $\mathbf{s}$ is called a pre-image trapdoor sampler. Given the public-key $\mathbf{A}$, it is straightforward to verify that $\mathbf{s}$ is a valid signature: it is sufficient to check that the signature $\mathbf{s}$ is indeed short and verify that $\mathbf{s}\mathbf{A}^* = \mathcal{H}(\mathtt{m})$. The GPV framework is proven secure in the (quantum) random oracle model assuming the hardness of the SIS problem.

NTRU Lattices Let $\phi = x^n + 1$ for $n = 2^\kappa$ and $q \in \mathbb{N}^\star$. A set of NTRU secrets consists of four polynomials $f, g, F, G \in \mathbb{Z}[x]/(\phi)$ which verify the NTRU equation $fG - gF = q \bmod \phi$. Additionally, it is required that f is invertible modulo q. Then the public polynomial h is defined by $h \leftarrow gf^{-1} \bmod q$. The matrices $\mathbf{A} = \left[\begin{array}{c|c} 1 & h \\ \hline 0 & q \end{array}\right]$ and $\mathbf{B} = \left[\begin{array}{c|c} f & g \\ \hline F & G \end{array}\right]$ generate the same lattice, but the matrix $\mathbf{A}$ contains large polynomials, whereas the $\mathbf{B}$ matrix contains only small polynomials. Recovering the secret polynomials from the knowledge of h corresponds to breaking the NTRU problem.

Trapdoor Sampling A trapdoor sampler takes a matrix $\mathbf{A}$, some additional information T (the 'trapdoor') and a target vector $\mathbf{c}$ as inputs and returns a short vector $\mathbf{s}$ as output, such that $\mathbf{s}\mathbf{A}^* = \mathbf{c} \bmod q$. Note that, using basic linear algebra, it is easy to find a vector $\mathbf{c_0}$ such that $\mathbf{c_0}\mathbf{A}^* = \mathbf{c} \bmod q$. Finding a short vector $\mathbf{s}$ is then equivalent to finding a vector $\mathbf{v} = \mathbf{s} - \mathbf{c_0} \in \Lambda_q^\perp(\mathbf{A})$ close to $\mathbf{c_0}$. In the GPV framework, the secret basis $\mathbf{B}$ serves as the trapdoor to achieve this.

Gram-Schmidt Orthogonalization (GSO) Let n, m be integers. Let $\mathbf{B} \in \mathbb{R}^{n\times m}$ be a full-rank matrix. The Gram-Schmidt Orthogonalization (GSO) of $\mathbf{B}$ is the unique matrix $\tilde{\mathbf{B}} = (\tilde{\mathbf{b}}_0, ..., \tilde{\mathbf{b}}_{n-1}) \in \mathbb{R}^{n\times m}$ such that $\mathbf{B} = \mathbf{L}\tilde{\mathbf{B}} \in \mathbb{R}^{n\times m}$, where $\mathbf{L}$ is a lower triangular with 1's on the diagonal and the vectors $\tilde{\mathbf{b}}_i$ are pairwise orthogonal. The value $\|\mathbf{B}\|_{\mathsf{GS}} = \max_{\tilde{\mathbf{b}}_i \in \tilde{\mathbf{B}}} \|\tilde{\mathbf{b}}_i\|$ is called the Gram-Schmidt norm of $\mathbf{B}$.

3 Falcon

Falcon stands for "Fast Fourier lattice-based compact signatures over NTRU" and instantiates the theoretical signature framework of [GPV07] using NTRU lattices and fast Fourier trapdoor sampling. For brevity, we will refer to the Falcon version v1.2 (as submitted to NIST [PFH+22]) simply as 'Falcon' and to the version defined in [LPa17] as 'deterministic Falcon'. In the following, we provide a brief description of those parts of Falcon which are required to make this paper self-contained. We refer the interested reader to the full Falcon specification [PFH+22] for those parts that are omitted here.

Parameters The degrees of the reduction polynomial $\phi = x^n + 1$ are 512 and 1024 for NIST security level I (128-bit) and security level V (256-bit), respectively. Hence, there are two variants of Falcon which are called Falcon-512 and Falcon-1024, respectively. The integer modulus is fixed to $q = 12289$ for both variants.

Key Generation The private key is generated by drawing two private polynomials $f, g \xleftarrow{\$} \mathcal{R}$ with small coefficients[1], where f is invertible modulo q, and by computing two unique private polynomials $F, G \in \mathcal{R}$ such that $fG - gF = q \bmod (x^n + 1)$. The public key is computed by computing a public polynomial $h \in \mathcal{R}$ such that $h = gf^{-1} \bmod q$. From these polynomials, the public basis matrix $\mathbf{A} \in \mathcal{R}^2$ and the private basis matrix $\mathbf{B} \in \mathcal{R}^{2\times 2}$ are defined as follows:

$$\mathbf{A} = \begin{pmatrix} 1 & h^* \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} g & -f \\ G & -F \end{pmatrix}, \tag{1}$$

where the corresponding lattices are defined as follows:

$$\Lambda^{\perp}(\mathbf{A}) = \{\mathbf{y} \in \mathcal{R}^2 : \mathbf{yA} = \mathbf{0}\}, \quad \Lambda(\mathbf{B}) = \{\mathbf{y} \in \mathcal{R}^2 : \mathbf{y} = \mathbf{xB} \,\text{for some}\, \mathbf{x} \in \mathcal{R}^2\}. \tag{2}$$

Note that these bases, and thus their associated lattices, are orthogonal modulo q, i.e., $\mathbf{BA}^* = \mathbf{0} \bmod q$ and $\Lambda^{\perp}(\mathbf{A}) = \Lambda(\mathbf{B})$. Also note that the secret key is not directly stored as $\mathbf{B}$, rather as $\mathsf{sk} = (\hat{\mathbf{B}}, \mathsf{T})$, where $\hat{\mathbf{B}}$ is the entry-wise Fourier representation of the private basis $\mathbf{B}$, and T is a binary tree representing the GSO of $\mathbf{B}$. The public key of Falcon is defined as $\mathsf{pk} = h$.

Signature Generation Signature generation consists in hashing a message $\mathtt{m}$, along with a random nonce r, into a polynomial c modulo ϕ, whose coefficients are uniformly mapped to integers in the 0 to $q-1$ range. Then, a pair of short polynomials (s_1, s_2) such that $s_1 = c - s_2 h \bmod \phi \bmod q$ is generated using the secret lattice basis (f, g, F, G). Note that it is sufficient to transmit s_2 as the signature, because s_1 can be computed from (s_2, c, h). The signature algorithm of Falcon is illustrated in Alg.1.

Algorithm 1 Falcon.Sign(m, sk)

```
Require: A message $\mathtt{m}$, a secret key $\mathsf{sk} = (\hat{\mathbf{B}}, T)$
Ensure: A signature sig
1: procedure Falcon-Sign(m,sk)
2:     $r \xleftarrow{\$} \{0,1\}^{320}$
3:     $c \leftarrow$ HashToPoint($r||\mathtt{m}, q, n$)          ▷ $c \in \mathcal{R}$
4:     $\hat{\mathbf{t}} \leftarrow (\hat{c}, 0)\hat{\mathbf{B}}^{-1}$          ▷ pre-image $\hat{\mathbf{t}}$
5:     do
6:         $\hat{\mathbf{z}} \leftarrow$ ffSampling($\hat{\mathbf{t}}$, T)          ▷ trapdoor sampler
7:         $\hat{\mathbf{s}} \leftarrow (\hat{\mathbf{t}} - \hat{\mathbf{z}})\hat{\mathbf{B}}$
8:     while $\|\mathbf{s}\|^2 > \lfloor 2.42 n\sigma^2 \rfloor$
9:     return sig = $(r, \mathbf{s})$
```

Trapdoor Sampling The trapdoor sampler of Falcon is implemented by the function ffSampling, which takes a binary tree T (representing the GSO of the private basis) and a pre-image $\hat{\mathbf{t}}$ as input and returns an output vector $\hat{\mathbf{z}}$ such that $\mathbf{z} \in \Lambda$ and $\mathbf{s} = (\mathbf{t} - \mathbf{z})\mathbf{B} \sim D_{(c,0)+\Lambda(\mathbf{B}),\sigma,0}$, as illustrated in Alg. 2. The trapdoor sampler of Falcon samples signatures of norm proportional to $\|\mathbf{B}\|_{GS}$ using a discrete Gaussian sampler SamplerZ with varying mean μ and standard deviation σ', as shown in Alg. 3.

The discrete Gaussian sampler SamplerZ calls another random sampler called BaseSampler, whose distribution is statistically close to a half Gaussian distribution centered in 0 with a fixed standard deviation.

[1] In practice, the coefficients of the polynomials f and g are generated following a discrete Gaussian distribution with center 0 and standard deviation $\sigma = 1.17\sqrt{q/2n}$.

The BaseSampler function is shown in Alg. 4, where $[\![P]\!]$ is a function that, for any logical proposition P, returns 1 if P is true or 0 otherwise, and RCDT is a reverse cumulative distribution table.

Algorithm 2 ffSampling$_n(\mathbf{t}, \mathsf{T})$

```
Require: $\hat{\mathbf{t}} = (t_0, t_1) \in \mathsf{FFT}(\mathbb{Q}[x]/(x^n+1))^2$, a Falcon-tree $\mathsf{T}$
Ensure: $\hat{\mathbf{z}} = (z_0, z_1) \in \mathsf{FFT}(\mathbb{Z}[x]/(x^n+1))^2$
 1: procedure ffSampling$_n(\mathbf{t}, \mathsf{T})$
 2:     if $n = 1$ then
 3:         $\sigma' \leftarrow \mathsf{T.value}$
 4:         $z_0 \leftarrow$ SamplerZ$(t_0, \sigma')$
 5:         $z_1 \leftarrow$ SamplerZ$(t_1, \sigma')$
 6:         return $\hat{\mathbf{z}} = (z_0, z_1)$
 7:     $(\ell, \mathsf{T}_0, \mathsf{T}_1) \leftarrow (\mathsf{T.value}, \mathsf{T.leftchild}, \mathsf{T.rightchild})$
 8:     $\hat{\mathbf{t}}_1 \leftarrow$ splitfft$(t_1)$
 9:     $z_1 \leftarrow$ ffSampling$_{n/2}(\hat{\mathbf{t}}_1, \mathsf{T}_1)$
10:     $z_1 \leftarrow$ mergefft$(\hat{\mathbf{z}}_1)$
11:     $t_0' \leftarrow t_0 + (t_1 - z_1) \cdot \ell$
12:     $\hat{\mathbf{t}}_0 \leftarrow$ splitfft$(t_0')$
13:     $\hat{\mathbf{z}}_0 \leftarrow$ ffSampling$_{n/2}(\hat{\mathbf{t}}_0, \mathsf{T}_0)$
14:     $z_0 \leftarrow$ mergefft$(\hat{\mathbf{z}}_0)$
15:     return $\hat{\mathbf{z}} = (z_0, z_1)$
```

Algorithm 3 SamplerZ(-)

```
Require: $\mu, \sigma' \in \mathbb{R}$ with $\sigma_{\min} \leq \sigma' \leq \sigma_{\max}$
Ensure: $z \in \mathbb{Z}$ is sampled from a distribution close to $D_{\mathbb{Z},\mu,\sigma'}$
 1: procedure SamplerZ$(\mu, \sigma')$
 2:     $r \leftarrow \mu - \lfloor \mu \rfloor$
 3:     $c \leftarrow \sigma_{\min}/\sigma'$
 4:     while True do
 5:         $z_0 \leftarrow$ BaseSampler()
 6:         $b \leftarrow$ UniformBits(8)&1
 7:         $z \leftarrow b + (2b-2)z_0$
 8:         $x \leftarrow \frac{(z-r)^2}{2\sigma'^2} - \frac{z_0^2}{2\sigma_{\max}{}^2}$
 9:         if BerExp$(x, c) = 1$ then        ▷ See [PFH+22] for the definition of BerExp.
10:             return $z + \lfloor \mu \rfloor$
```

Algorithm 4 BaseSampler(-)

```
Require: -
Ensure: An integer $z^+ \sim D_{\mathbb{Z}^+,\sigma_{\max},0}$
 1: procedure BaseSampler(-)
 2:     $u \leftarrow$ UniformBits(72)
 3:     $z^+ \leftarrow 0$
 4:     for $i \leftarrow 0 \rightarrow 17$ do
 5:         $z^+ \leftarrow z^+ [\![u < \mathrm{RCDT}[i]]\!]$
 6:     return $z^+$
```

Deterministic Signature Generation In deterministic Falcon, signature generation uses a fixed (versioned) salt value instead of generating a random value r as described on Line 3. The fixed salt value consists of a byte representing the value of $\ell = \log_2(n)$ and an ASCII representation of the string FALCON_DET followed by all-zero padding bytes. The fixed salt is omitted from the signatures themselves. We refer the interested reader to the full specification of deterministic Falcon given in [LPa17].

Signature Verification Given a message $\mathtt{m}$, a signature s_2 and a public key h, the verifier first computes the hash of the message $c = \mathcal{H}(\mathtt{m})$, then computes the first signature component $s_1 = hs_2$, and finally verifies that (s_1, s_2) is sufficiently short. If so, the signature is accepted, otherwise the verifier rejects the signature.

4 The fault attack against deterministic Falcon

In this section, we first explain the general idea of the attack against deterministic Falcon and then go into the details and describe how to make it work in practice.

Suppose we can obtain two different signatures $\mathbf{s}, \mathbf{s}'$ for the same message hash c as in Alg. 3, but with different outputs of the trapdoor sampler ffSampling. We define the vector $\mathbf{v}$ as the difference between the two signatures:

$$\mathbf{v} = (v_1, v_2) = (s_1 - s_1', s_2 - s_2') = \mathbf{s} - \mathbf{s}'. \tag{3}$$

Because

$$\mathbf{v}\mathbf{A}^* = s\mathbf{A}^* - s'\mathbf{A}^* = (0, c) - (0, c) = 0, \tag{4}$$

we have found a vector $\mathbf{v}$ in the lattice generated by $\mathbf{B}$:

$$\mathbf{v} \in \Lambda^{\perp}(\mathbf{A}) = \Lambda(\mathbf{B}). \tag{5}$$

Moreover, because both signatures $\mathbf{s}$ and $\mathbf{s}'$ are relatively short, so it is also their difference $\mathbf{v}$ short. More precisely, we have:

$$\|\mathbf{v}\| = \|\mathbf{s} - \mathbf{s}'\| \leq \|\mathbf{s}\| + \|\mathbf{s}'\| \leq 2\sqrt{\lfloor 2.42 n\sigma^2 \rfloor}. \tag{6}$$

Also note that, due to the definition of $\mathbf{B}$ in Eq. (1), there exists a vector $\mathbf{u} = (u_1, u_2) \in \mathcal{R}^2$ such that

$$(v_1, v_2) = \mathbf{v} = \mathbf{u}\mathbf{B} = (u_1, u_2)\begin{pmatrix} g & -f \\ G & -F \end{pmatrix} = (u_1 g + u_2 G, -u_1 f - u_2 F), \tag{7}$$

so v_2 lies in the sublattice of $\mathcal{R}$ generated by the secret key elements f, F.

Now, if we inject faults into the pseudo-random number generator called by ffSampling, while repeatedly signing the same message with a deterministic variant of Falcon, we are in exactly the following situation: we obtain different valid signatures for the same message hash and subtracting different faulty signatures from each other we obtain several v_2 as just described above. If we have enough v_2 we can try to apply lattice reduction techniques to obtain f, F or an equivalent representation of the private key.

To summarize the main idea of our attack, by injecting faults into subroutines of ffSampling, we want to obtain a set of different signatures $\{\mathbf{s}^{(0)}, \mathbf{s}^{(1)}, \ldots \mathbf{s}^{(m-1)}\}$ for the same input message. Subtracting pairs of different signatures from each other, gives us a set of "somewhat short" vectors

$$\{s_2^{(j)} - s_2^{(k)} \,|\, 0 \leq j < m, 0 \leq k < m, j \neq k\} \tag{8}$$

in a sublattice generated by f, F.

The obvious approach to obtain f or F from these vectors is lattice reduction. However, since the rank of the lattice generated by f, F is 512 for Falcon-512 and 1024 for Falcon-1024, reducing a "somewhat short" basis of vectors like v_2 to find a "really short" f can be still computationally hard in practice. To make our attack feasible in practice, we show a simple way to work in a suitable sublattice of smaller rank in the next subsections.

4.1 Fault attack in a suitable sublattice

In order to obtain a practically feasible attack, we would like to work in a lower-rank sublattice that still contains the private key f. Injecting a fault in ffSampling routine will result in an incorrect value $\hat{\mathbf{z}}$ of the signing procedure (cf. Alg. 1, line 6). Looking at Alg. 1, line 7 we see that

$$\hat{\mathbf{s}} - \hat{\mathbf{s}}' = (\hat{\mathbf{t}} - \hat{\mathbf{z}})\hat{\mathbf{B}} - (\hat{\mathbf{t}} - \hat{\mathbf{z}}')\hat{\mathbf{B}} = (\hat{\mathbf{z}}' - \hat{\mathbf{z}})\hat{\mathbf{B}}. \tag{9}$$

Substituting the definition of private basis matrix $\mathbf{B}$ from Eq. (1) we obtain

$$\hat{s}_2 - \hat{s}_2' = (\hat{z}_0 - \hat{z}_0') \odot \hat{f} + (\hat{z}_1 - \hat{z}_1') \odot \hat{F}, \tag{10}$$

where $\odot$ denotes component-wise multiplication.

Now let us have a closer look at Alg. 2 which performs trapdoor sampling in a recursive way. The output z_1 does not depend on the recursive call on line 13. If we track the call-tree of recursions in ffSampling, we see therefore that z_1 does not depend on the later calls to SamplerZ on lines 4 and 5. These later calls to SamplerZ in turn result in calls of BaseSampler and UniformBits. So faults injected in the later calls to UniformBits will only affect the output z_0 of the ffSampling call at the top of the recursion call tree. Hence, if the fault is injected in this way, then $z_1 = z_1'$ and Eq. (10) becomes

$$\hat{s}_2 - \hat{s}_2' = (\hat{z}_0 - \hat{z}_0') \odot \hat{f}. \tag{11}$$

Because the Fourier inverse is a linear operation, we obtain

$$s_2 - s_2' = (z_0 - z_0')f. \tag{12}$$

Note that the same calculations hold if we assume that $\mathbf{s}$ is also a faulty signature (but different from $\mathbf{s}'$). In this case, $z_0 - z_0'$ is the difference between the two injected faults.

Injecting faults repeatedly and taking the difference between two faulty signatures (or a faulty and a correct one) we can produce a set

$$\Delta = \{\delta_1 f, \delta_2 f, \ldots, \delta_m f\}, \tag{13}$$

where $\delta_i = z_i - z_i'$. We consider the lattice generated by Δ:

$$\Lambda(\Delta) \subset \Lambda(f) \subset \mathcal{R}. \tag{14}$$

If Δ is large enough, then there is a possibility that $f \in \Lambda(\Delta)$. Additionally, if the rank of $\Lambda(\Delta)$ is small enough, e.g. 40, we can apply lattice reduction techniques to find f.

Now the question is, what are the possible values of the δ_i, because this determines the rank of the sublattice generated by Δ. For example, if $\delta_i(x) = a_i + b_i(x)$ for all i, then the rank of the lattice $\Lambda(\Delta)$ is at most 2. In order to get a better understanding of the range of possible values for the δ_i, we have to have a closer look at the effect of a fault injection in concrete instance of the pseudo-random number generator (PRNG) used by UniformBits.

4.2 The PRNG of deterministic Falcon

This paragraph is specific to the ChaCha20-based PRNG implementation used in the pre-image sampler of the deterministic FALCON reference implementation provided in [LPa17], which is actually the the same as in the reference implementation of standard FALCON submitted to NIST as [PFH+22]. However, note that similar considerations are transferable to other PRNG implementations whenever necessary.

Essentially, the PRNG uses the key stream generated by the ChaCha20 stream cipher as pseudo-random output. The PRNG maintains a 256 byte buffer. So a fresh ChaCha20 pseudo-random output is only produced, whenever the buffer has been exhausted.

It seems entirely possible that a fault can be injected even during the fetching from this buffer. However, it seems far easier to inject a fault into a ChaCha20 computation, because this takes a lot longer than just fetching data from a buffer. The fault can have any of several well-known effects like instruction skipping or data corruption to cause a faulty output. Note that the structure of a ChaCha20 computation is normally easy to identify in a power trace. Hence, it is generally not difficult to identify trigger points for injecting a suitable fault.

We have to distinguish between two types of faults. The first type is a fault that just affects the output u of a single ChaCha20 computation underlying the call of UNIFORMBITS in Alg. 4, line 2. Because the output of ChaCha20 is buffered, a single fault can lead to up to 256 faulty output bytes. This will affect several calls to UNIFORMBITS and hence affect several z_0, z_1 in Alg. 2, lines 4 and 5.

The second type of fault changes the seed of the PRNG in UNIFORMBITS. The reference implementation uses ChaCha20 to implement UNIFORMBITS. Because ChaCha20 adds its output to its internal state, this type of fault is quite likely. Note that this second type of fault affects the current and all future outputs of BASESAMPLER and hence potentially more z_0, z_1 in Alg. 2, lines 4 and 5 than the first type of fault.

Note that a fault may not necessarily result in a different signature. For example, different values of u can result in the same value of z^+ in Alg. 4.

Because of the 256 byte buffer and because SAMPLERZ needs 10 pseudo-random bytes at a time, we expect a single fault of the first type to generate values δ from a roughly 25 dimensional subspace of $\mathsf{FFT}(\mathbb{Q}[x]/(x^n+1))$. As Fourier transformation is a linear operation, we can hence assume that the resulting differences v between faulty signatures will lie in a sublattice of rank ≈ 25. This is well within reach of current lattice reduction algorithms. See, for example, [ADH+19] for a discussion of modern sieving algorithms. We only need a basis of a sublattice of rank about 25, so we expect that with around 25 suitable faults an attacker should have a good chance of finding the secret key.

4.3 Combining the fault attack with exhaustive search

Recall that, according to Eq. (14), $\Lambda(\Delta)$ is only a sublattice of $\Lambda(f)$. So there is no guarantee that lattice reduction with Δ finds the secret private key f. In general, it will only output cf, for some polynomial $c \in \mathcal{R}$.

If $c = x^k$, then cf is just a negacyclic rotation of the original f. This is not a problem for the attacker, as all negacyclic rotations of f are equivalent private keys. Indeed, if (f, g, F, G) is a set of secret NTRU-polynomials, then $(x^k f, x^k g, -x^{n-k}F, -x^{n-k}G)$ also satisfies the NTRU equation

$$(x^k f)(-x^{n-k}G) - (x^k g)(-x^n - kF) = -x^n fG + x^n gF = fG - gF = q \bmod \phi. \quad (15)$$

The public polynomial $h = g/f = (x^k g)/(x^k f)$ is the same for both sets of NTRU secrets. Hence, both NTRU secrets are equivalent in the sense that messages signed with either of them can be verified by the same public key.

If $c = c_k x^k$ with $c_k \neq \pm 1$, and cf is relatively short, then the coefficients of cf will have a greatest common divisor $|c_k| \neq 1$. This situation is easy to recognize for an attacker. The attacker just has to divide the candidate private key component cf by the greatest common divisor of its coefficients.

If c is not simply a monomial, but cf is relatively short (say up to about four times the length of f), then c is most likely sparse and its few non-zero coefficients are small. In these cases, an attacker can try to find c by exhaustive search and hence still determine the private key f. Note that, because rotated keys are equivalent, as we have just seen, the attacker can assume that one of the non-zero coefficients of c is simply c_0.

4.4 Extending the attack to Falcon

The fault attack described so far works only for a deterministic version of Falcon, for example, as described in [LPa17]. The standard non-deterministic version of Falcon described in [PFH+22] randomizes the function HashToPoint, which hashes the input to a polynomial in $\mathcal{R}$, as illustrated in Alg. 1. Hence, if the same message is signed twice with the standard version of Falcon, then ffSampling in line 6 is called for two different values of $\hat{\mathbf{t}}$ with overwhelming probability. Thus, taking the difference of these two signatures is very unlikely to return a short vector, and so it becomes computationally infeasible to find f by lattice reduction techniques.

To apply our attack to standard Falcon, a second fault injection is necessary to make standard Falcon behave in a deterministic way. For example, if the random value r in Alg. 1, line 3 can be forced to a fixed value by a fault injection or other means, then it will be possible to apply our fault attack as described above. Note, that the fault to force r to a fixed value does not have to be necessarily perfectly reproducible. Faulty signatures with the "wrong" r can be easily recognized by looking the difference with another signature, then this difference vector would be noticeably larger than expected. Note also that, therefore, it is not necessary to force r into one particular value. It is sufficient to force it to one of a small set $\{r_0, r_1, \ldots, r_{m-1}\}$ of possible values. The resulting faulty signatures can easily be grouped into sets coming from r_1, r_2 etc. by looking at the size of pairwise differences. This highlights the importance of using a fault attack resistant random number generator in Alg. 1, line 3, e.g., by using a fault-resistant implementation of ChaCha20, e.g., [ZMR19].

5 Practical evaluation

In order to validate the feasibility of our attack and our hypotheses on the number of faults required to a successful attack, we conducted a number of experiments with the Falcon implementations provided in [PD22] and [LPa17].

5.1 Simulations on a PC

We modified the code in [PD22] slightly to make it deterministic as described in [LPa17] and added instrumentation for simulating the injection of faults of the two types described above.

We simulated the attack for 10 randomly chosen keys for each of Falcon-512 and Falcon-1024. For each key we chose 10 random inputs. These inputs were hashed and signed by the implementation we have just described. As mentioned in Sec. 4.2, we distinguish between two types of faults: A fault may either only affect one block of output of the PRNG or it may affect the state of the PRNG and hence all future outputs. We call the former type of fault 'transient' and the latter 'persistent'. We simulated each type of fault with 25 and 50 attacks each. Key and input were fixed for each attack, as this is a

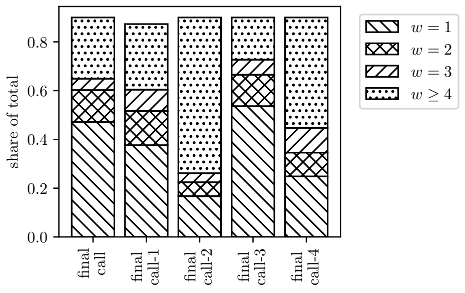

Figure 1: 25 attacks against FALCON-512 with persistent faults. See text for further details.

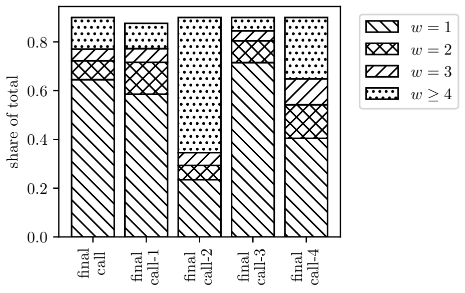

Figure 2: 50 attacks against FALCON-512 with persistent faults. See text for further details.

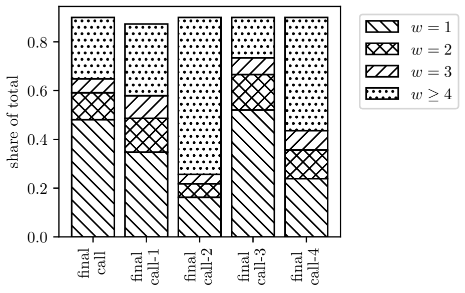

Figure 3: 25 attacks against FALCON-512 with transient faults. See text for further details.

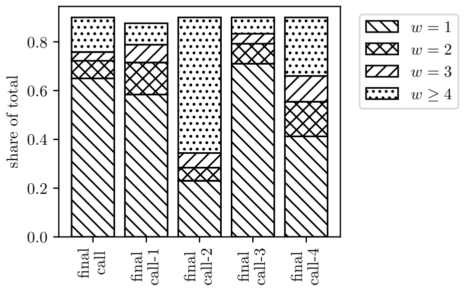

Figure 4: 50 attacks against FALCON-512 with transient faults. See text for further details.

requirement for the attack to work. We injected exactly one fault in each experiment. We attacked each of the final five calls of the PRNG. We repeated each of these experiments 10 times. To give an idea of the remaining effort to find the key by exhaustive search after the lattice reduction step as described in Sec. 4.3, we also give the number w of non-zero coefficients of s, when sf is the key candidate found by the lattice reduction step and f is the actual private key the attacker is looking for.

Summaries of the results of these experiments are illustrated in Fig. 1 to 8. The sum of the slices with $w = 1, 2, 3$ and $w \geq 4$ is not necessarily 1, because there are cases in which all signatures are the same, despite injected faults. This can happen, for example, because of rounding. Interestingly, all figures look quite similar. It is not suprising to see that the probability for an (almost) immediate success, i.e., a result with $w = 1$, is higher with 50 faults than with 25. In every experiment, for each of the 10 keys, there was one attack that revealed a rotation of the key immediately without exhaustive search. This is not obvious from the figures, but it also illustrates the effectiveness of the attack. To summarize, we see that the attack has a high probability of success, even with a very moderate number of faults.

5.2 Simulations on ARM Cortex-M4

We simulated our attack also on a STMF407G-DISC1 board featuring an a 32-bit ARM Cortex-M4 (`STMF407VGT6`) high-performance microcontroller with FPU core, 1-Mbyte Flash memory, and 192-Kbyte RAM. We ran our attack using the C implementation of deterministic FALCON-1024 available at [LPa17][2] compiled with `gcc version 12.2.0` and optimization flags `-O3 -mfloat-abi=hard -mfpu=fpv4-sp-d16`. We simulated the

[2] `commit 02a2a64c44147775e6870b2d957f2cfda1437895`

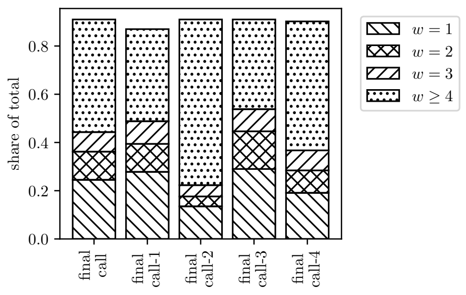

Figure 5: 25 attacks against Falcon-512 with persistent faults. See text for further details.

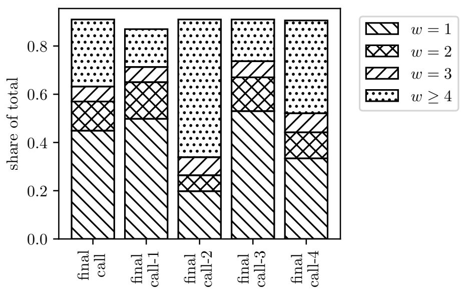

Figure 6: 50 attacks against Falcon-512 with persistent faults. See text for further details.

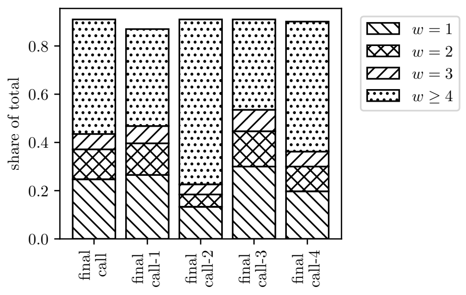

Figure 7: 25 attacks against Falcon-512 with transient faults. See text for further details.

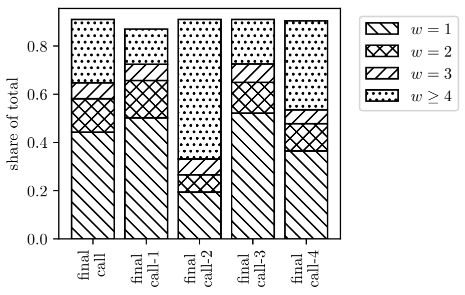

Figure 8: 50 attacks against Falcon-512 with transient faults. See text for further details.

injection of random faults in the register `r2` right before the execution of the assembly instruction `eors r3, r2` (cf. Listing 2 on line 4) with the aid of a debugger (`GNU gdb 13.1`). In practice, this corresponds in randomly changing the content of lower part of the 64-bit variable `cc` at line 287 of the file `rng.c` (cf. Listing 1 on line 4).

Listing 1: C code

```
1   uint64_t cc;
2   uint32_t state[16];
3   ...
4   state[14] ^= (uint32_t)cc;
```

Listing 2: Assembly code

```
1   ldr     r2, [sp, #32]
2   ldr     r3, [sp, #136]
3   ...
4   eors    r3, r2
```

In this way, we are able to perturbate the state of ChaCha20, hence the generation of random numbers operated by the function `falcon_inner_prng_refill()`, thus leading to the generation of different signatures for the same input message. We were able to fully recover (a rotation) of the secret key f using 100 faulty signatures, when a random fault was injected in the fourth to last execution of `falcon_inner_prng_refill()` for a fixed input message with 78 executions, hence leading us to be able to forge legitimate signatures for it. When the faults were injected in the third to last execution of `falcon_inner_prng_refill()`, it was possible to recover the secret key f up to an additional (yet practically doable) bruteforce effort. The attack recovered cf, as in section 4.3, where c had only two non-zero coefficients; these were both from the set $\{-1, 1\}$.

We repeated the attack injecting random instruction skips during the third to last execution of `falcon_inner_prng_refill()`. We injected 100 faults, 20 of which lead to an unresponsive behavior, and 8 of which were unsuitable because lead to the same output. In this case, we were able to recover the secret key f up to a practicable final bruteforce effort. We stress the fact these attacks were run the attack only for a fixed key and input message to verify their applicability in practice. By using different inputs results may change as shown by the statistical simulations of Sec. 5.1.

Finally, please note that, although the attack was carried out without using a real fault injection setup, the simulation results on real hardware clearly indicate that the attack is practically feasible as long as random instructions skip or random faults in a 32-bit register, memory, or during the computation of an XOR-operation are possible during the execution of the `falcon_inner_prng_refill()` function. All these kind of fault injections are typically not considered difficult to achieve with a low-cost fault injection setup, e.g., [GGD17].

6 Countermeasures

The typical defence against fault attacks in signature generation is to verify the signature before returning it as output. However, this does not work in our case, because the faulty signatures generated by our attack are all perfectly valid signatures. (An attacker can actually exploit this property to filter the faulty signatures generated during an attack: A fault attack may sometimes produce random output or a fault of the wrong type. To avoid these 'faulty faults' breaking the lattice reduction step, an attacker can check whether the faulty signatures are valid and throw away those that are not.) An obvious countermeasure is a complete re-calculation of each signature or at least re-compute ffSampling. Of course this incurs a significant performance penalty. Also, it does not prevent the attack if the attacker is able to inject the same fault in both signature calculations.

Randomly shuffling the order of the computation of the sampled coefficients in ffSampling may increase significantly the effort of an attacker. In fact, it is impossible for an attacker to guarantee that an injected fault only affects the value z_0 returned by ffSampling in this case. The attacker is still able to find $\mathbf{s} - \mathbf{s}' \in \Lambda(\mathbf{B})$ as in Eq. (9), but $s_2' - s_2$ is not necessarily in the sublattice just generated by f, so the simplification from Eq. (10) to Eq. (11) does not apply anymore. Instead, the attacker obtains vectors in the larger lattice generated by both f and F. This makes the lattice reduction harder and may make it infeasible to obtain f or at least pf for some sparse polynomial p with small coefficients.

However, note that this is not possible with the procedure illustrated in Alg. 2, because the input to the second recursive call in line 13 depends on the output of the first recursive call in line 9. Also note that the ChaCha20-based PRNG that is currently used by Falcon does not allow to compute an arbitrary coefficient z_j in Alg. 2, lines 4 and 5 without computing all previous (in the standard order) z_i first. The reason is, that Alg. 3 does not consume a fixed number of pseudo-random bits. Hence, the same number of pseudo-random bits generated for the computation of each z_i may be different for different i.

So the introduction of shuffling in ffSampling would require replacing the sampling algorithm by one with parallelizable steps and changing the PRNG in such a way that the index of the coefficient becomes a parameter. Such a change would allow to generate the bitstreams each coefficient independently and would hence make it possible to shuffle the order in which the coefficients are calculated. Note that signature generation would still be deterministic in this case.

If changes to the PRNG are undesirable in a concrete instantiation of a deterministic Falcon signature scheme, then the PRNG can be re-run and the output (or a checksum over the output) and its final state compared against the values for the original run when the siganture was produced. However, this offers only incomplete protection, because a fault does not necessarily have to be injected into the PRNG for our attack.

7 Conclusion and future work

In this work, we have described the first fault attack of deterministic Falcon that exploits specific algorithmic properties of Falcon. The attack works under a very generic and realistic fault model using random faults injected during the execution of the pre-image trapdoor sampler. The attack can be naturally transferred to non-deterministic standard Falcon by suppressing the entropy in the call to HashToPoint in standard Falcon with another fault injection (or, by other attack means).

It would be interesting to transfer the result of this work to deterministic variants of other signature schemes based on the GPV framework, such as Mitaka and ModFalcon (see [EFG+21] and [CPS+19], respectively). Another possible line of research is the practical investigation of the countermeasures we have suggested in Sec. 6. Finally, it would be interesting to perform an evaluation of our attack with a real fault injection setup.

Acknowledgments This work was partly funded by the German Federal Ministry of Education and Research in the project AQUORYPT through grant number 16KIS1012.

References

[ABB+16] Sedat Akleylek, Nina Bindel, Johannes A. Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. An efficient lattice-based signature scheme with provably secure instantiation. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 44–60. Springer, Heidelberg, April 2016.

[ABF+18] Christopher Ambrose, Joppe W. Bos, Björn Fay, Marc Joye, Manfred Lochter, and Bruce Murray. Differential attacks on deterministic signatures. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 339–353. Springer, Heidelberg, April 2018.

[ADH+19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 717–746. Springer, Heidelberg, May 2019.

[BAA+17] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. qTESLA. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[BBK16] Nina Bindel, Johannes Buchmann, and Juliane Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*, pages 63–77. IEEE Computer Society, 2016.

[BP16] Alessandro Barenghi and Gerardo Pelosi. A note on fault attacks against deterministic signature schemes. In Kazuto Ogawa and Katsunari Yoshioka, editors, *IWSEC 16*, volume 9836 of *LNCS*, pages 182–192. Springer, Heidelberg, September 2016.

[BP18] Leon Groot Bruinderink and Peter Pessl. Differential fault attacks on deterministic lattice signatures. *IACR TCHES*, 2018(3):21–43, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7267.

[CPS$^+$19] Chitchanok Chuengsatiansup, Thomas Prest, Damien Stehlé, Alexandre Wallet, and Keita Xagawa. ModFalcon: compact signatures based on module NTRU lattices. Cryptology ePrint Archive, Report 2019/1456, 2019. https://eprint.iacr.org/2019/1456.

[CSC$^+$22] Weiqiong Cao, Hongsong Shi, Hua Chen, Jiazhe Chen, Limin Fan, and Wenling Wu. Lattice-based fault attacks on deterministic signature schemes of ECDSA and eddsa. In Steven D. Galbraith, editor, *Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings*, volume 13161 of *Lecture Notes in Computer Science*, pages 169–195. Springer, 2022.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, Heidelberg, August 2013.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 22–41. Springer, Heidelberg, December 2014.

[EFG$^+$21] Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Mitaka: a simpler, parallelizable, maskable variant of falcon. Cryptology ePrint Archive, Report 2021/1486, 2021. https://eprint.iacr.org/2021/1486.

[EFGT16] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Loop-abort faults on lattice-based Fiat-Shamir and hash-and-sign signatures. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 140–158. Springer, Heidelberg, August 2016.

[GGD17] Oscar M. Guillen, Michael Gruber, and Fabrizio De Santis. Low-cost setup for localized semi-invasive optical fault injection attacks - how low can we go? In Sylvain Guilley, editor, *COSADE 2017*, volume 10348 of *LNCS*, pages 207–222. Springer, Heidelberg, April 2017.

[GLP12] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, Heidelberg, September 2012.

[GPV07] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. Cryptology ePrint Archive, Report 2007/432, 2007. https://eprint.iacr.org/2007/432.

[HBD$^+$22] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[LDK$^+$17] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[LDK$^+$22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[LPa17] David Lazar, Chris Peikert, and algoidan. Deterministic falcon implementation. https://github.com/algorand/falcon, Accessed 2022-11-17.

[LZA18] Xingye Lu, Zhenfei Zhang, and Man Ho Au. Practical signatures from the partial Fourier recovery problem revisited: A provably-secure and Gaussian-distributed construction. In Willy Susilo and Guomin Yang, editors, *ACISP 18*, volume 10946 of *LNCS*, pages 813–820. Springer, Heidelberg, July 2018.

[MHS$^+$19] Sarah McCarthy, James Howe, Neil Smyth, Seamus Brannigan, and Máire O'Neill. BEARZ attack FALCON: Implementation attacks with countermeasures on the FALCON signature scheme. Cryptology ePrint Archive, Report 2019/478, 2019. https://eprint.iacr.org/2019/478.

[NIS22] NIST. NIST announces first four quantum-resistant cryptographic algorithms. https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms, 2022. Accessed 2022-12-21.

[PD22] Thomas Prest and 'Dan'. falcon.py. https://github.com/tprest/falcon.py, 2022. Accessed 2022-12-31.

[PFH$^+$22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[PSS$^+$17] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. Attacking deterministic signature schemes using fault attacks. Cryptology ePrint Archive, Report 2017/1014, 2017. https://eprint.iacr.org/2017/1014.

[PSS$^+$18] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. Attacking deterministic signature schemes using fault attacks. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 338–352. IEEE, 2018.

[RP17] Yolan Romailler and Sylvain Pelissier. Practical fault attack against the ed25519 and eddsa signature schemes. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2017, Taipei, Taiwan, September 25, 2017*, pages 17–24. IEEE Computer Society, 2017.

[SAB$^+$22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[SB18] Niels Samwel and Lejla Batina. Practical fault injection on deterministic signatures: The case of EdDSA. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 306–321. Springer, Heidelberg, May 2018.

[ZMR19] Alexander Zeh, Manuela Meier, and Viola Rieger. Parity-based concurrent error detection schemes for the chacha stream cipher. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2019, Noordwijk, Netherlands, October 2-4, 2019*, pages 1–4. IEEE, 2019.