# Safe-Error Analysis of Post-Quantum Cryptography Mechanisms

Luk Bettale [*], Simon Montoya[*‡], Guénaël Renault[†‡]
[*]*Crypto and Security Lab IDEMIA Courbevoie, France.*
*Email: firstname.lastname@idemia.com*
[†]*ANSSI Paris, France.*
*Email: firstname.lastname@ssi.gouv.fr*
[‡]*LIX, INRIA, CNRS, École Polytechnique, Institut Polytechnique de Paris, France.*
*Email: firstname.lastname@lix.polytechnique.fr*

***Abstract*—The NIST selection process for standardizing Post-Quantum Cryptography Mechanisms is currently running. Many papers already studied their theoretical security, but the resistance in deployed device has not been much investigated so far. In particular, fault attack is a serious threat for algorithms implemented in embedded devices. One particularly powerful technique is to use safe-error attacks. Such attacks exploit the fact that a specific fault may or may not lead to a faulty output depending on a secret value. In this paper, we investigate the resistance of various Post-Quantum candidates algorithms against such attacks.**

***Keywords*-fault attacks; safe-error; post-quantum cryptography;**

## I. Introduction

Hypothetical quantum computers can theoretically be used to break some hard mathematical problems that would be intractable with classic computers. These problems such as the integer factorization [1] problem and the discrete logarithm problem are used today as the foundations of classic asymmetric cryptography in cryptosystems such as RSA or ECC. A practical break of these classes of cryptosystems would shatter all of the digital security. Fortunately, quantum computers are far from being ready to scale up to the size of modern crypto algorithms. Even if the threat is not imminent, studying alternatives is essential. In this context, the National Institute of Standard and Technology (NIST) has initiated a process to submit, evaluate and eventually standardize post-quantum asymmetric cryptographic algorithms (denoted PQC). These new algorihtms are based on various NP-hard problems that are supposed to be resistant to quantum computers.

The subsequent step is to deploy PQC in real-life applications. Cryptographic algorithms are usually deployed in various systems and devices. In particular, embedded devices are used in contexts where security is necessary: bank cards, sim cards, passports, *etc*. Such devices have much less computation power and RAM capacity. Among the NIST PQC candidates, some may just be too big to fit in an embedded device. Lattice-based cryptography seems to be the most suitable for embedded devices.

In order to protect the implementations against side-channel attacks, the NIST asks for constant-time implementation. This may be enough for implementations on a server, but insufficient for embedded devices. Contrarily to cryptography deployed in a distant server, embedded devices can be physically accessed by an attacker. The attacker could easily perform side-channel analysis [2]–[4], or fault injection [5], [6] in order to recover the embedded secrets. Fault attacks have been very useful to break embedded cryptosystems. Many powerful fault attacks have been descibed in the litterature [7]. Safe-Error Attack (denoted SEA) [8]–[10] is a powerful way to exploit fault injection, especially on constant-time implementation such as the ones proposed to the NIST.

An independent work in [11] performs SEA during the decryption algorithm of some lattice-based candidates (Kyber and Newhope). More precisely, they perform a fault injection during the reconciliation step. If the outcome is not modified, then they learn information depending on the secret key. By gathering such information they retrieve the entire secret key. Their SEA attack focuses on the error distribution while our work focuses on the secret distribution. Our work proposes a different way of performing SEA applied to more schemes than [11] and thus is it complementary.

To the best of our knowledge, no other public researches on fault attacks in the PQC context (e.g. [12]–[16]) consider SEA. In this paper, we investigate the security of PQC against such attacks.

In the following section, we describe our analysis framework, in particular, we describe the attacker model and the tools used for our analysis. In Sect. III, we apply our framework to some PQC algorithms that fit in embedded device. Finally we conclude in Sect. V.

## II. Framework Description

### A. Attacker Model

For a safe-error attack to be effective, an attacker has to know first how a specific fault impacts an implementation. This knowledge is usually acquired by characterising his attack on a similar device as the victim. To obtain the best characterisation possible, the attacker needs devices where he can *know/set* the manipulated secrets. For example, this assumption could be obtained by using an *open sample*. Then, the attacker has to find a fault that will lead to a faulty output only if some conditions on a secret value are fulfilled.

Safe-error attacks are particularly useful against constant-time implementations that are designed to resist timing side-channel attacks. A typical example is a regular implementation of a square-and-multiply algorithm used for modular exponentiation in RSA:

```
Input: $m, N, d = \sum_{i=0}^{n-1} d_i\, 2^i$
Output: $r = m^d \bmod N$
1: for $i = n-1$ to $0$ do
2:     $r_1 \leftarrow r_1^2 \bmod N$
3:     $r_{d_i} \leftarrow r_1 \times m \bmod N$
4: end for
5: return $r_1$
```

The operation at line 3 is useful only if $d_i$ equals 1. More precisely, it is just a dummy operation if $d_i$ equals 0 and does not contribute to the output computation. Thus, faulting line 3 will lead to a faulty ciphertext only if $d_i = 1$. Hence, if a fault can skip this operation, the attacker can, step by step, detect all the zeroes of the exponent $d$.

The fault has not to be necessarily an instruction skip. For instance, an attacker could modify a value during a RAM access. In the previous example, if the attacker can set to 0 the value of $d_i$ when it is read, one obtains the same result.

Thus, throughout this paper, we assume that the attacker can:

- precisely set the fault to a target operation ;
- skip an instruction or function call
- or set a variable to 0.

To keep our assumptions closed from practical fault attacks, we do not suppose that the attacker can set a variable to a chosen value different from zero.

We will see that in our PQC context, these assumptions are sufficient to let an attacker retrieve the positions of all the null coefficients in a secret key.

### B. Safe-error attack on lattice-based cryptography

The central operation in lattice-based cryptography is the polynomial multiplication. Such operation usually involves a secret polynomial with coefficients close to 0. Such operation must be performed in a timing-resistant manner to be secure against timing attacks. As described earlier, this context is usually a perfect fit for safe-error attacks.

Moreover, most of the lattice-based NIST finalist candidates use a centered binomial distribution or a uniform distribution in $\{-1, 0, 1\}$ ensuring that 0-coefficients are numerous. Hence, fault injections setting a secret coefficient to 0 at a precise operation, reveal information about secret 0-coefficients positions. Our study focuses on the NIST lattice-based Key Exchange Mechanisms (denoted KEM) and signatures. Then, the fault injection is performed during a decryption or a signature algorithm. Therefore, if the obtained plain text or signature is correct despite of a fault injection on a secret coefficient that ensures this coefficient is a 0.

Finding the positions of the zeroes may not be enough to recover the key, but it surely reduce the security of the scheme. In order to precisely measure the security loss, we need to use an estimation tool and compare our result with the claimed security.

### C. Security analysis of lattice-based cryptography

The NIST call for post-quantum safe cryptography has mentioned 5 security categories: 1 to 5. The categories $1/3/5$ corresponding to a claimed security equivalent, in terms of computational capabilities, of key search on a block cipher AES-128/192/256. The categories $2/4$ corresponding to a claimed security equivalent, in terms of computational capabilities, of collision search on a $256/384$-hash functions as SHA3-256/384.

The main goal of these categories is to facilitate security comparison between all the PQC proposals. However, some candidates under-estimate and others over-estimate these security categories. Then, the security comparison between these algorithms is an arduous task. In the following, we refer to these categories only to estimate how proposals securities downgrade due to safe-error attacks.

### D. Security Estimation loss

In the following, the attacker performs SEA during the execution of some lattice-based KEM or signatures to learn some information about the secret key. To estimate the security loss, we use the toolkit introduced in [17]. This tool is a security estimation framework for lattice-based schemes under lattice reduction attack when an attacker obtains some "*hints*" about the secret key. The "*hints*" can be obtained from side-channel attacks or, in our case, from fault injection.

Let $s$ be a secret vector and $v, l, k$ be parameters known by the attacker. This tools can handle 4 types of hints:

- Perfect hints: $\langle s, v\rangle = l$.
- Modular hints: $\langle s, v\rangle = l \mod k$.
- Approximate hints: $\langle s, v\rangle = l + e$, where $e$ is an error following a known distribution.
- Short vector hints: $v \in \Lambda$, where $\Lambda$ is a lattice related to the secret $s$.

These hints can be used to estimate cryptosystems security relying on LWE, LWR and NTRU problems. The security estimation is done using a unit called *bikz*, denoted by $\beta$. This unit corresponding to the BKZ-$\beta$ used to solve the Distorted Bounded Distance Decoding (DBDD) instance associated to the secret. As mentioned in the toolkit paper [17], there is no *bikz*-to-bit exact conversion. Then, in the following, security estimations are done in terms of *bikz* $\beta$ or NIST security categories.

To estimate the cryptosystems security, this tool requires to instantiate a DBDD instance related to our secret vector. Hence, some parameters are required: the secret's length and distribution $(n, \mathcal{D}_\sigma)$, the error's length and distribution $(m, \mathcal{D}_{\sigma_e})$, and the modulo $q$. Our script using this toolkit is available at [18].

In Sect. III, we present an attack description as well as the required parameters to instantiate the tool. We use it to provide a security loss estimation for each attacked cryptosystems.

## III. Application on Post-Quantum Cryptography

### A. NTRU

NTRU [19] is a KEM based on the eponymous *Nth degree Truncated Polynomial Ring Units* problem which is assimilated to a lattice-based one. The secret polynomial $f$ has coefficients belonging to $\{-1, 0, 1\}$ uniformly distributed and the public polynomial $h = (3gf_q) \mod (q, \phi_1\phi_n)$ has coefficients modulo $q$. The polynomials manipulated in NTRU are defined in $R'_q = \mathbb{Z}_q/(x^n - 1)$ with $n \in \{509, 677, 821, 701\}$ and $q \in \{2048, 4096, 8192\}$ depending on the security level. In this proposal, we focus the decryption function:

NTRU Decryption*:*

```
Input: $((f, f_p, h_q), c)$
Output: $r, m$
 1: if $c \not\equiv 0 \mod (q, \phi_1)$ return$(0,0,1)$
 2: $a \leftarrow (c \cdot f) \mod (q, \phi_1\phi_n)$
 3: $m \leftarrow (a \cdot f_p) \mod (3, \phi_n)$
 4: $m' \leftarrow \text{Lift}(m)$
 5: $r \leftarrow ((c - m') \cdot h_q) \mod (q, \phi_n)$
 6: if $(r, m) \in \mathcal{L}_r \times \mathcal{L}_m$ then
 7:     return $(r, m, 0)$
 8: else
 9:     return $(0, 0, 1)$
10: end if
```

*SEA application:* Our objective is to learn the zero coefficients positions of $f$ during the computation at line 2: $a \leftarrow (c \cdot f) \mod (q, \phi_1\phi_n)$. In the reference implementation, the secret coefficients are stored in a 16-bit integer (`uint16_t`) and the polynomial multiplication is done with the algorithm described hereafter

NTRU Polynomial multiplication*:*

```
Input: $a, c, f$ {all $a[i], c[i], f[i]$ are uint16_t}
Output: $a$
 1: for $k = 0$ to $n$ do
 2:     $a[k] \leftarrow 0$
 3:     for $i = 1$ to $n$ do
 4:         $a[k] \leftarrow a[k] + c[k+i] \times f[n-i]$
 5:     end for
 6:     for $i = 0$ to $k+1$ do
 7:         $a[k] \leftarrow a[k] + c[k-i] \times f[i]$
 8:     end for
 9: end for
10: return $a$
```

The fault injection is performed during the computation $a[k] \leftarrow a[k] + c[k+i] \times f[n-i]$. If the decryption succeeds then the coefficient $n - i$ is equal to 0. This attack is performed during several decryption procedures to recover all the coefficients of $f$ that are equal to 0.

*Security impact:* Using the tool introduced in Section II we can determine the security impact of this knowledge.

The column "Classical" determines the security level in terms of "bikz $\beta$" of the NTRU cryptosystem without any knowledge. The "Classical" NTRU DBDD instance is done with the following parameters:

- The length of $f$: $n$
- The distribution of $f$: coefficients uniform over $\{-1, 0, 1\}$
- The distribution for $g$ (see NTRU paper [19])

The column "Attacked" determines the security level of NTRU with the knowledge of $n/3$ secret coefficients. Indeed, we suppose that the secret coefficients are well distributed, then $f$ has $n/3$ coefficients equal to 0. The "Attacked" NTRU DBDD instance is done with the following parameters:

- The secret vector length without the known coefficients: $n = n - n/3$.
- A uniform distribution over two elements.
- The same distribution for $g$.

| | Classical | Attacked |
|---|---|---|
| NTRU HPS 1<br>$n = 509, q = 2048$ | Dim = 1018<br>$\beta = 172.15$ | Dim = 680<br>$\beta = 95.53$ |
| NTRU HPS 2<br>$n = 677, q = 2048$ | Dim = 1354<br>$\beta = 249.95$ | Dim = 904<br>$\beta = 146.20$ |
| NTRU HPS 3<br>$n = 821q, q = 4096$ | Dim = 1642<br>$\beta = 308.42$ | Dim = 1096<br>$\beta = 183.35$ |
| NTRU HRSS<br>$n = 701, q = 8192$ | Dim = 1402<br>$\beta = 236.30$ | Dim = 936<br>$\beta = 135.96$ |

In average the safe-error attack provides, in terms of *bikz*, a 42% security loss. The NTRU specification [19] mentioned that the NTRU HPS 1 security is lower than an actual AES-128 security. However, for all parameters the SEA brings the bikz security under NTRU HPS 1 original security. Therefore, with this attack an attacker can reduce the security of NTRU cryptosystem under $2^{128}$ security bits in a pre-quantum world and under $2^{64}$ security bits in a quantum world.

### B. Saber

Saber [20] is a MLWR lattice-based KEM. The secret vector $S = (s_1, \ldots, s_k)$, where $s_i$'s are polynomials following a centered binomial distribution $\mathcal{D}_\sigma$ in $R_q = \mathbb{Z}_q/(x^n + 1)$.

Depending on the security level, the Saber parameters are: $q = 2^{13}, p = 2^{10}, n = 256, \sigma \in \{5, 4, 3\}$ and $k \in \{2, 3, 4\}$. We focus the decryption function for applying SEA.

SABER DECRYPTION:

```
Input: $S, c = (c_m, b')$
Output: $m'$
1: $v = b'^T \cdot (S \mod p) \in R_p$
2: $m' = ((v - 2^{\epsilon_p - \epsilon_T} c_m + h_2) \mod p) >> (\epsilon_p - 1) \in R_2$
3: return $m'$
```

*SEA application:* Our objective is to determine the positions of the zero coefficients in $s$. One way is to perform a fault injection during the computation in line 1: $v = b'^T \cdot (S \mod p) \in R_p$. However, as the polynomial multiplication is computed with a 4 way Toom-cook algorithm, the secret coefficients are combined between each other. This makes the attack difficult in practice.

To overcome this issue, the attack can be done during the conversion of the secret input. Indeed, the coefficients of the secret polynomial are stored as a compact byte string and it is manipulated as a polynomial structure which encodes each secret coefficients to an `uint16_t`. Our attack focus on the byte-to-poly conversion.

SABER BYTE-TO-POLY FUNCTION:

```
Input: $(\texttt{uint8\_t} * sk, \texttt{uint16\_t} * S)$
Output: $S$
1: for $j = 0$ to $j < n/8$ do
2:     os_sk, os_s $= 13j,\ 8j$
3:     $S[\text{os\_s}] = sk[\text{os\_sk}] \mid ((sk[\text{os\_sk}+1] \& 0\text{x}1\text{F}) << 8)$
4:     $S[\text{os\_s} + 1] = \ldots$
5:     $\ldots$
6:     $S[\text{os\_s} + 7] = \ldots$
7: end for
8: return $s$
```

We present here, for the sake of clarity, only a sketch of the reference algorithm. The main idea is that each coefficient of $S$ is determined by multiple bytes from $sk$. Thus, the fault injection can be done during this transformation. If the decryption algorithm succeeded then, the faulted coefficient is equal to 0. As in NTRU, this attack is performed during several decryption procedure over different coefficients to recover all the 0-coefficients positions.

*Security impact:* Depending of the centered binomial distribution parameters $\sigma$, the secret vector has more or less 0. As previously, we suppose that our secret is well distributed. Hence, for the three possible values of $\sigma$ (5,4 or 3 respectively), the ratio of secret coefficients equal to 0 is $24,6\%$, $27,3\%$, and $31,25\%$ respectively.

The "Classical" LWR DBDD instance is done with the following parameters:

- The secret vector length: $n$,
- The secret vector centered binomial distribution: $\mathcal{D}_\sigma$,
- The length of the rounding: $m = n$.

The "Attacked" LWR DBDD instance is done with the following parameters:

- The secret vector length without the known coefficients: $n = n - an$, where $a$ is 0.246, 0.273 or 0.3125.
- The previous distribution without the 0-coefficients,
- The length of the rounding $m$ (unchanged).

The following table shows the security impact of such knowledge using the tool introduced in Section II.

| | Classical | Attacked |
|---|---|---|
| Light Saber<br>$n, m = 512, \sigma = 5$ | Dim = 1025<br>$\beta = 404.38$ | Dim = 900<br>$\beta = 292.05$ |
| Saber<br>$n, m = 768, \sigma = 4$ | Dim = 1537<br>$\beta = 648.72$ | Dim = 1328<br>$\beta = 462.57$ |
| Fire Saber<br>$n, m = 1024, \sigma = 3$ | Dim = 2049<br>$\beta = 892.21$ | Dim = 1729<br>$\beta = 613.26$ |

In average the safe-error attack provides, in terms of *bikz*, a $30\%$ security loss. The SABER specification [20] describes the security level in terms of AES-128/192/256. The following table present the specified security and the obtained one after SEA:

| | Security claimed | SEA security |
|---|---|---|
| Light Saber | AES-128 | $\leq$ AES-128 |
| Saber | AES-192 | $\approx$ AES-128 |
| Fire Saber | AES-256 | $\approx$ AES-192 |

*Remark 1:* The NTRU HPS 3 claimed an equivalent AES-192 security as Saber. However, the NTRU HPS 3 value of $\beta$ is 308.42 whereas Saber's value of $\beta$ is 648.72. This main difference is due to the secret coefficients distribution. Indeed, NTRU cryptosystem underestimates the theoretical security by lake of concrete attack, rather than Saber cryptosystem which overestimate it.

## C. Dilithium

Dilithium [21] is a lattice-based signature, based on the MLWE problem. The secret is given by $(S_1, S_2)$ two a vectors of $l$ polynomials. Their polynomials have coefficients following a centered binomial distribution $\mathcal{D}_\sigma$ in $R_q = \mathbb{Z}_q/(x^n + 1)$. Depending of the security level, $q = 8380417, n = 256, \sigma \in \{2, 4\}, (k, l) \in \{(4, 4), (6, 5), (8, 7)\}$

*SEA attack:* The polynomial multiplication is performed with the NTT algorithm. However, for compactness, the secret vectors are not stored in the NTT domain. One way to attack the secret coefficients, is to perform fault injection during the NTT computation. However as for Saber, it is easier to attack the secret coefficients during the unpacking process:

DILITHIUM UNPACK:

```
Input: $(\texttt{uint8\_t} * sk, \texttt{uint16\_t} * s, \eta)$
Output: $s$
1: for $j = 0$ to $j < n/8$ do
2:     $s[8\,i] = (sk[3\,i] >> 3)\ \&\ 7$
3:     $s[8\,i + 1] = (sk[3\,i] >> 3)\ \&\ 7$
```

```
4:      ...
5:      $s[8i+7] = (sk[3i+2] >> 5) \& 7$
6:      $s[8i] = \eta - s[8i]$
7:      $s[8i+1] = \eta - s[8i+1]$
8:      ...
9:      $s[8i+7] = \eta - s[8i+7]$
10:  end for
11:  return $s$
```

Here, the fault injection is performed as of line 6, depending of the targeted secret coefficient. For example a fault injection at line 7 ensures that the $8i+1$ coefficient is stored as a 0. The attack on the previous algorithm is similar to the one on Saber. The fault injection is repeated on each coefficient during several signature procedures. If the signature is not different than the original one, then the coefficient is 0. This attack is performed on $S_1$ and $S_2$.

*Security impact:* The number of 0 depends on the value $\sigma$. Hence, for $\sigma = 4$ (resp. $\sigma = 2$), $27,3\%$ (resp. $37,5\%$) of the secret coefficients of $S_1$ and $S_2$ are equal to 0. The following table shows the security impact of such knowledge using the tool introduced in Sect. II. The parameter $n$ (resp. $m$) corresponds to the number of coefficients in $S_1$ (resp. $S_2$). The "Classical" LWE DBDD instance is done with the following parameters:

- The length of $S_1$: $n$,
- The distribution of $S_1$: centered binomial distribution $\mathcal{D}_\sigma$.
- The length of $S_2$: $m$,
- The distribution of $S_2$: centered binomial distribution $\mathcal{D}_\sigma$ (same as $S_1$).

The "Attacked" LWE DBDD instance is done with the following parameters:

- The length of $S_1$ without the known coefficients: $n = n - an$. where a = 0.273 or 0.375.
- The distribution of $S_1$ and $S_2$ without the 0-coefficients.
- The length of $S_2$ without the known coefficients: $m = m - am$. where $a$ is 0.273 or 0.375.

The following table shows the security impact of such knowledge using the tool introduced in Section II.

| | Classical | Attacked |
|---|---|---|
| Dilithium 1 $(n,m) = (1024, 1024)$ $\sigma = 2$ | Dim = 2049 $\beta = 348.84$ | Dim = 1281 $\beta = 192.84$ |
| Dilithium 2 $(n,m) = (1280, 1536)$ $\sigma = 4$ | Dim = 2817 $\beta = 499.65$ | Dim = 2049 $\beta = 340.06$ |
| Dilithium 3 $(n,m) = (1792, 2048)$ $\sigma = 2$ | Dim = 3841 $\beta = 717.52$ | Dim = 2401 $\beta = 411.13$ |

Due to small centered binomial distributions, the security impact of SEA is consequent for Dilithium. The Dilithium specification [21] describes security level in terms of equivalence of SHA3-256/AES-192/AES-256. The following table presents the specified security and the one obtained after SEA:

| | Security claimed | SEA security |
|---|---|---|
| Dilithium 1 | SHA3-256 | $\leq$ SHA3-256 |
| Dilithium 2 | AES-192 | $\approx$ SHA3-256 |
| Dilithium 3 | AES-256 | $\leq$ AES-192 |

## D. Kyber

Kyber [22] is a MLWE lattice-based KEM. The secret vector $S = (s_1, \ldots, s_k)$, where, $s_i$ is a polynomial with coefficients following a centered binomial distribution $\mathcal{D}_\sigma$ in $R_q = \mathbb{Z}_q/(x^n + 1)$. In this proposal $q = 3329$, $n = 256$, $\sigma \in \{3, 2\}$ and $k \in \{2, 3, 4\}$ depending of the security level. Again we focus its decryption function:

KYBER DECRYPTION:

```
Input: $sk, c$
Output: Message $m$
1: $u \leftarrow \mathrm{Decompress}_q(\mathrm{Decode}(c), d_u)$
2: $v \leftarrow \mathrm{Decompress}_q(\mathrm{Decode}(c + \mathrm{offset}), d_u)$
3: $\hat{s} \leftarrow \mathrm{Decode}(sk)$
4: $\tilde{c} \leftarrow v - \mathrm{NTT}^{-1}(\hat{s}^T \cdot \mathrm{NTT(u)})$
5: $m \leftarrow \mathrm{Encode}(\mathrm{Compress}(\tilde{c}), 1)$
6: return $m$
```

However, in this scheme the secret coefficients of all the $s_i$'s are stored in the NTT domain. Then, the structure of the binomial distribution cannot be exploited to perform a stuck-at 0 during the computation of $\hat{s}^T \circ \mathrm{NTT(u)}$. In fact, the NTT domain ensures that the coefficients are values in $\{0, \ldots, q-1\}$ rather than $\{-\sigma, \ldots, \sigma\}$. Hence, by design, Kyber decryption algorithm is safe against our SEA at a cost of extra memory storage.

## IV. COUNTERMEASURES

The lattice-based cryptosystems are vulnerable to safe-error attacks in general. Indeed, most of their distribution are small and then the null coefficients are numerous and their positions thus provide important advantageous to an attacker.

One way to protect these schemes is to mask the distribution as in Kyber proposal. However, all the lattice-based algorithm are not compliant with the NTT domain. Another way would be to mask the secret coefficients by a uniform random in $\mathbb{Z}_q$. Then, the manipulated secret coefficients would be random in $\{0, \ldots, q\}$ rather than in $\{-\sigma, \ldots, \sigma\}$, where $\sigma$ is close to 0. The main drawbacks of masking are the extra storage for the coefficients, and the larger cost of the central operation: polynomial multiplication.

Another possible countermeasure would be the shuffling. Indeed, this countermeasure ensures, in theory, that the positions of the null coefficients would not leak. However this technique requires a significant amount of random generation.

## V. Conclusion

Our work is the first overall analysis of the resilience of NIST lattice-based finalist schemes against safe-error attacks. In this work, we determine the security impact of safe-error attacks against lattice-based cryptography. Due to small secret distributions, this kind of attack decreased significantly the theoretical security. However, without additional knowledge, finishing the attack to retrieve the entire secret key seems not practical. Nonetheless, this vulnerability must be taken into account to avoid devastating hybrid attack combining SEA and others side-channel leakage. Fortunately, classical countermeasures as masking or shuffling are possible to mitigate the safe-error attacks.

In this paper, our attacker model only consider fault injection setting variable to 0. It would be interesting to consider a stuck-at `FF` model since in this case the attacker sets a variable to $-1$, especially in the lattice-based schemes using a modulo equal to a power of two.

## References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[2] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology - CRYPTO '96*, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 104–113.

[3] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology - CRYPTO '99*, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397.

[4] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, M. Joye and J. Quisquater, Eds., vol. 3156. Springer, 2004, pp. 16–29.

[5] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *Advances in Cryptology - EUROCRYPT '97*, W. Fumy, Ed., vol. 1233. Springer, 1997, pp. 37–51.

[6] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology - CRYPTO '97*, B. S. K. Jr., Ed., vol. 1294. Springer, 1997, pp. 513–525.

[7] M. Joye and M. Tunstall, Eds., *Fault Analysis in Cryptography*, ser. Information Security and Cryptography. Springer, 2012.

[8] S. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," *IEEE Trans. Computers*, vol. 49, 2000.

[9] A. Berzati, C. Canovas-Dumas, and L. Goubin, "A survey of differential fault analysis against classical RSA implementations," in *Fault Analysis in Cryptography*, ser. Information Security and Cryptography, M. Joye and M. Tunstall, Eds. Springer, 2012.

[10] C. Clavier, "Attacking block ciphers," in *Fault Analysis in Cryptography*, ser. Information Security and Cryptography, M. Joye and M. Tunstall, Eds. Springer, 2012.

[11] P. Pessl and L. Prokop, "Fault attacks on cca-secure lattice kems," 2021, https://ia.cr/2021/064.

[12] A. A. Kamal and A. M. Youssef, "Fault analysis of the ntruencrypt cryptosystem," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 94-A, no. 4, pp. 1156–1158, 2011.

[13] ——, "Strengthening hardware implementations of ntruencrypt against fault analysis attacks," *J. Cryptogr. Eng.*, vol. 3, no. 4, pp. 227–240, 2013.

[14] F. Valencia, T. Oder, T. Güneysu, and F. Regazzoni, "Exploring the vulnerability of R-LWE encryption to fault attacks," in *Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems, CS2 2018*, J. Goodacre, M. Luján, G. Agosta, A. Barenghi, I. Koren, and G. Pelosi, Eds. ACM, 2018, pp. 7–12.

[15] F. Valencia, I. Polian, and F. Regazzoni, "Fault sensitivity analysis of lattice-based post-quantum cryptographic components," in *Embedded Computer Systems: Architectures, Modeling, and Simulation SAMOS 2019*, D. N. Pnevmatikatos, M. Pelcat, and M. Jung, Eds., vol. 11733. Springer, 2019.

[16] N. Bindel, J. Buchmann, and J. Krämer, "Lattice-based signature schemes and their sensitivity to fault attacks," in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography FDTC*. IEEE Computer Society, 2016, pp. 63–77.

[17] D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi, "Lwe with side information: Attacks and concrete security estimation," in *Advances in Cryptology – CRYPTO 2020*, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 329–358.

[18] "SEA PQC Script," https://github.com/paper16FDTC2021/SEAPQC.

[19] C. Chen, O. Danba, J. Hoffstein, A. Hülsing, J. Rijneveld, J. M.Schank, P. Schwabe, W. Whyte, and Z. Zhang, "NTRU," 2020, available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-3-Submissions.

[20] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber," 2020, available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-3-Submissions.

[21] S. Bai, L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium," 2019, available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions.

[22] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber," 2020, available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-3-Submissions.