Lattice-Based Signature Schemes and their Sensitivity to Fault Attacks

Nina Bindel and Johannes Buchmann and Juliane Krämer
Technische Universität Darmstadt, Germany
Email: {nbindel, buchmann, jkraemer}@cdc.informatik.tu-darmstadt.de

Abstract Due to their high efficiency and their strong security properties, lattice-based cryptographic schemes seem to be a very promising post-quantum replacement for currently used public key cryptography. The security of lattice-based schemes has been deeply analyzed mathematically, whereas little effort has been spent on the analysis against implementation attacks.
In this paper, we start with the fault analysis of one of the most important cryptographic primitives: signature schemes. We investigate the vulnerability and resistance of the currently most efficient lattice-based signature schemes BLISS (CRYPTO 2013), ring-TESLA (AfricaCrypt 2016), and the GLP scheme (CHES 2012) and their implementations. We consider different kinds of (first-order) randomizing, zeroing, and skipping faults. For each of the signature schemes, we found at least six effective attacks. To increase the security of lattice-based signature schemes, we propose countermeasures for each of the respective attacks.

Keywords. lattice-based cryptography, signature scheme, fault attack, side channel analysis

I. Introduction

Since the invention of Shor's algorithm, which solves the discrete logarithm and the integer factorization problem in polynomial time using quantum computation, most of our daily used public key cryptography is on threat as soon as large enough quantum computers can be built. Due to the expectable development of large quantum computers in the near future, research on cryptographic constructions which are secure even in the presence of large quantum computers - called post-quantum cryptography - has seen a boost in recent years. This is also reflected in two current announcements by the National Security Agency (NSA) and the National Institute of Standards and Technology (NIST): in 2015, NSA advertised lattice-based cryptography over elliptic curve cryptography [24] and in 2016, NIST announced to start a standardization process for post-quantum cryptography [25]. These developments show that post-quantum cryptography is standing on the edge of being used in practical applications.

Lattice-based constructions promise to be a valuable post-quantum replacement for current public-key cryptography because of their broad applicability, their high efficiency, and their strong security properties. However, when novel cryptographic schemes are brought into practice, their mathematical security is not sufficient. Physical attacks which target cryptographic schemes while they are being executed also have to be considered to provide the desired level of security. For lattice-based cryptographic schemes, until now, little effort has been spent in analyzing their vulnerability against such attacks. While lately the research on side channel attacks has advanced [8], [27], there are no results on fault attacks against schemes over general lattices yet[1]. Hence, the natural question arises whether lattice-based primitives and their implementations are vulnerable against fault attacks.

1) Contribution: In this paper, we start to investigate the vulnerability of lattice-based cryptography towards fault attacks. We analyze signature schemes and their implementations and scrutinize whether certain schemes or instantiations thereof are more vulnerable than others.

We consider the signature schemes BLISS [11], ring-TESLA [1], and the GLP scheme [16], since these are the most promising ones with respect to efficiency, i.e., runtime, signature sizes, and key sizes. Concerning the faults that we consider, we focus on first-order fault attacks, regarding randomizing[2], skipping, and zeroing faults. We explore the reasons for the vulnerability and resistance, respectively, of the key generation, sign, and verification algorithms. Furthermore, we propose countermeasures for each of the developed attacks.

To reduce the number of necessary faults, we propose a hybrid approach of fault attacks and lattice analysis. In the three analyzed signature schemes, the secret is a polynomial $s \in \mathbb{Z}_q[x]/(x^n + 1)$ with small coefficients. The hybrid approach allows an attacker to determine not

[1]A few results for fault attacks on NTRU-lattices and -schemes exist, e.g., [18]. These cannot be transfered to signature schemes over non-NTRU lattices which are used nowadays.

[2]We do not analyze bit flips separately since they are either covered by randomization faults or in practice considered as unrealistic [15].

all, but only a necessary amount of the coefficients of the secret with the help of faults such that the remaining lattice problem can be solved mathematically. We generally analyze how many coefficients of the secret must be recovered by fault attacks in order to use the hybrid approach for BLISS, ring-TESLA, and the GLP scheme. We apply this approach on our randomization fault attack.

Our research shows that all examined signature schemes and their implementations are vulnerable to all three kinds of considered fault attacks, since we find effective attacks against each of them. A summary of the analyzed attacks and the respective vulnerabilities of the three signature schemes is given in Table I. Note that certain effects can be achieved with different kinds of fault attacks, e.g., some variables can be zeroed out both with a zeroing fault and a skipping fault. Such fault attacks are only listed once in Table I, but mentioned and explained in all relevant sections in the remainder of this paper.

TABLE I
COMPARISON OF THE GLP SCHEME, BLISS, AND RING-TESLA WITH RESPECT TO THEIR VULNERABILITY TO THE ATTACKS DESCRIBED IN THIS PAPER. THE TABLE SHOWS THE ALGORITHMS WHICH THE FAULT ATTACKS TARGET, I.E., KEY GENERATION (KG), SIGNATURE GENERATION (S), AND VERIFY (V), AND IF THE SCHEME IS VULNERABLE TO THE RESPECTIVE ATTACK, I.E., VULNERABLE ●, VULNERABLE WITH A HUGE NUMBER OF NEEDED FAULTS (●), NOT VULNERABLE ❍, NOT APPLICABLE -.

Fault Attack	Algorithm	GLP	BLISS	ring-TESLA	Section
Rand. of secret	S	●	●	❍	III-A
Rand. of error	S	❍	❍	❍	III-B
Rand. of modulus	S	❍	❍	❍	III-C
Rand. of randomness	S	❍	❍	❍	III-D
Skip of mod-reduction	KG	❍	-	❍	IV-A1
Skip of addition	KG	●	●	●	IV-A2
Skip of rejection	S	(●)	(●)	(●)	IV-B1
Skip of addition	S	●	❍	❍	IV-B2
Skip of mod-reduction	S	❍	-	❍	IV-B3
Skip of correct-check	V	●	●	●	IV-C1
Skip of size-check	V	●	●	❍	IV-C2
Zero. of secret	KG	●	-	❍	V-A
Zero. of randomness	S	●	●	●	V-B
Zero. of hash value	S	❍	❍	❍	V-C
Zero. of hash polynomial	V	●	●	●	V-D

We show that two commonly used tweaks in the construction of lattice-based signature schemes, which are deployed for efficiency reasons, should only be carefully applied. First, our analysis shows a difference between the vulnerability of ideal-lattice-based schemes and standard-lattice-based schemes with respect to fault attacks. In standard-lattice-based schemes the underlying lattice is defined by a uniformly sampled matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, whereas ideal lattices can be defined via a polynomial of degree n. The resulting additional structure of ideal lattices leads to much smaller key sizes and better runtimes. Up to know, mathematical cryptanalysis could not exploit the additional cyclic structure of ideal lattices. However, based on our results, we show that ideal-lattice-based schemes are indeed more vulnerable to zeroing attacks than schemes over standard lattice. Secondly, we show that instantiating a scheme with common (ideal-) lattice problems, i.e., choosing the secret polynomial to be Gaussian distributed, is less vulnerable than more efficient instantiations. For example, the GLP scheme and BLISS[3] are more vulnerable to randomizing faults since the coefficients of the secret polynomial are chosen to be in $\{-1, 0, 1\}$ or $\{-2, ..., 3\}$, respectively, instead of Gaussian distributed as it is done for ring-TESLA. This has already been feared from a theoretical perspective [26] and is now shown in this work from the practical point of view.

We expect that our fault analysis and the proposed countermeasures are not limited to signature schemes, but that they can easily be transfered to most of the existing lattice-based constructions.

2) Organization: In Sec. II, we introduce the analyzed signature schemes, i.e., GLP, BLISS, and ring-TESLA, and explain the hybrid approach for combined fault attacks and lattice analyses. In Sec. III, we present randomizing attacks on all three schemes. Skipping attacks for the same schemes are explained in Sec. IV and in Sec. V, their vulnerability towards zeroing faults is investigated. We present countermeasures against the attacks and guidelines for fault-attack-resistant implementations in Sec. VI and conclude in Sec. VII.

II. PRELIMINARIES

A. Notation

Let $k \in \mathbb{N}$ and $n = 2^k \in \mathbb{N}$ throughout this paper. We define $q \in \mathbb{N}$ to be a prime with $q = 1 \pmod{2n}$. We denote by $\mathbb{Z}_q$ the finite field $\mathbb{Z}/q\mathbb{Z}$ with representatives in $[-q/2, q/2] \cap \mathbb{Z}$. We write $\pmod{q}$ to denote the unique representative in $\mathbb{Z}_q$. Furthermore, we define the rings $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$ and $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ and the sets $\mathcal{R}_{q,[B]} = \{\sum_{i=0}^{n-1} a_i x^i \mid a_i \in [-B, B] \cap \mathbb{Z}\}$ for $B \in [0, q/2] \cap \mathbb{Z}$ and $\mathbb{B}_{n,\omega} = \left\{a = \sum_{i=0}^{n-1} a_i x^i \mid a_i \in \{0,1\}, \|a\|^2 = \omega\right\}$ for $\omega \in [0, n] \cap \mathbb{Z}$. We indicate the Euclidean norm of a vector $\mathbf{v} \in \mathbb{R}^n$ by $\|\mathbf{v}\|$. Similarly, we define $\|a\| = \sqrt{a_0^2 + ... + a_{n-1}^2}$ for $a = \sum_{i=0}^{n-1} a_i x^i$. We denote polynomials by lower case letters (e.g., p) and (column) vectors by bold lower case letters (e.g., $\mathbf{v}$). We write matrices by bold upper case letters (e.g., $\mathbf{M}$) and the transpose of a matrix $\mathbf{M}$ by $\mathbf{M}^T$.

[3]We are aware of the fact that the publicly available implementation of BLISS is not to be used for security applications. However, it is one of the most efficient signature scheme implementations. By analyzing this software, the research community still can gain insights about how to implement schemes resistant against fault attacks.

Via the coefficient embedding, we identify a polynomial $a = a_0 + a_1x + ... + a_{n-1}x^{n-1} \in \mathcal{R}_q$ with its coefficient vector $\mathbf{a} = (a_0, \ldots, a_{n-1})^T$. Without further mentioning, we denote the coefficient vector of a polynomial $a \in \mathcal{R}_q$ by $\mathbf{a}$. We define $\mathsf{rot}(\mathbf{a}) = (-a_{n-1}, a_0, \ldots, a_{n-2})^T$ and $\mathsf{Rot}(a) = (\mathbf{a}, \mathsf{rot}(\mathbf{a}), \mathsf{rot}^2(\mathbf{a}), \ldots, \mathsf{rot}^{n-1}(\mathbf{a})) \in \mathbb{Z}_q^{n\times n}$. Polynomial multiplication of $a, b \in \mathcal{R}_q$ is equivalent to the matrix-vector multiplication $\mathsf{Rot}(a)\mathbf{b}$ in $\mathbb{Z}_q$. For values $a, b \in \mathbb{R}$, we write $a << b$ if a is much smaller than b. All logarithms are in base 2.

Let $d \in \mathbb{N}$, $c \in \mathbb{Z}$. We denote by $[c]_{2^d}$ the unique representative of c modulo 2^d in $(-2^{d-1}, 2^{d-1}] \cap \mathbb{Z}$. Let $\lfloor \cdot \rceil_d$ be the rounding operator $\lfloor \cdot \rceil_d : \mathbb{Z} \to \mathbb{Z}, c \mapsto (c - [c]_{2^d})/2^d$. We naturally extend these definitions to vectors and polynomials by applying $\lfloor \cdot \rceil_d$ and $[\cdot]_{2^d}$ to each component of the vector and to each coefficient of the polynomial, respectively. We abbreviate $\lfloor v \pmod q \rceil_d$ by $\lfloor v \rceil_{d,q}$.

Let $\sigma \in \mathbb{R}_{>0}$. Let $\mathcal{D}_\sigma$ be the discrete Gaussian distribution on $\mathbb{Z}$ with standard deviation σ. We denote by $d \leftarrow \mathcal{D}_\sigma$ the operation of sampling an element d with distribution $\mathcal{D}_\sigma$. When writing $v \leftarrow \mathcal{D}_\sigma^n$ we mean sampling each coefficient of a polynomial v Gaussian distributed. For a finite set S, we write $s \leftarrow_\$ S$ to indicate that an element s is sampled uniformly at random from S.

Let $n \geq k > 0$. A k-dimensional lattice Λ is a discrete additive subgroup of $\mathbb{R}^n$ containing all integer linear combinations of k linearly independent vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\} = \mathbf{B}$, i.e., $\Lambda = \Lambda(\mathbf{B}) = \{\mathbf{Bx} \mid \mathbf{x} \in \mathbb{Z}^k\}$. The determinant of a lattice is defined by $\det(\Lambda(\mathbf{B})) = \sqrt{\det(\mathbf{B}^\top \mathbf{B})}$. The basis is not unique for a lattice and moreover, the determinant of a lattice is independent of the basis. Throughout this paper we are mostly concerned with q-ary lattices. $\Lambda \in \mathbb{Z}^n$ is called a q-ary lattice if $q\mathbb{Z} \subset \Lambda$ for some $q \in \mathbb{Z}$. Let $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{m\times n}$. We define the q-ary lattices $\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^n \mid \mathbf{Ax} = \mathbf{0} \pmod q\}$ and $\Lambda_q(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^n \mid \exists \mathbf{s} \in \mathbb{Z}^m \text{ s.t. } \mathbf{x} = \mathbf{A}^\top \mathbf{s} \pmod q\}$.

B. Description of the Lattice-Based Signature Schemes

In this subsection, we recall the signature schemes that we analyze in this work. We describe the GLP scheme by Güneysu et al. [16], ring-TESLA by Akleylek et al. [1], and BLISS by Ducas et al. [11]. We depict the three schemes in Appendix A in Fig. 3, 4, and 5, respectively. We follow the notation of the respective figures.

The security of the signature schemes is based on the ring learning with errors (R-LWE), the ring short integer solution (R-SIS), or the decisional compact knapsack (DCK) problem. We give a formal definition of R-LWE in Appendix B. For a formal definition of DCK and R-SIS we refer to the original works.

1) GLP: The secret key consist of two polynomials $s, e \leftarrow_\$ \mathcal{R}_{q,[1]}$ with ternary coefficients, i.e., with coefficients in $\{-1, 0, 1\}$; the public key is a tuple of $a \leftarrow_\$ \mathcal{R}_q$ and $b = as + e \pmod q$. On input μ, the sign algorithm first samples $y_1, y_2 \leftarrow_\$ \mathcal{R}_{q,[k]}$. Afterwards, it hashes the most significant bits of $ay_1 + y_2$ together with μ. The signature polynomials z_1 and z_2 are computed. To hide the secret, rejection sampling is applied, i.e., z_2 is compressed to $z_2^\star$ and the signature is returned only with some probability (see [20] for further information on the rejection sampling). The verification algorithm checks the size of z_1 and $z_2^\star$ and the equality of c and $H\left(\lfloor az_1 + z_2^\star - bc \rceil_{d,q}, \mu\right)$. Güneysu et al. [16] state the parameter set GLP-Set-I with $n = 512$ and $q = 8383489$. This instantiation of the DCK gives a hardness of at least 80 bit [11].

2) ring-TESLA: The secret key sk is a tuple of three polynomials s, e_1, and e_2 with small coefficients; the public key vk consists of the polynomials $a_1, a_2 \leftarrow_\$ \mathcal{R}_q$, $b_1 = a_1 s + e_1 \pmod q$, and $b_2 = a_2 s + e_2 \pmod q$. During signing a message μ, a random polynomial $y \leftarrow_\$ \mathcal{R}_{q,[B]}$ is sampled. Afterwards, the hash value c' of the most significant bits of the products $a_1 y$ and $a_2 y$ and μ is computed and encoded as the polynomial $c \in \mathbb{B}_{n,\omega}$. The signature σ of μ consists of c' and the polynomial $z = y + sc$. Before returning the signature, rejection sampling is applied. For verification of the signature (c', z), the size of z and the equality of c' and $H(\lfloor a_1 z - b_1 c \rceil_{d,q}, \lfloor a_2 z - b_2 c \rceil_{d,q}, \mu)$ is checked, where c is again the encoded polynomial of c'. Akleylek et al. [1] proposed the parameter set ring-TESLA-II to achieve 128-bit hardness from the underlying R-LWE problem: $n = 512$, $\sigma = 52$, $q = 39960577$.

3) BLISS: The key pair is chosen NTRU-like, i.e., the public key is $\mathsf{vk} = (a_1, a_2) = \left(2\frac{2g+1}{f} \pmod q, q - 2\right)$, where $f, g \leftarrow_\$ F_{d_1,d_2} = \{\sum_{i=0}^{n-1} h_i x^i | h_i \in \{-2, -1, 0, 1, 2\}, \ |\{h_i = \pm 1\}| = d_1, \ |\{h_i = \pm 2\}| = d_2\}$ and f is invertible modulo q. The secret key sk consists of $\mathsf{sk} = (s_1, s_2)^T = (f, 2g + 1)^T$. Furthermore, the vectors (a_1, a_2), $(s_1, s_2)^T$, and $\xi \in \mathbb{Z}$ are chosen such that $(a_1, a_2)(s_1, s_2)^T = q = -q \pmod{2q}$, $\xi(q - 2) = 1 \pmod{2q}$, and hence $\xi(a_1, a_2) = (\xi a_1, 1) \pmod{2q}$. To sign a message μ, random vectors y_1 and y_2 are sampled with Gaussian distribution. A hash value c is computed from the randomness, the public key, ξ, and the message μ. Afterwards, the value $b \leftarrow_\$ \{0, 1\}$ is chosen, the polynomials $z_1 = y_1 + (-1)^b s_1 c$ and $z_2 = y_2 + (-1)^b s_2 c$ are computed, rejection sampling is applied, and z_2 is compressed to $z_2^\star$. During verification of the signature $(z_1, z_2^\star, c)$, the sizes of z_1 and $z_2^\star$ and the equality of c and $H\left(\lfloor \xi a_1 z_1 + \xi q c (\bmod\ 2q) \rceil_{d,2q} + z_2^\star (\bmod\ p), \mu\right)$ are checked. Ducas et al. [11] give two parameter sets to achieve 128-bit hardness of the underlying problem: BLISS-I with $n = 512$, $\sigma = 215$, $q = 12289$ and BLISS-II with $n = 512$, $\sigma = 107$, $q = 12289$. Furthermore, we like to emphasize that in the instantiations BLISS-I and

BLISS-II, $d_2 = 0$. Hence, it holds true that

$$s_{j,i} \in \begin{cases} \{-1,0,1\} & \text{if } j = 1, \\ \{-1,1,3\} & \text{if } j = 2, i = 0, \\ \{-2,0,2\} & \text{if } j = 2, i \in \{1,...,n-1\}, \end{cases}$$

where $s_1 = \sum_{i=0}^{n} s_{1,i}x^i$ and $s_2 = \sum_{i=0}^{n} s_{2,i}x^i$.

C. Description of the Hybrid Approach of Lattice Analysis and Fault Attacks

In this section, we describe how to combine fault attacks and algorithms that solve lattice problems such as LWE or SIS. Via this combination, which we call hybrid approach, we can reduce the number of faults necessary to receive the secret drastically. Revealing all coefficients of the secret of a lattice problem with high dimension might require a huge amount of fault attacks. Instead, we analyze that it is sufficient to reveal just enough coefficients with the help of faults to solve the remaining instance with algorithms that solve lattice problems, e.g., the embedding approach. We describe our hybrid approach for the LWE problem next. We describe the analysis for SIS in the full version of this paper [6].

Let $\mathbf{As} + \mathbf{e} = \mathbf{b} \pmod q$ be an LWE instance, with $\mathbf{A} \in \mathbb{Z}_q^{m\times n}, \mathbf{s} \in \mathbb{Z}_q^n$, and $\mathbf{e} \in \mathbb{Z}_q^m$. Assume that k coefficients of the secret $\mathbf{s}$ are known. W.l.o.g., we can assume that the first k coefficients of $\mathbf{s}$ are known, since the samples of an LWE instance can be reordered. Then, this instance can be written as

$$(\mathbf{A_1}|\mathbf{A_2})\left(\mathbf{s_1}, \mathbf{s_2}\right)^T + \mathbf{e} = \mathbf{A_1 s_1} + \mathbf{A_2 s_2} + \mathbf{e} = \mathbf{b},$$

with $\mathbf{A_1} \in \mathbb{Z}_q^{m\times k}$, $\mathbf{A_2} \in \mathbb{Z}_q^{m\times(n-k)}$, and $\mathrm{s}_1 \in \mathbb{Z}_q^k, \mathrm{s}_2 \in \mathbb{Z}_q^{n-k}$. Let $\mathbf{b}' = \mathbf{b} - \mathbf{A_1 s_1}$. Thus, $\mathbf{A_2 s_2} + \mathbf{e} = \mathbf{b}' \pmod q$ defines an LWE instance with the same number m of samples but with a decreased dimension $n-k$ of the secret vector.

To compute the minimal value of k, we first choose the time T (in seconds) how long the LWE solver should run, e.g., one day $T = 86400$. Afterwards, we compute the corresponding Hermite delta by the estimation made by Linder and Peikert [19]

$$\log_2(\delta(T)) = \frac{1.8}{\log_2(T) + 110}. \quad (1)$$

The Hermite delta is a measurement for the quality of a basis reduction, for more information we refer to [19]. Given n, m, and δ and following the embedding approach proposed in [14], we can compute the value k. We emphasize that we are aware that the embedding approach is not always the best attack to solve LWE. Nevertheless, it yields an upper bound on the number of fault attacks needed.

In the embedding approach the LWE instance is reduced to an instance of the unique shortest vector problem (uSVP) [3], [4]. To this end, an embedding lattice Λ is defined in which the error vector $\mathbf{e}$ is embedded. Following the explanation by Dagdelen et al. [31], we know that a short vector can be found if

$$\delta^{dim(\Lambda)} \leq \frac{\Gamma(1 + \frac{dim(\Lambda)}{2})^{\frac{1}{dim(\Lambda)}}}{||\mathbf{e}||\tau\sqrt{\pi}} det(\Lambda)^{\frac{1}{dim(\Lambda)}}, \quad (2)$$

where $\tau \approx 0.4$ is constant and $\Gamma(n)$ is the gamma function.

Two different ways to define Λ were proposed, which are described next. During the standard embedding approach we apply a uSVP solver on the lattice $\Lambda = \Lambda_q\left(\mathbf{A_{st}}\right)$ with $\mathbf{A_{st}} = \begin{pmatrix} \mathbf{A_2} & \mathbf{b}' \\ 0 & 1 \end{pmatrix} \in \mathbb{Z}_q^{(m+1)\times(n-k+1)}$. Hence, $dim(\Lambda_q(\mathbf{A_{st}})) = m+1$ and $det(\Lambda_q(\mathbf{A_{st}})) = q^{m-n+k}$. Thus, Equation (2) gives the following inequality:

$$\delta^{m+1} \leq \frac{\Gamma(1 + \frac{m+1}{2})^{\frac{1}{(m+1)}}}{\sqrt{\pi}||e|| \cdot \tau} \cdot q^{k-n-1}. \quad (3)$$

During the dual embedding approach we apply a uSVP solver on the lattice $\Lambda = \Lambda_q^{\perp}(\mathbf{A_D})$ with $\mathbf{A_D} = \left(\mathbf{A_2}|\mathbf{I_m}|\mathbf{b}'\right) \in \mathbb{Z}_q^{m\times(n-k+m+1)}$. Hence, $dim(\Lambda_q^{\perp}(\mathbf{A_D})) = n-k+m+1$ and $det(\Lambda_q^{\perp}(\mathbf{A_D})) = q^m$. Thus, Equation (2) gives

$$\delta^{n-k+m+1} \leq \frac{\Gamma(1 + \frac{n-k+m+1}{2})^{\frac{1}{(n-k+m+1)}}}{\tau \cdot \sqrt{\pi}||\mathbf{e}||} \cdot q^{k-n-1}. \quad (4)$$

Assume that the computations should not run longer than a day (resp., a week). By Equation (1) this corresponds to the Hermite delta $\delta_1 = 1.0099$ (resp., $\delta_2 = 1.0097$). Finally, given n, m, and δ, we can compute the minimal value for k such that Equation (3) or Equation (4) is fulfilled.

Applying the hybrid approach to BLISS, ring-TESLA, and the GLP scheme with δ_1 shows that it is sufficient to reveal $k = 344$, $k = 405$, and $k = 118$, respectively, instead of all secret coefficients by fault attacks. Given δ_2 the minimal values are $k = 337$, $k = 389$, and $k = 105$, respectively. Note that the bit-security of the GLP scheme is 80-bit, whereas the proposed instantiations BLISS-I, BLISS-II, and ring-TESLA-II give 128 bit of security. We explain the derivation of k for each of the three schemes in detail in the full version of this paper [6].

III. Randomization Faults

A randomization fault randomly changes the value of a variable that is processed in the attacked alorithm, i.e., the attacker does not know the value of the variable after the attack, but benefits from knowing that it has been changed within a certain range. Depending on the attacker's abilities, the fault targets the whole variable or only some bytes or bits of it [29]. We analyze the effects of a randomization fault targeting the secret polynomial (Section III-A), the error polynomial (Section III-B), the modulus (Section III-C), and the randomness (Section III-D) during the signature generation.

A. Randomization of the Secret Polynomial

In 1996, Bao et al. introduced a method to attack signature schemes with binary secret keys [5]. In particular, they show how to attack RSA, the ElGamal scheme, and Schnorr signature schemes. In this section, we first describe how to adjust the attack from [5] to lattice-based Schnorr-like signature schemes instantiated with binary secret over standard lattices. Afterwards, we describe a more evolved attack on the GLP scheme.

Take $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \pmod{q}$ with $\mathbf{s} \in \{0,1\}^n$ and $\mathbf{e} \leftarrow \chi$, where χ is some error distribution over $\mathbb{Z}^n$. The public key is $(\mathbf{A}, \mathbf{b})$ and the secret key consists of $(\mathbf{s}, \mathbf{e})$. The signature of a message μ is computed as follows: choose randomness $\mathbf{y}$, compute the hash value $\mathbf{c} = H(\lfloor \mathbf{A}\mathbf{y} \rceil_{d,q}, \mu)$, compute $\mathbf{z} = \mathbf{s}\mathbf{c} + \mathbf{y}$, and return $\sigma = (\mathbf{z}, \mathbf{c})$.

Assume one coefficient of $\mathbf{s}$ is changed via a fault attack, i.e., the secret $\mathbf{s}' = (s_1, ..., s_{i-1}, s'_i, s_{i+1}, ..., s_n)^T$ is used to generate a signature. Hence, $\sigma' = (\mathbf{z}', \mathbf{c}) = (\mathbf{s}'\mathbf{c} + \mathbf{y}, \mathbf{c})$ is returned as faulty signature of μ. Now, an attacker checks whether $H\left(\lfloor \mathbf{A}\mathbf{z}' - \mathbf{b}\mathbf{c} - \mathbf{A}\mathbf{v}_{\mathbf{i},\alpha}\mathbf{c} \rceil_{d,q}, \mu\right) = \mathbf{c}$, where $\mathbf{v}_{\mathbf{i},\alpha}$ is the zero vector except that the i-th entry is equal to $\alpha \in \{-1, 0, 1\}$. Depending on the value of α and the index i, the attacker can determine the value of s_i:

$$\text{If } \alpha = \begin{cases} 0 & \text{then } s_i = s'_i, \text{ run attack again,} \\ 1 & \text{then } s_i = 1, \\ -1 & \text{then } s_i = 0. \end{cases}$$

Hence, in case of a successful fault attack, i.e., $s_i \neq s'_i$, an attacker finds out one coefficient of the secret for each injected fault. To our knowledge, there is no lattice-based signature scheme instantiated over binary LWE. However, recent results on the hardness of binary LWE [2], [10], [21] show an interest in this instantiation and also a lattice-based encryption scheme with binary secret was recently proposed [9]. With our description above we stand in line with those being cautious about instantiations of schemes with binary LWE.

1) Applying the Attack to the GLP Scheme: In this section we describe a generalization of the attack by Bao et al. [5] to ternary secret keys, i.e., to secret keys with coefficients in $\{-1, 0, 1\}$. We explain the attack by applying it to the GLP scheme, since its secret key is chosen to be ternary. Furthermore, we assume that the attack changes up to r consecutive coefficients instead of only a single coefficient of the secret. This is generally considered to be a more realistic scenario [15].

Assume that an attacker changes r consecutive coefficients of the secret s, i.e., $s' = s + \varepsilon$ with $\varepsilon = \sum_{i=j}^{j+r} \varepsilon_i x^i$, $0 \leq j < n$ where all $\varepsilon_i, s'_i \in \{-1, 0, 1\}$. The attack consists of three steps: inducing a randomization fault, querying a signature on some message, i.e., $\sigma' = (z'_1, z^\star_2, c) = (s'c + y, z^\star_2, c)$ with s' being the faulty secret, and analyzing the output by running a software implementation of the algorithm $\mathsf{GeneralBao}(\cdot)$ that is depicted in Fig. 1. The attacker repeats those three steps until sufficiently many coefficients of the secret are determined such that the hybrid approach described in Sec. II-C can be applied.

The algorithm $\mathsf{GeneralBao}(\cdot)$ gets as input the public key, a signature of a message μ, and two lists: the list *secret* where the determined coefficients of the secret are saved and the list *determined* where the information whether or not a coefficient is already determined is saved. The algorithm $\mathsf{GeneralBao}(\cdot)$ returns updated lists *secret* and *determined*.

Let α be the difference between the secret s and the faulty secret s', i.e., $\alpha = \sum_{i=0}^{n-1} \alpha_i x^i$ is a polynomial with $\alpha_i \in \{-2, -1, 0, 1, 2\}$. The attacker checks whether $H(\lfloor az_1 + z_2 - bc - a\alpha c \rceil_{d,q}, \mu) = c$ with $\alpha_i, ..., \alpha_{i+r} \in \{-2, -1, 0, 1, 2\}$ for $i \in \{0, ..., n-1-r\}$. Thereby, the attacker gains information about the value and index of s_i. The possible values for s_i, s'_i, and α_i are shown in Table II.

TABLE II
POSSIBLE COMBINATIONS FOR THE COEFFICIENTS OF s, s', AND α.

s'_i	0	0	0	1	1	1	-1	-1	-1
s_i	0	1	-1	0	1	-1	0	1	-1
$\alpha_i = s'_i - s_i$	0	-1	1	1	0	2	-1	-2	0

As indicated by Fig. 1, the procedure $\mathsf{GeneralBao}(\cdot)$ distinguishes between five different cases for each coefficient of α once the correct values of $\alpha_0, ..., \alpha_{n-1}$ are found.

$$\text{If } \alpha_i = \begin{cases} 2 & \text{then } s_i = -1, \\ -2 & \text{then } s_i = 1, \\ 1 & \text{then } s_i = 1 \text{ or } s_i = 0, \\ -1 & \text{then } s_i = -1 \text{ or } s_i = 0, \\ 0 & \text{then } s_i = s'_i. \end{cases}$$

In the latter cases, the attacker can not determine s_i uniquely. Let s_j be a coefficient which was changed during a fault attack such that $\alpha_{j,1} = \alpha_j = \pm 1$. Assume that s_j is changed again by another fault attack with difference $\alpha_{j,2}$. Then the attacker can determine s_j uniquely if $\alpha_{j,1} \neq \alpha_{j,2}$ and $\alpha_{j,2} \neq 0$. The list *determined* is used for exactly this purpose: to remember which coefficients were changed but could not be determined uniquely.

```
GeneralBao(·):
Input: $\sigma = (z_1, z_2^\star, c), \mu, \mathsf{vk} = (a, b)$, list $determined$, list $secret$;
signature $\sigma$ is computed with a faulty secret
Output: $determined, secret$
poly $\alpha$ = 0 #$\alpha = \sum_{i=0}^{n-1} \alpha_i x^i$
For $i \in \{0, ..., n-1\}$:
  For $\alpha_i \in \{0, -2, -1, 1, 2\}$:
    For $\alpha_{i+1} \in \{0, -2, -1, 1, 2\}$:
      ...
        For $\alpha_{i+r} \in \{0, -2, -1, 1, 2\}$:
          If $\left(H(\lfloor az_1 + z_2 - bc - a\alpha c\rceil_{d,q}, \mu) = c\right)$:
            For $j \in \{i, ..., i+r\}$:
              If $(\alpha_j = 2)$:
                $secret[j] = -1$, $determined[j] = 2$
              If $(\alpha_j = -2)$:
                $secret[j] = 1$, $determined[j] = 2$
              If $(\alpha_j = -1)$:
                If $(determined[j] = 1)$:
                  $secret[j] = 0$, $determined[j] = 2$
                Else:
                  $determined[j] = -1$
              If $(\alpha_j = 1)$:
                If $(determined[j] = -1)$:
                  $secret[j] = 0$, $determined[j] = 2$
                Else:
                  $determined[j] = 1$
Return $determined, secret$
```

Fig. 1. Algorithm to compute coefficients of the secret given a signature computed with a faulty secret where maximal r of the coefficients are changed by a randomization fault

As described in Sec. II-C, at most $k = 118$ coefficients of s have to be determined via fault attacks to compute the whole secret via the hybrid approach. Hence, next we analyze the expected number of faults that we have to induce to determine $k = 118$ coefficients of s. We assume that the index of the first of the r changed coefficients is chosen uniformly random in $\{0, ..., n-1\}$. Since $r << n$, we assume that the changed (and hence the determined) coefficients are uniformly distributed over all coefficients $s_0, ..., s_{n-1}$. Assume the j-th fault attack is induced after i_j coefficients have already been determined uniquely. Then the number of newly determined coefficients after the j-th fault attack is given by $\frac{2}{9} r \frac{512 - i_{j-1}}{n}$, since n is the number of coefficients of s and following Table II, the probability that a coefficient is changed such that it can be determined uniquely is $2/9$. Assume the fault attack targets one byte. Since each coefficient can be saved in two bits, this corresponds to four changed coefficients, i.e., $r = 4$. Hence, solving the following equation for m gives us the number of $m = 151$ (expected) needed faults to determine $k = 118$ coefficients of the secret s:

$$\sum_{j=1}^{m} \frac{2r}{9} \cdot \frac{512 - i_{j-1}}{512} \geq k,$$

with $i_0 = 0$. In case $r = 1$, the expected number of needed fault attacks to uniquely determine $k = 118$ coefficients of s is $m = 604$. Hence, the generalization to targeting more coefficients is not only more realistic, but also needs a much smaller number of fault attacks. On the other hand the runtime of the software to find the polynomial α is longer, since it takes $512 \cdot 5^r$ times the runtime of the hash query for every fault attack.

2) Application to BLISS: As described in Sec. II-B3, the secret keys of the instantiations BLISS-I and BLISS-II[4] are two polynomials s_1, s_2 with ternary-like instantiation. Therefore, the attack on the 128-bit instantiations of BLISS can be described in a similar manner as it was done for the GLP scheme.

As before, assume r coefficients are changed during a fault attack. We assume that the coefficients of the faulty secret polynomial(s) are in the set $\{-3, ..., 3\}$, since it can be assumed that each coefficient is saved in three bits. Let $\alpha = \sum_{i=0}^{n-1} \alpha_i x^i$. Given a faulty signature $\sigma = (z_1, z_2^\star, c)$ of the message μ, the attacker runs an algorithm similar to GeneralBao(·), Fig. 1. The only difference is that the values of α lie in different intervals: in case the fault was induced on s_1, the attacker checks $H\left(\lfloor \xi a_1 z_1 + \xi qc - \xi \alpha c\rceil_{d,2q} + z_2^\star, \mu\right) = c$ for $\alpha_{i_1}, ..., \alpha_{i_r} \in \{-4, ..., 4\}$ for $i_1, ..., i_r \in \{0, ..., n-1\}$. In case the fault was induced on s_2, the attacker checks $H\left(\lfloor \xi a_1 z_1 + \xi qc - \alpha c\rceil_{d,2q} + z_2^\star, \mu\right) = c$ for $\alpha_{i_1}, ..., \alpha_{i_r} \in \{-5, ..., 5\}$ for $i_1, ..., i_r \in \{0, ..., n-1\}$[5]. As in the attack against the GLP scheme, sometimes the secret coefficients can be determined uniquely. The probability that a coefficient of s_1 (resp., s_2) is determined uniquely is $2/21$ (resp., $4/21$). We assume again that the indices of the changed coefficients are distributed uniformly random over the original coefficients. Hence, we roughly estimate the expected number of needed fault attacks by

$$\sum_{j=1}^{m} \frac{r}{7} \cdot \frac{1024 - i_{j-1}}{1024} \geq k,$$

with $i_0 = 0$ and notations as in Sec. III-A1. Hence, with

[4]For the higher-security instantiations BLISS-III and BLISS-IV, $d_2 \in \{0.03, 0.06\}$, i.e., there are a very few coefficients outside $\{-1, 0, 1\}$. Hence, extending our attack to those instantiations will increase the run time, but the attack still weakens the security.

[5]To be exact, in case the fault was induced on s_2, the attacker checks the hash values for $\alpha_0 \in \{-4, ..., 6\}$ and $\alpha_1, ..., \alpha_{n-1} \in \{-5, ..., 5\}$.

$r = 1$ an expected number of $m = 2934$ faults is needed to determine $k = 344$ (see Sec. II-C) coefficients of the secret (with $r = 4$ an expected number of $m = 733$ is needed). We conclude that the attack is less efficient on BLISS than on the GLP scheme, but it is still applicable.

3) Application to ring-TESLA: The coefficients of the secret $s = \sum_{i=0}^{n-1} s_i x^i$ of ring-TESLA are chosen with Gaussian distributed. Hence, the possible values of α would be in a very large range. Even if we assume that the coefficients are with high probability bounded by $|s_i| \leq \sigma$ (with $\sigma = 52$ for instantiation ring-TESLA-I), the number of needed fault attacks would be huge. Hence, this attack does not seem to be a threat for ring-TESLA in particular and for instantiations with Gaussian distribution in general.

B. Randomization of the Error Polynomial

A similar attack as described in Sec. III-A could be used to compute the (secret) error polynomial e. Leaking e is as bad as leaking s, since s can be computed easily once e is known. Moreover, the success probability of most of the mathematical lattice analyses would increase since those algorithms also benefit from reducing the key space from $\{-1, 0, 1\}$ to $\{-1, 0\}$ or $\{0, 1\}$. However, the GLP scheme is not vulnerable to this variant because of its compression and rounding functions, since then the equation $H(\lfloor az_1 + z_2 - bc - a\alpha x^i c\rceil_{d,q}, \mu) = c$ holds for several values of α. Hence, α can not be determined. Nevertheless, we mention this attack to raise awareness during the construction and instantiation of schemes. For example, instantiating ring-TESLA (which does not come with such a compression function) over ternary LWE should be considered very carefully.

C. Randomization of the Modulus

The randomization of the modulus does not seem to reveal any information that helps to forge signatures, because the value of the faulty modulus would remain unknown. Furthermore, key and signature generation of lattice-based signature schemes are randomized at several points by construction. Hence, tools like the Chinese remainder theorem do not give access to the secret.

D. Randomization of the Randomness

As expected, also the randomization of the random values, e.g., the product ay (or similar) or the hash output c, does not reveal information that the attacker can use to forge signatures, because such values look like (or are) random values by default.

IV. Skipping Faults

Skipping faults consist in skipping, i.e., ignoring, selected lines of the program code. This can, for example, be achieved via CPU clock glitching [7]. By showing that even higher-order skipping faults against a real-word cryptographic implementation are practical, the relevance of this kind of fault attacks has recently been strengthened [7].

We analyze different ways to exploit skipping faults during the key generation (Section IV-A), the signature generation (Section IV-B), and the verification algorithm (Section IV-C). In the majority of cases, we explain the fault attacks using the implementations of the signature schemes. Therefore, we use the publicly available software, i.e., we use the C++-implementation of BLISS [12], the C-implementation of the GLP scheme [17], and the C-implementation of ring-TESLA [1]. Since the implementation of ring-TESLA and the GLP scheme use the benefits of the AVX instructions, we sometimes also consider the X86 assembly code of respective code lines.

A. During Key Generation

In this section we describe two possible skipping attacks during the key generation.

1) Skipping the Modulus Reduction: Let $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{n\times m}$, and $\mathbf{s}, \mathbf{e}$ are chosen with some distribution over $\mathbb{Z}^n$ and $\mathbb{Z}^m$, respectively. Afterwards compute $\mathbf{b} = \mathbf{A}\mathbf{s}+\mathbf{e}$ without reducing modulo q. Solving SVP or CVP in the lattice $\Lambda = \{\mathbf{v} \in \mathbb{Z}^n \mid Aw = v \text{ for some } w \in \mathbb{Z}^m\}$ is most often much easier than solving the same problem in $\Lambda_q(\mathbf{A})$, since $\det(\Lambda) \geq \det(\Lambda_q(\mathbf{A}))$, especially in case $\mathbf{A}$ is invertible. Hence, skipping the modulo operation during the key generation algorithm seems to be a security flaw. However, this fault attack is already prevented in the three considered signature schemes. We use the implementation of the GLP scheme to explain the prevention.

As indicated by Listing 1, the value `t` (corresponding to b in our notation) is computed without the reduction step. The modulo operation is performed in the subroutine `poly_pack`. Skipping Line 78 of Listing 1 thwarts the modulo reduction. Afterwards, only the least 32 bits of (the faulty) `t` are saved in `r`. Hence, skipping Line 78 leads to a randomization fault on b which does not reveal secret information.

```
48   poly_mul_a(t, s1);
49   poly_add_nored(t, t, s2);
51   poly_pack(pk, t);
     [...]
71   void poly_pack(unsigned char r[3*POLY_DEG],
    const poly f)
72   {
73     int i;
74     signed long long t;
75     for(i=0;i<POLY_DEG;i++)
76     {
77       t = (unsigned long long)f[i];
78       t = ((t % PARAM_P) + PARAM_P) % PARAM_P;
79       r[3*i+0] = t          & 0xff;
80       r[3*i+1] = (t >>  8) & 0xff;
81       r[3*i+2] = (t >> 16) & 0xff;
82     }
83   }
```

Listing 1. C code of the GLP scheme for the computation of the public value $b = as + e$ in the subroutine `crypto_sign_keypair` and of the modulus operation and compression in the subroutine `poly_pack`; the value `t` corresponds to the value b, the value `s1` corresponds to s, and `s2` corresponds to e in our notation.

2) Skipping the Addition: We explain the following attack using examples from the C-implementation of the GLP scheme. However, the attack can also be successfully applied to ring-TESLA and BLISS. Details about the attacks against ring-TESLA or BLISS can be found in the full version of this paper [6].

In the GLP implementation, the public key is computed as follows, see Listing 1: first a and s are multiplied and saved in the value `b` (Line 48). Afterwards, the error e is added to `b` (Line 49). Hence, skipping the second operation yields $b = as$ and an attacker can easily recover s by Gaussian reduction. Note that skipping Line 48 in Listing 1 results in an unallocated variable `b`, triggering a segmentation fault. Hence, no (predictable) information is returned which could be used by an attacker.

Considering the assembly code[6] of the addition, see Listing 2, one can see that in Line 962 the command `poly_add_nored@PLT` is called. Hence, skipping this line results in $b = as$ as described above.

```
958                      .loc 1 49 0
959  1254 4C89E2         movq   %r12, %rdx
960  1257 4C89F6         movq   %r14, %rsi
961  125a 4C89F7         movq   %r14, %rdi
962  25d E8000000        call   poly_add_nored@PLT
962      00
```

Listing 2. Assembly code corresponding to Line 49 in Listing 1 of the GLP implementation.

Skipping Line 962 (or Line 49 in Listing 1) yields $b = as$ instead of $b = as + e$. Hence, although the attacker can compute s, the error vector e will remain unknown. In the GLP scheme, however, e is used to compute z_2. Next we describe how an attacker can forge signatures anyway: the attacker chooses randomness y_1, y_2, computes the hash value c for a message μ, $z_1 = y_1 + sc$, and $z_2 = y_2$ (instead of $z_2 = y_2 + ec$). The attacker applies rejection sampling and compresses z_2 to $z_2^\star$ as usual and returns the signature $\sigma = (z_1, z_2^\star, c)$. The verify algorithm accepts this signature σ as we show in the following. Due to the rejection sampling, $z_1, z_2^\star \in \mathcal{R}_{q,[k-32]}$. Furthermore, it holds that $\lfloor az_1 + z_2^\star - tc \rceil_{d,q} = \lfloor az_1 + z_2 - tc \rceil_{d,q} = \lfloor asc + ay_1 + y_2 - asc \rceil_{d,q} = \lfloor ay_1 + y_2 \rceil_{d,q}$. Thus, by skipping a single line an attacker can reveal s and also forge a signature for any message μ.

[6]We create a description of the code written in C in assembly code via the command `/usr/bin/gcc -Wall -g -O3 -c -Wa,-a,-ad -shared -fPIC file.c -o libfile.so > assemblyoutputoffile.lst`

B. During Signature Generation

The signature generation of the schemes that we consider in this paper is rather simple since it is a short sequence of (polynomial) additions and multiplications. Furthermore, the signature generation is randomized. Hence, there are not many skipping operations which lead to information about the secret key. We describe two skipping attacks during the sign algorithm in this section.

1) Skipping the Rejection Condition: Lyubashevsky first applied rejection sampling (introduced by von Neumann [30]) to lattice-based signature schemes to assure that signatures are statistically independent of the secret used to generate them. Thus, learning-the-parallelepiped-attacks introduced by Nguyen and Regev [23] and improved by Ducas and Nguyen [13] are prevented. Ducas and Nguyen need roughly 8000 signatures to reveal the secret. In case of BLISS, ring-TESLA, and the GLP scheme, the rejection sampling is implemented as an `if`-condition, which would have to be skipped in order to circumvent the rejection sampling. Skipping this rejection-sampling-condition in many runs of the sign algorithm might introduce the same security flaw as used by the attacks described in [23] and [13]. Since these attacks exploit the special structure of NTRU-lattices, BLISS might be especially vulnerable because its keys are chosen in an NTRU-like manner. However, to find out the exact number of needed faults, the mentioned attacks have to be adapted to BLISS, ring-TESLA, and the GLP scheme, and to be simulated, which we leave for future work.

2) Skipping the Addition of the Randomness: In the C-implementation of the GLP scheme, z_1 and z_2 of the signature are computed in two steps: first s (resp., e) and c are multiplied. Afterwards, sc and y_1 (resp., ec and y_2) are added, as can be seen in Listing 3. Hence, skipping Line 108 (resp., Line 120) in Listing 3 yields $z_1 = sc$ (resp., $z_2 = ec$).

```
95   poly_setrandom_maxk(y1);
96   poly_setrandom_maxk(y2);
     [...]
107  poly_mul(z1,c,s1);
108  poly_add_nored(z1,z1,y1);
109  poly_coeffreduce(z1);
     [...]
119  poly_mul(z2,c,s2);
120  poly_add_nored(z2,z2,y2);
121  poly_coeffreduce(z2);
```

Listing 3. C code of the GLP implementation for the computation of the signature values z_1 and z_2.

As before, the assembly code of the respective code lines corresponds to jumping to another operation. Hence, this attack gives the same result as zeroing the whole randomness as described in Sec. V-B, i.e., by skipping one line an attacker knows the secret key.

A similar attack is not possible in case of the BLISS or the ring-TESLA implementation. We explain the reason on the example of ring-TESLA. Explanations for BLISS can be found in the full version of this paper [6].

In the implementation of ring-TESLA the value sc is added to the value y as can be seen in Listing 4. Hence, skipping Line 323 in Listing 4 yields $z = y$ (the value vec_y is the output value in the implementation). Since the randomness changes for every run of the sign algorithm, the attacker does not learn anything about the secret.

```
323  poly_add(vec_y, vec_y, Sc);
```

Listing 4. C Code of ring-TESLA for the addition of sc and y.

3) Skipping the Modulus Reduction: Skipping the reduction modulo q during the signature generation does not reveal information about the secret since during the computation of $z = y + sc$ (the only value that is returned and depending on the secret) no modulo reduction is computed in all of the signature schemes. Moreover, the modulo operation is computed very often during the sign algorithm, i.e., it is rather difficult to skip all the modulo operations during the computation via fault attacks.

C. During Verification

To prevent the installation of malicious malware it is not enough to use cryptographic signatures for software updates. It is also necessary to ensure that the verification of these signatures is computed correctly [28]. Hence, we analyze fault attacks during the verification algorithm in this section. We identify two ways to force the acceptance of an invalid signature for any message μ via skipping attacks. In all three signature schemes that are considered in this paper, the verify algorithm consists essentially of computing a hash value c', checking whether this is the same as the input value c (called the correctness check), and checking whether z (resp., z_1 and $z_2^\star$) are small enough (called the size check). Note that we do not consider skipping the computation of the encoding function of the hash value c, since this would lead to an unallocated value. However, we consider zeroing c in Sec. V-D.

1) Skipping the Correctness Check: An adversary chooses c uniformly at random and chooses z (reps., z_1 and $z_2^\star$) small enough and of the expected form (e.g., correct number of zero-coefficients), such that the size check goes through. Afterwards, the attacker computes the hash value c'. Hence, skipping the correctness check yields an acceptance of the (invalid) signature of any message. In the software of ring-TESLA, the correctness check is implemented as the following single `if`-condition:

```
378 if(memcmp(c,c_sig,32)) return -1;
```

where `c_sig` corresponds to c' in our notation and returning `-1` corresponds to not accepting a signature. In case of BLISS and the GLP scheme, the correctness checks are implemented as `if`-conditions for each entry of c, see as an example the respective lines of the GLP implementation in Listing 5.

```
for(i=0;i<20;i++)
 if(sm[i] != h[i])
   goto fail;
```

Listing 5. C Code of the correctness check in the GLP scheme; `sm[0],...,sm[19]` corresponds to c and `h` corresponds to c' in our notation; `goto fail` corresponds to not accepting a signature.

Therefore, the skip has to be realized as a jump out of the `for`-loop after the first iteration. Hence, the invalid signature is accepted as long as c, z_1, and $z_2^\star$ are chosen such that $c[0] = c'[0]$.

2) Skipping the Size Check: We explain this attack by the example of the GLP scheme. It can similarly be applied to BLISS, while ring-TESLA is resistant to this kind of fault attack. The attack works as follows: the attacker chooses $y_1, y_2 \leftarrow_\$ \mathcal{R}_{q,[k]}$ and computes the hash value c for some self-chosen message. Afterwards, the attacker computes $z_1 = a^{-1}(ay_1 + bc)$ (recall that the polynomial a is invertible) and $z_2 = y_2$. Easy computation shows that as long as the size check is skipped, the signature $\sigma = (z_1, z_2^\star, c)$ is accepted. In case of GLP the size check is again implemented as a simple `if`-condition. This is also the case for BLISS, but by construction of the verify algorithm, two `if`-conditions have to be checked for z_1 as indicated by Fig. 5. However, the attack does not work for ring-TESLA for the following reason: to accept the signature (z, c) the equation $c = H\left(\lfloor w_1' \rceil_{d,q}, \lfloor w_2' \rceil_{d,q}, \mu\right)$ has to hold. Therefore $\lfloor w_i' \rceil_{d,q} = \lfloor a_i z - b_i c \rceil_{d,q} = \lfloor a_i y \rceil_{d,q}$ has be fulfilled for $i = 1, 2$, i.e., the signature value z has to fulfill two equations during the verification. During the attack the value z would be uniquely computed via a_1 or a_2. Hence, the probability that z would fulfill both equations is very small.

V. Zeroing Faults

Zeroing fault attacks assume that the attacker can set a whole variable or a part thereof to zero. We present zeroing attacks during the key generation, the sign, and the verify algorithm. Although it has often been questioned if this is a realistic attack scenario, zeroing faults have been realized in practice [22]. In certain cases, zeroing attacks can be realized with skipping attacks, which is why we refer to Section IV in the respective cases.

A. Zeroing the Secret or Error During Key Generation

Zeroing the error polynomial can be implemented as skipping addition operations during the key generation. Hence, we refer to Sec. IV-A for more information.

Similarly, one can zero the secret polynomial. Assume that during the key generation a zeroing fault is induced such that $s = 0$, hence the value $b = e \pmod q$ is returned and the attacker knows the error polynomial e. In case of the GLP scheme this is enough to forge signatures: in case of the GLP scheme, the attacker, knowing e, can compute a (valid) signature for any message μ by choosing $y_1, y_2 \leftarrow_\$ \mathcal{R}_{q,[k]}$ and computing $c \leftarrow H(\lfloor ay_1 + y_2 \rceil_{d,q}, \mu)$, $z_2 = y_2 + ec$, and its compression $z_2^\star$ as usual. Then $z_1 = y_1$ with $y_1 \leftarrow_\$ \mathcal{R}_{q,[k-32]}$ (instead of $z_1 = y_1 + sc$). As in Sec. IV-A, easy computation shows that $(z_1, z_2^\star, c)$ will be accepted by the verify algorithm.

In case of ring-TESLA, signatures cannot be forged in a similar manner, because a signature (c, z) has to fulfill two equations in order to be verified correctly: $\lfloor a_1 z - b_1 c \rceil_{d,q}$ and $\lfloor a_2 z - b_2 c \rceil_{d,q}$. This only occurs if and only if $e_1 c = e_2 c$, which is very unlikely. The attack is not applicable to BLISS since the public value $a_q = 0$ if one of the secret polynomials f, g is set to zero. Hence, the attacker gains not additional information.

In the publicly available software implementation of GLP and ring-TESLA, the key generation and the sign algorithm do not test whether the keys are of the correct form. Thus, the respective attacks would not be detected in the currently available implementations.

B. Zeroing the Randomness During the Signature Generation

In the following, we describe a zeroing attack on the randomness of the signature generation. First, we describe the attack on ring-TESLA. Afterwards, we describe similar attacks on BLISS and the GLP scheme.

1) Description of the Attack Against ring-TESLA: Throughout this section we use the following notation: let the secret polynomial be $s = \sum_{j=0}^{n-1} s_j x^j$. Let $(z^{(1)}, c^{(1)}), ..., (z^{(m)}, c^{(m)})$ be signatures for any messages with $z^{(i)} = \sum_{j=0}^{n-1} z_j^{(i)} x^j$ and $c^{(i)} = \sum_{j=0}^{n-1} c_j^{(i)} x^j$ where $c_j^{(i)} \in \{0, 1\}$ for $i = 1, ..., m$. Furthermore, let $y^{(1)}, ..., y^{(m)}$ be the faulty randomnesses with $y^{(i)} = \sum_{j=0}^{n-1} y_j^{(i)} x^j$. Finally, let $r \in \{1, ..., n\}$ be the number of coefficients that are changed to zero during the fault attack. We assume that the coefficients are changed blockwise, i.e., the coefficients $y_j^{(i)}, ..., y_{j+r}^{(i)}$ are changed for $i \in \{1, ..., m\}$ and $j \in \{0, ..., n-1\}$, and the attacker cannot control which block of r coefficients is set to zero.

The idea of the attack is as follows: first the attacker induces a zeroing fault on the randomness and checks which of the coefficients were changed to zero (we explain later in this section how this is done). The attacker collects equations with $y_j^{(i)} = 0$, i.e., $(s_j, ..., s_0, -s_{n-1}, ..., -s_{j+1})(c_0^{(i)}, ..., c_{n-1}^{(i)})^T = z_j^{(i)}$. The attacker repeats those steps until the set of $n' \geq n$ collected equations is sufficient, i.e., every coefficient of $s_0, ..., s_{n-1}$ is at least once multiplied with a non-zero $c_{j_k}^{(i_k)}$. Hence, the attacker receives the following system of equations, which can be solved uniquely:

$$\mathbf{C} \cdot (s_0, ..., s_{n-1})^T = (z_{j_1}^{(i_1)}, ..., z_{j_{n'}}^{(i_{n'})})^T, \tag{5}$$

with $\mathbf{C} = \begin{pmatrix} c_{j_1}^{(i_1)} & ... & c_0^{(i_1)} & -c_{n-1}^{(i_1)} & ... & -c_{j_1+1}^{(i_1)} \\ & ... & & & ... & \\ c_{j_{n'}}^{(i_{n'})} & ... & c_0^{(i_{n'})} & -c_{n-1}^{(i_{n'})} & ... & -c_{j_{n'}+1}^{(i_{n'})} \end{pmatrix}$.

Next we describe how an attacker can find out which coefficients of the randomness were changed to zero during the i-th fault attack. The equation $z^{(i)} = sc^{(i)} + y^{(i)}$ is equivalent to $\mathbf{z^{(i)}} = \mathsf{Rot}(s)\mathbf{c^{(i)}} + \mathbf{y^{(i)}}$. To simplify the explanation we assume w.l.o.g. that $c_0^{(i)} = ... = c_{\omega-1}^{(i)} = 1$ and $c_\omega^{(i)} = ... = c_{n-1}^{(i)} = 0$. Hence, we can write

$$\begin{aligned} z_0^{(i)} &= s_0 - s_{n-1} + ... - s_{n-\omega} + y_0^{(i)} \\ z_1^{(i)} &= s_1 + s_0 - s_{n-2} + ... - s_{n-\omega+1} + y_1^{(i)} \\ &\cdots \\ z_{n-1}^{(i)} &= s_{n-1} + s_{n-2} + ... + s_{n-\omega-1} + y_{n-1}^{(i)}. \end{aligned} \tag{6}$$

We define $z_j^{(i)} = \varsigma_j^{(i)} + y_j^{(i)}$ for $j = 0, ..., n-1$. Since $s_j \leftarrow D_\sigma$, the expectation value of $|s_j|$ is given by $\mathrm{E}[|s_j|] = \sigma\sqrt{2}/\pi$ for $j = 0, ..., n-1$. Furthermore, since $\|c\| = \omega$, $\varsigma_j^{(i)}$ is Gaussian distributed with standard deviation $\sqrt{\omega}\sigma$ and $\mathrm{E}[|\varsigma_j^{(i)}|] = \sqrt{2\omega/\pi}\sigma$. Via the triangle inequality it holds that

$$B/2 - \sqrt{2\omega/\pi}\sigma \leq \mathrm{E}[|\varsigma_j^{(i)} + y_j^{(i)}|] \leq \sqrt{2\omega/\pi}\sigma + B/2.$$

For the parameter set ring-TESLA-II, i.e., with $B = 2^{22} - 1$, $\omega = 19$, and $\sigma = 52$, $\mathrm{E}[|z_j^{(i)}|]$ is given by

$$\mathrm{E}[|z_j^{(i)}|] \approx \begin{cases} 102, & \text{if } y_j^{(i)} = 0, \\ 2^{21}, & \text{if } y_j^{(i)} \neq 0. \end{cases}$$

Since the difference between the expectation values is very large, we assume that the attacker can unambiguously determine whether or not $y_j^{(i)}$ was changed to zero.

The number of needed zeroing faults strongly depends on the value r. Let m be the number of necessary successful fault inductions to reveal the secret and let S be the set of equations which will be part of Equation (5). We assume that every successful fault induction adds r new equations to the set S, since the hash value c changes for every sign query. Hence, solving the following equation for m gives the number of necessary faults:

$$\frac{1}{3} \cdot \sum_{k=1}^{m} r - \frac{1}{2^n}(k-1) - \frac{1}{2^r} \geq n. \tag{7}$$

Thus, $m \geq \frac{3n}{r - 1/2r}$, where the factor 3 comes from the rejection probability of 0.34 stated in [1]. Assume the attacker can set 12 bytes to zero, i.e., $r = 4$ since each coefficient of y can be saved in three bytes, then $m = 384$. For $r = 1$, we get $m = 1536$.

In case $r = n$, i.e., the complete randomness can be set to zero, only a single successful fault attack is necessary, since then the linear system of equations in Equation (6) can be solved uniquely with high probability. The reason is that the rows of a rotation matrix are $\mathbb{Z}$-linearly independent and $\pm s_0, ..., \pm s_{n-1}$ are independent random variables with high probability.

2) Application to BLISS or the GLP Scheme: The attack can also be applied on BLISS and the GLP scheme. Assume that the zeroing fault was induced on y_1 of the GLP scheme. As explained above, we can recover s. Afterwards, we can compute e by $e = t - as$. Because of the compression function the attack is not effective if y_2 is faulty. Due to the compression algorithm, a maximum of six coefficients are in the final signature for the proposed instantiation. Similarly, we can assume that the zeroing fault was induced on y_1 during the sign algorithm of BLISS. As explained above, we can then recover s_1 and we can recover s_2 by $s_2 = a_1 s_1$. For similar reasons as for the GLP scheme, the attack does not work effectively in case y_2 instead of y_1 is faulty.

3) Application to Signature Schemes over Standard-Lattices: The attack is far less efficient and only applicable for $r = n$ when applied to schemes defined over standard lattices instead of ideal lattices.

Let $r = n$, i.e., the randomness is equal to the zero vector. Let the notation and assumptions be as described above. Then the following system of equations gives $n \cdot m$ equations and n^2 unknowns and can be solved uniquely:

$$\left(z^{(1)}, ..., z^{(m)}\right)^T = \mathsf{Rot}(s) \left(c^{(1)}, ..., c^{(m)}\right)^T.$$

Still it is less efficient than in the ring setting, since the attacker needs to induce at least $3n$ zeroing faults (again the factor 3 comes from the rejection probability). In the ring setting with the same assumption $r = n$, the attacker needs to induce (on average) three zeroing faults successfully.

In case $r < n$, the attack is in general not applicable to signature schemes over standard lattices, since the multiplication of matrices is not commutative, a condition that we need in Equation (5).

C. Zeroing the Hash Value During the Signature Generation

Zeroing the hash value c during the sign algorithm does not lead to more information about the secret key since only the product sc occurs in the final signature. Hence, the attacker only gets access to the randomness, which changes for every run of the sign algorithm in BLISS, ring-TESLA, and the GLP scheme. Hence, knowledge of one specific random value does not reveal information to successfully forge signatures.

D. Zeroing Fault During the Verification Algorithm

In this subsection, we describe a zeroing attack on the polynomial computed from the hash value c using the pseudo code of ring-TESLA. This attack works similarly on the GLP scheme and BLISS since they use the same mechanism as it is used in ring-TESLA, although this is not made explicit in their pseudo code.

The goal of the attacker is to force the verify algorithm to accept a (unvalid) signature for a message μ. To this end, the attacker chooses $z \leftarrow_{\$} \mathcal{R}_{q,[B-U]}$, computes $c' \leftarrow H(\lfloor a_1 z \rceil_{d,q}, \lfloor a_2 z \rceil_{d,q}, \mu)$, and returns (c, z) as signature of μ. During the verify algorithm, first the value $c \leftarrow F(c')$ is computed. Assume c was set to zero during a fault attacks. Hence, $w_1 = a_1 z$ and $w_2 = a_2 z$, and $c'' \leftarrow H(\lfloor a_1 z \rceil_{d,q}, \lfloor a_2 z \rceil_{d,q}, \mu)$. Thus, $c' = c''$ and the signature is accepted.

VI. Countermeasures and Guidelines

We describe countermeasures to prevent fault attacks for each kind of attack described in this paper. Additionally to our guidelines, we refer to the intensive literature about countermeasures in general [29]. Note that it is crucial that a countermeasure can not be easily circumvented by another fault attack. Hence, implementations of countermeasures should always consider preventions against all three kinds of attacks.

A. Countermeasures Against Randomization Faults

One way to prevent the randomization attack described in Sec. III is to check the correctness of the secret key. As long as the randomization fault is not implemented as a skipping attack, it can be prevented by simple correctness checks or comparisons. Our approach is somewhat different: let a^{-1} be the inverse polynomial of a in $\mathcal{R}_q$, s' be the faulty secret, s be the original secret, and let $b' = as' + e \pmod q$. Instead of Line 8 of the GLP scheme (see Fig. 3), we compute $z_1 = a^{-1}(b' - b)c + s'c + y_1 = sc + y_1$. Hence, we always return a signature generated with the correct secret key even if the fault attack described in Sec. III occurred. As long as implemented with respect to the guidelines mentioned in the next section, this countermeasure should not induce vulnerabilities against the described skipping or zeroing attacks. A disadvantage of this countermeasure is that the public key b has to be given as input, i.e., the key sizes are increased. Furthermore, the inverse of a has to be computed. Similarly to the protection of the secret polynomial, the error term could be protected if necessary. Our analysis in Sec. III indicates that aggressive instantiations such as DCK or NTRU are more vulnerable to randomization attacks. Hence, instantiating BLISS or the GLP scheme over ring-LWE or ring-SIS would strengthen the security of those schemes with respect to fault attacks. Most probably this would lead to a serious efficiency penalty.

B. *Countermeasures Against Skipping Faults*

We describe countermeasures to prevent the skipping attacks presented in Sec. IV.

One way to prevent skipping faults addressing the addition in general is to define a new variable to save the resulting sum, e.g., Listing 6. Skipping Line 49 of the countermeasure in Listing 6 does not lead to a successful attack since the value `b2` would not be allocated and a segmentation fault would be triggered. Hence, no information about the secret is revealed.

```
   // original                  // countermeasure
48 poly_mul_a(b, s);            poly_mul_a(b1, s);
49 poly_add_nored(b,b,e);       poly_add_nored(b2,b1,e);
```

Listing 6. Comparison of the original code of the GLP scheme and an example of a countermeasure against skipping the addition during key generation.

A different approach which prevents skipping attacks in certain cases is to add secret information to random information and not the other way around, e.g., use the code shown on the bottom of Listing 7 instead of the original GLP code. Hence, skipping Line 108b in Listing 7 results in $z_1 = y_1$ instead of $z_1 = y_1 + as$. Since y_1 changes for every sign query, the attacker does not gain information about the secret. This is already realized in the implementations of BLISS and ring-TESLA.

```
        // original
95a     poly_setrandom_maxk(y1);
        [...]
107a    poly_mul(z1,c,s1);
108a    poly_add_nored(z1,z1,y1);
109a    poly_coeffreduce(z1);

        //countermeasure
95b     poly_setrandom_maxk(z1);
        [...]
107b    poly_mul(v1,c,s1);
108b    poly_add_nored(z1,z1,v1);
109b    poly_coeffreduce(z1);
```

Listing 7. Comparison of the original code of the GLP scheme and an example of a countermeasure against skipping the addition of the randomness during signature generation.

Since our analysis focuses on (and our countermeasures protect against) first order fault attacks we assume that an attack can not know the content of the cache and induce a skipping fault at the same time. Hence, the countermeasure described above should prevent first order fault attacks. However, a stronger adversary might know the content of the cache, skip Line 95b, and hence can compute s_1. We leave the analysis of this scenario for future work.

Besides the countermeasures mentioned above, it should be ensured that only correctly formed or totally random keys are returned. In the following, we describe a method to prevent the skipping attack presented in Sec. IV-A2. The goal of this attack is to skip operations during the key generation algorithm such that the public key is not of the correct form. A faulty b can either be $b = as+e \pmod q$, $b = as \pmod q$, $b = e \pmod q$, or $b = 0$. To prevent returning a faulty b, the additional computations shown in Fig. 2 should be implemented for the GLP scheme (and similarly for ring-TESLA and BLISS). In case b is not

```
1  $b = as + e \pmod q$
2  $u \leftarrow_{\$} \mathbb{Z}_q$
3  $\nu = \frac{\|t-as\|+u}{\|e\|+u}$
4  If $s = \nu s \wedge e = \nu e$:
5      Return $sk = (\nu s, \nu e)$
6  Else:
7      Restart key generation
```

Fig. 2. Pseudo code of a countermeasure to check whether the key pair is generated correctly: the returned key pair is either of the correct form or the secret and the public key do not correspond to each other.

faulty, $\nu = 1$ and the correct elements s, e are returned. In case b is faulty, no security flaw occurs because even if Line 4 is skipped, the secret and the public key do not correspond to each other. Hence, at worst the signer uses the invalid keys to sign messages which can not be verified with the corresponding faulty b.

Due to (Gaussian) sampling of elements, it is rather difficult to induce a skipping fault to skip the rejection sampling at the right time. Nevertheless, it is advisable to make sure that rejection sampling is applied correctly. In case of ring-TESLA and the GLP scheme the rejection sampling is implemented as an `if`-condition such that the signature is returned if the `if`-condition is true. In assembly code this means, that the signature is returned if the zero flag is equal to 1. Hence, when the `if`-condition is skipped by fault, a signature is returned if and only if the zero flag was set equal to 1 in an earlier computation. It is reasonable to assume that this happens with probability 0.5. Hence, formulating the `if`-condition as it is done for ring-TESLA and the GLP scheme does not prevent the skipping attack completely, but it doubles the (expected) number of necessary fault injections. In case of the BLISS implementation, the rejection sampling is implemented as `if`-condition(s) such that a signature is rejected if the `if`-condition holds true. Hence, skipping this `if`-condition means to skip the rejection sampling. Reformulating the `if`-condition as it is done for ring-TESLA and the GLP scheme would make this skipping attack much more complicated.

C. Countermeasures Against Zeroing Faults

Zeroing faults can often be categorized as randomization or skipping faults. Hence, zeroing faults can often be prevented by the countermeasures described in Sec. VI-A and VI-B.

Assuming that the zeroing fault is not caused by skipping or randomizing faults, we can prevent a zeroing attack by simply checking whether the values of the secret or error polynomial (Sec. V-A), the randomness during signing (Sec. V-B), the hash value (Sec. V-C), or the encoding polynomial (Sec. V-D) are zero, since we only consider first-order faults in this paper.

VII. Conclusion

In this paper, we analyzed the lattice-based signature schemes BLISS, ring-TESLA, and the GLP scheme and their implementations with respect to fault attacks. Furthermore, we presented countermeasures against the described attacks. Hereby, we considered three types of faults: randomization, skipping, and zeroing faults.

For nine of the 15 considered attacks at least one of the three schemes was vulnerable. We summarize our results in Table I. All three schemes are vulnerable against zeroing faults during the sign algorithm, against zeroing faults during the verification, against skipping faults during the key generation, against two kinds of skipping faults during the verification algorithm, and (to a variable extent) against skipping faults during the signature generation algorithm. Moreover, the GLP scheme is vulnerable to an additional skipping attack during the sign algorithm and an additional zeroing attack during the key generation. In Table III, we recall the (expected) minimal number of successful faults needed for the respective fault attacks.

TABLE III
Comparison of the GLP scheme, BLISS, and ring-TESLA with respect to the expected number of successful faults such that the fault attack succeeds. The table shows the algorithms which the fault attacks target, i.e., key generation (KG), signature generation (S), and verify (V). If the scheme is vulnerable to the respective attack, the number of necessary successful faults is given, otherwise we write -.

Fault Attack	Algorithm	GLP	BLISS	ring-TESLA
Rand. of secret, $r = 4$	S	151	733	-
Skip of addition	KG	1	1	1
Skip of rejection	S	?	?	?
Skip of addition	S	1	-	-
Skip of correct-check	V	1	1	1
Skip of size-check	V	1	1	-
Zero. of secret	KG	1	-	-
Zero. of randomness, $r = 1$	S	1	2[7]	1
Zero. of hash polynomial	V	1	1	1

We state that the three signature schemes and their implementations behave rather similar under fault attacks. However, the different instantiations of the schemes lead to different vulnerabilities. BLISS and the GLP scheme are more vulnerable to a randomization attack during the key generation because of their aggressive instantiation with ternary secret and error. Moreover, our analysis shows that ideal-lattice-based schemes are in general more vulnerable to zeroing attacks during the sign algorithm than standard-lattice-based schemes. We propose effective countermeasures for each of the analyzed attacks. Most of them are very efficient, since they do not require time-consuming computations.

Future Work. This work is a starting point for fault analysis of lattice-based cryptography. It is not comprehensive, e.g., we did not analyze underlying algebraic computations such as polynomial multiplications and we did not consider fault attacks targeting these underlying computations, e.g., we did not analyze the effects of zeroing attacks on the most (or least) significant bits of polynomial coefficients or the modulus.

In addition to these ideas for deeper fault analysis, we leave for future work to compare the original implementations with implementations that take our countermeasures into account and to verify the effectiveness of the proposed measures by a software simulation. Moreover, since this work focuses on theoretical fault analysis, the practical realization of the proposed attacks remains to be done.

References

[1] S. Akleylek, N. Bindel, J. Buchmann, J. Krämer, and G. A. Marson, "An efficient lattice-based signature scheme with provably secure instantiation," in International Conference on Cryptology – AFRICACRYPT 2016, D. Pointcheval, T. Rachidi, and A. Nitaj, Eds. Springer, 2016, pp. 44–60. 1, 3, 7, 10, 15

[2] M. R. Albrecht, C. Cid, J. Faugère, R. Fitzpatrick, and L. Perret, "Algebraic algorithms for LWE problems," Cryptology ePrint Archive, Report 2014/1018, 2014. 5

[3] M. R. Albrecht, R. Fitzpatrick, and F. Göpfert, "On the efficacy of solving LWE by reduction to unique-svp," in Information Security and Cryptology - ICISC 2013, 2013, pp. 293–310. 4

[4] S. Bai and S. D. Galbraith, "An improved compression technique for signatures based on learning with errors," in Topics in Cryptology - CT-RSA 2014, 2014, pp. 28–47. 4

[5] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimhalu, and T. Ngair, "Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults," in Security Protocols: 5th International Workshop Paris, France, B. Christianson, B. Crispo, M. Lomas, and M. Roe, Eds. Springer, 1998, pp. 115–124. 5

[6] N. Bindel, J. Buchmann, and J. Krämer, "Lattice-based signature schemes and their sensitivity to fault attacks," Cryptology ePrint Archive, Report 2016/415, 2016. 4, 8, 9

[7] Since the rejection sampling is applied independently from the randomness.

[7] J. Blömer, R. G. da Silva, P. Günther, J. Krämer, and J. Seifert, "A practical second-order fault attack against a real-world pairing implementation," in 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014, 2014, pp. 123–136. 7

[8] L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom, "Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme," Cryptology ePrint Archive, Report 2016/300, 2016. 1

[9] J. Buchmann, F. Göpfert, T. Güneysu, T. Oder, and T. Pöppelmann, "High-performance and lightweight lattice-based public-key encryption," in To appear in IoTPTS 2016. 5

[10] J. Buchmann, F. Göpfert, R. Player, and T. Wunderer, "On the hardness of lwe with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack," in International Conference on Cryptology AFRICACRYPT 2016. Springer, 2016, pp. 24–43. 5

[11] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in Advances in Cryptology - CRYPTO 2013, 2013, pp. 40–56. 1, 3, 15

[12] L. Ducas and T. Lepoint, "Bliss: Bimodal lattice signature schemes," http://bliss.di.ens.fr/. 7

[13] L. Ducas and P. Q. Nguyen, "Learning a zonotope and more: Cryptanalysis of ntrusign countermeasures," in Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings, 2012, pp. 433–450. 8

[14] N. Gama and P. Q. Nguyen, "Predicting lattice reduction," in Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings, 2008, pp. 31–51. 4

[15] C. Giraud and E. W. Knudsen, "Fault attacks on signature schemes," in Information Security and Privacy: 9th Australasian Conference, ACISP, 2004, pp. 478–491. 1, 5

[16] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann, "Practical lattice-based cryptography: A signature scheme for embedded systems," in Cryptographic Hardware and Embedded Systems - CHES 2012, 2012, pp. 530–547. 1, 3, 14

[17] G. Güneysu, T. Oder, T. Pöppelmann, and P. Schwabe, "Software speed records for lattice-based signatures," https://cryptojedi.org/crypto/index.shtml#lattisigns. 7

[18] A. A. Kamal and A. M. Youssef, "Fault analysis of the ntruencrypt cryptosystem," IEICE Transactions, vol. 94-A, no. 4, pp. 1156–1158, 2011. 1

[19] R. Lindner and C. Peikert, "Better key sizes (and attacks) for lwe-based encryption," in Topics in Cryptology - CT-RSA 2011, 2011, pp. 319–339. 4

[20] V. Lyubashevsky, "Lattice signatures without trapdoors," in Advances in Cryptology - EUROCRYPT 2012, 2012, pp. 738–755. 3

[21] D. Micciancio and C. Peikert, "Hardness of SIS and LWE with small parameters," in Advances in Cryptology - CRYPTO 2013, 2013, pp. 21–39. 5

[22] D. Naccache, P. Q. Nguyen, M. Tunstall, and C. Whelan, "Experimenting with faults, lattices and the DSA," in Public Key Cryptography - PKC 2005, 2005, pp. 16–28. 9

[23] O. Nguyen, Phong Q.and Regev, "Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures," pp. 271–288, 2006. 8

[24] N. S. A. (NSA), "Cryptography today," https://www.nsa.gov/ia/programs/suiteb_cryptography/, Aug 19, 2015. 1

[25] N. I. of Standards and T. (NIST), "Post-quantum cryptography: Nist's plan for the future," https://pqcrypto2016.jp/data/pqc2016_nist_announcement.pdf, Aug 19, 2015. 1

[26] C. Peikert, "How (not) to instantiate ring-lwe," Cryptology ePrint Archive, Report 2016/351, 2016. 2

[27] M.-J. O. Saarinen, "Arithmetic coding and blinding countermeasures for ring-lwe," Cryptology ePrint Archive, Report 2016/276, 2016. 1

[28] J. Seifert, "On authenticated computing and rsa-based authentication," in Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, 2005, pp. 122–127. 9

[29] I. Verbauwhede, D. Karaklajic, and J. Schmidt, "The fault attack jungle - A classification model to guide you," in 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, 2011, pp. 3–8. 5, 11

[30] J. von Neumann, "Various techniques used in connection with random digits," in Monte Carlo Method, ser. National Bureau of Standards Applied Mathematics Series, A. S. Householder, G. E. Forsythe, and H. H. Germond, Eds., 1951, vol. 12, pp. 36–38. 8

[31] Özgür Dagdelen, R. E. Bansarkhani, F. Göpfert, T. Güneysu, T. Oder, T. Pöppelmann, A. H. Sánchez, and P. Schwabe, "High-speed signatures from standard lattices," in Progress in Cryptology – LATINCRYPT 2014, D. F. Aranha and A. Menezes, Eds., vol. 8895. Springer, 2015, pp. 84–103. 4

Appendix

A. *Signature Schemes*

In this section we depict the signature schemes GLP, ring-TESLA, and BLISS in Fig. 3, 4, and 5, respectively.

```
KeyGen($1^\lambda$) :
    $s, e \leftarrow_\$ \mathcal{R}_{q,[1]}$
    $a \leftarrow_\$ \mathcal{R}_q$
    $b \leftarrow as + e \pmod{q}$
    sk $\leftarrow (s, e)$, vk $\leftarrow (a, b)$
    Return (sk, vk)

Sign($\mu; a, s, e$) :
    $y_1, y_2 \leftarrow_\$ \mathcal{R}_{q,[k]}$
    $c \leftarrow H\left(\lfloor a y_1 + y_2 \rceil_{d,q}, \mu\right)$
    $z_1 \leftarrow y_1 + sc$
    $z_2 \leftarrow y_2 + ec$
    If $z_1, z_2 \notin \mathcal{R}_{k-32}$:
        Restart
    Else: $z_2^\star \leftarrow$ compress$(a z_1 - tc, z_2, p, k-32)$
    Return $(z_1, z_2^\star, c)$
Verify($\mu; z_1, z_2^\star, c; a, b$)
    $c' \leftarrow H\left(\lfloor a z_1 + z_2^\star - bc \rceil_{d,q}, \mu\right)$
    If $c = c' \wedge\ z_1, z_2^\star \in \mathcal{R}_{k-32}$:
        Return 1
    Else: Return 0
```

Fig. 3. Specification of the GLP scheme by Güneysu et al. [16]. The rounding operator $\lfloor \cdot \rceil_{d,q}$ corresponds to the function used in the original paper with $d = \log_2(k)$, where k is the parameter used in [16]. For detailed information about the system parameters and the procedure compress, we refer to the original work.

B. *The Ring Learning with Errors Problem (R-LWE)*

In this section, we recall the ring learning with errors problem (R-LWE). We start by defining the LWE distribution and then state the ring variants of the search and the decisional learning with errors problem.

```
$\mathsf{KeyGen}(1^\lambda)$ :
    $a_1, a_2 \leftarrow_\$ \mathcal{R}_q$
    $s, e_1, e_2 \leftarrow D_\sigma^n$
    If $\mathsf{checkE}(e_1) = 0 \vee \mathsf{checkE}(e_2) = 0$
        Restart
    $b_1 \leftarrow a_1 s + e_1 \pmod q$
    $b_2 \leftarrow a_2 s + e_2 \pmod q$
    $\mathsf{sk} \leftarrow (s, e_1, e_2)$, $\mathsf{vk} \leftarrow (b_1, b_2)$
    Return $(\mathsf{sk}, \mathsf{vk})$

$\mathsf{Sign}(\mu; a_1, a_2, s, e_1, e_2)$ :
    $y \leftarrow_\$ \mathcal{R}_{q,[B]}$
    $v_1 \leftarrow a_1 y \pmod q$
    $v_2 \leftarrow a_2 y \pmod q$
    $c' \leftarrow H\left(\lfloor v_1 \rceil_{d,q}, \lfloor v_2 \rceil_{d,q}, \mu\right)$
    $c \leftarrow F(c')$
    $z \leftarrow y + sc$
    $w_1 \leftarrow v_1 - e_1 c \pmod q$
    $w_2 \leftarrow v_2 - e_2 c \pmod q$
    If $|[w_1]_{2^d}|, |[w_2]_{2^d}| \notin \mathcal{R}_{2^d - L} \vee z \notin \mathcal{R}_{B-U}$:
        Restart
    Return $(z, c')$

$\mathsf{Verify}(\mu; z, c'; a_1, a_2, b_1, b_2)$
    $c \leftarrow F(c')$
    $w_1' \leftarrow a_1 z - b_1 c \pmod q$
    $w_2' \leftarrow a_2 z - b_2 c \pmod q$
    $c'' \leftarrow H\left(\lfloor w_1' \rceil_{d,q}, \lfloor w_2' \rceil_{d,q}, \mu\right)$
    If $c' = c'' \wedge\ z \in \mathcal{R}_{B-U}$:
        Return 1
    Else: Return 0
```

Fig. 4. Specification of the scheme ring-TESLA by Akleylek et al. [1]. For detailed information about the system parameters and the procedure checkE, we refer to the original work.

Definition 1 (Learning with Errors Distribution): Let $n, q > 0$ be integers, $s \in \mathcal{R}_q$, and χ be a distribution over $\mathcal{R}$. We define by $\mathcal{D}_{s,\chi}$ the R-LWE distribution which outputs $(a, \langle a, s\rangle + e) \in \mathcal{R}_q \times \mathcal{R}_q$, where $a \leftarrow_\$ \mathcal{R}_q$ and $e \leftarrow \chi$.

Definition 2 (Ring Learning with Errors Problem): Let $n, q > 0$ be integers and $n = 2^k$ for some $k \in \mathbb{N}_{>0}$ and χ be a distribution over $\mathcal{R}$.

Given n, q, and m LWE-samples, the (search) ring learning with errors problem $R - LWE_{n,m,q,\chi}$ is to find the polynomial s. Given n, q, and samples $(a_1, b_1), ..., (a_m, b_m)$ the (decisional) ring learning with errors problem is to decide whether the samples are LWE-samples or whether $b_1, ..., b_m$ are chosen uniformly random over $\mathbb{Z}_q[x]/(x^n + 1)$.

```
$\mathsf{KeyGen}(1^\lambda)$
    $f, g \leftarrow_\$ F_{d_1,d_2}$
    If $N_\lambda(\mathbf{S}) \geq 5C^2(\lceil \delta_1 n \rceil + 4\lceil 4\delta_2 n \rceil)\kappa$
        Restart
    $a_q = (2g + 1)/f \pmod q$
    If $f$ not invertible
        Restart
    $(s_1, s_2)^T \leftarrow (f, 2g + 1)^T$
    $(a_1, a_2) = (2a_q, q - 2) (\mathrm{mod}\ 2q)$
    $\mathsf{sk} \leftarrow (s_1, s_2)^T$, $\mathsf{vk} \leftarrow (a_1, a_2)$
    Return $(\mathsf{sk}, \mathsf{vk})$

$\mathsf{Sign}(\mu; \mathbf{A} = (a_1, q - 2); \mathbf{S} = (s_1, s_2)^T)$ :
    $y_1, y_2 \leftarrow D_\sigma$
    $u = \xi a_1 y_1 + y_2 (\mathrm{mod}\ 2q)$
    $c \leftarrow H\left(\lfloor u \rceil_{d,2q}, \mu\right)$
    $b \leftarrow_\$ \{0, 1\}$
    $z_1 \leftarrow y_1 + (-1)^b s_1 c$
    $z_2 \leftarrow y_2 + (-1)^b s_2 c$
    Continue with probability $1/\nu$
    $z_2^\star \leftarrow \left(\lfloor u \rceil_{d,2q} - \lfloor u - z_2 \rceil_{d,2q} \mod p\right)$
    Return $(z_1, z_2^\star, c)$

$\mathsf{Verify}(\mu; \mathbf{A} = (a_1, q - 2); z_1, z_2^\star, c)$
    $c' \leftarrow H\left(\lfloor \xi a_1 z_1 + \xi q c (\mathrm{mod}\ 2q) \rceil_{d,2q} + z_2^\star (\mathrm{mod}\ p), \mu\right)$
    If $c = c' \wedge\ ||(z_1|2^d z_2^\star)||_2 \leq B_2 \wedge ||(z_1|2^d z_2^\star)||_\infty \leq B_\infty$:
        Return 1
    Else: Return 0
```

Fig. 5. Specification of the scheme BLISS by Ducas et al. [11]. For detailed information about the system parameters and the definition of $N_\lambda(\cdot)$, we refer to the original work. The set F_{d_1,d_2} is defined as $F_{d_1,d_2} = \{h = \sum_{i=0}^{n-1} h_i x^i | h_i \in \{-2, -1, 0, 1, 2\},\ |\{h_i \in \{-1, 1\}\}| = d_1,\ |\{h_i \in \{-2, 2\}\}| = d_2\}$ and $\nu = \left(M \exp\left(-||Sc||^2/(2\sigma^2) cosh(\langle z, Sc\rangle/\sigma^2)\right)\right)$.