Submission to NIST's post-quantum project: lattice-based digital signature scheme qTESLA

Name of the cryptosystem: `qTESLA`

Principal and auxiliary submitters:

Nina Bindel, (Principal submitter)	Technische Universität Darmstadt, Hochschulstrasse 10, 64289 Darmstadt, Germany, Email: `nbindel@cdc.informatik.tu-darmstadt.de`, Phone: 004961511620667 Signature: Nina Bindel
Sedat Akleylek,	Ondokuz Mayis University, Turkey
Erdem Alkim,	Ege University, Turkey
Paulo S. L. M. Barreto,	University of Washington Tacoma, USA
Johannes Buchmann,	Technische Universität Darmstadt, Germany
Edward Eaton,	ISARA Corporation, Canada
Gus Gutoski,	ISARA Corporation, Canada
Juliane Krämer,	Technische Universität Darmstadt, Germany
Patrick Longa,	Microsoft Research, USA
Harun Polat,	Technische Universität Darmstadt, Germany
Jefferson E. Ricardini,	University of São Paulo, Brazil
Gustavo Zanon,	University of São Paulo, Brazil

Inventors of the cryptosystem:

All the submitters by name based on a previous scheme by Shi Bai and Steven Galbraith and several other previous works, as explained in the body of this document.

Owners of the cryptosystem:

None (dedicate to the public domain).

Changelog

This is the changelog of this document and the corresponding implementation of qTESLA.

Version	Date	Description of changes
1.0	11/30/2017	• Original submission to NIST.
2.0	06/14/2018	• qTESLA described generically using $k > 1$ R-LWE samples. • Signing algorithm changed to probabilistic (instead of deterministic). • New parameter sets proposed: three heuristic and two provably-secure parameter sets. • Improved explanation of the realization of the different functions (Section 2.4). • Minor changes and refinements throughout the document. • C-only reference implementation corrected; e.g., to have proper protection against timing and cache attacks. • C-only reference implementation improved; e.g., to have more resilience against certain fault attacks.
2.1	06/30/2018	• Corrected typo that assumed a exponent d or $d+1$ instead of $d-1$ or d (resp.) in some places. • Small fix in the bounds of the signature rejection evaluation, line 18 of Algorithm 7. Updated KATs accordingly. • Applied notation $\bmod^{\pm}$ to denote the use of a centered representative in Algorithms 7 and 8. • Updated correctness proof in Section 2.3. • Replaced bit-hardness by corresponding bit-security figures for the qTESLA parameter sets in Table 6.

Contents

1 Introduction

This document presents a detailed specification of qTESLA, a family of post-quantum signature schemes based on the hardness of the decisional Ring Learning With Errors (R-LWE) problem. qTESLA is an efficient variant of the Bai-Galbraith signature scheme —which in turn is based on the "Fiat-Shamir with Aborts" framework by Lyubashevsky— adapted to the setting of ideal lattices.

qTESLA utilizes *two* different approaches for parameter generation in order to target a wide range of application scenarios. The first approach, referred to as "heuristic qTESLA", follows a *heuristic* parameter generation. The second approach, referred to as "provably-secure qTESLA", follows a *provably-secure* parameter generation according to existing security reductions.

Concretely, qTESLA includes *five* parameter sets targeting *two* security levels:

I Heuristic qTESLA:

(1) qTESLA-I: NIST's security category 1.

(2) qTESLA-III-speed: NIST's security category 3 (option for speed).

(3) qTESLA-III-size: NIST's security category 3 (option for size).

II Provably-secure qTESLA:

(1) qTESLA-p-I: NIST's security category 1.

(2) qTESLA-p-III: NIST's security category 3.

The present document is organized as follows. In the remainder of this section, we summarize the main features of qTESLA and describe related previous work. In Section 2, we provide the specification details of the scheme, including a basic and a formal algorithmic description, the functions that are required for the implementation, and the proposed parameter sets. In Section 3, we analyze the performance of our implementations. Section 4 includes the details of the known answer values. Then, we discuss the (provable) security of our proposal in Section 5, including an analysis of the concrete security level and the security against implementation attacks. Section 6 ends this document with a summary of the advantages and limitations of qTESLA.

1.1 qTESLA highlights

qTESLA comes in two flavors: heuristic qTESLA and provably-secure qTESLA. The former is optimized for efficiency and key size while the latter is tailored for high-security applications in which the additional security assurances from the provably-secure proof are valued. In the following paragraphs we highlight relevant properties of each approach.

qTESLA's main features can be summarized as follows:

- **Simplicity.** qTESLA is simple and easy to implement, and its design makes possible the realization of compact and portable implementations that achieve high performance. In addition, the use of a simplified Gaussian sampler is limited to key generation.
- **Compactness of signatures.** qTESLA signatures are designed to be relatively small, at the expense of (slightly) larger public keys.
- **Security foundation.** The underlying security of qTESLA is based on the hardness of the decisional R-LWE problem, and comes accompanied by a tight security proof in the (quantum) random oracle model.
- **Practical security.** By design, qTESLA facilitates secure implementations. In particular, it supports *constant-time* implementations (i.e., implementations that are secure against timing and cache side-channel attacks), and is inherently protected against certain simple yet powerful fault attacks.
- **Scalability.** qTESLA's simple design makes it straightforward to easily support more than one security level and parameter set with a single, efficient implementation.

In addition, Heuristic qTESLA, comprising the parameters sets qTESLA-I, qTESLA-III-speed and qTESLA-III-size, features:

- **High speed.** Heuristic qTESLA achieves very high performance for the operations that are typically time-critical, namely, signing and verification. This is accomplished at the expense of a moderately more expensive key generation, which is usually performed offline.

For added flexibility, we present two options for Heuristic qTESLA for the NIST's security category 3, namely, qTESLA-III-speed, which prioritizes raw speed over signature and key sizes, and qTESLA-III-size, which prioritizes signature and key sizes over speed.

Security. The security of qTESLA is proven using the reductionist approach, i.e., we construct an efficient reduction that turns any successful adversary against qTESLA into one that solves R-LWE. Accordingly, we instantiate heuristic qTESLA such that the corresponding R-LWE parameters (namely the dimension n, the standard deviation of the discrete Gaussian distribution σ, and the modulus q) provide an R-LWE instance of a certain hardness. This approach features high-speed execution and a small memory footprint while requiring relatively compact keys and signatures.

Since our security reductions are also *explicit*, i.e., they explicitly relate an instantiation of qTESLA with an R-LWE instance, we go one step further and choose parameters according to our security reduction. That is, these qTESLA instantiations, which are called provably-secure qTESLA parameter sets, are *provably* secure in the (quantum) random oracle model. For provably-secure qTESLA we present the parameter sets qTESLA-p-I and qTESLA-p-III. Despite these security assurances, provably-secure qTESLA achieves relatively good performance and offers relatively compact signatures. On the downside, this option requires larger public keys.

In summary, the qTESLA family of post-quantum signature schemes offers great flexibility by allowing a selection of schemes that exhibit different degrees of performance and security. In each case, design decisions have been taken towards enabling efficient yet simple and compact implementations.

1.2 Related work

The signature scheme proposed in this submission is the result of a long line of research. The first work in this line is the signature scheme proposed by Bai and Galbraith [14] which is based on the Fiat-Shamir construction of Lyubashevsky [47]. The scheme by Bai and Galbraith is constructed over standard lattices and comes with a (non-tight) security reduction from the LWE and the short integer solution (SIS) problems in the random oracle model. Dagdelen *et al.* [26] presented improvements and the first implementation of the Bai-Galbraith scheme. The scheme was subsequently studied under the name TESLA by Alkim, Bindel, Buchmann, Dagdelen, Eaton, Gutoski, Krämer, and Pawlega [9], who provided an alternative security reduction from the LWE problem in the quantum random oracle model.

A variant of TESLA over ideal lattices was derived under the name ring-TESLA [1] by Akleylek, Bindel, Buchmann, Krämer, and Marson. Since then, there have appeared subsequent works aimed at improving the efficiency of the scheme [16, 39]. Most notably, a scheme called TESLA# [16] by Barreto, Longa, Naehrig, Ricardini, and Zanon included several implementation improvements. Finally, several works [19, 20, 34] have focused on the analysis of ring-TESLA against side-channel and fault attacks.

In this document, we consolidate the most relevant features of the prior works with the goal of designing the quantum-secure signature scheme qTESLA.

Acknowledgments

We are grateful to Michael Naehrig for his valuable feedback on an earlier version of this document, to Peter Pessl and Matthias Kannwischer for comments and suggestions for a previous version of our implementations, and to Vadim Lyubashevsky for pointing out a typo in an earlier version of the parameter generation script that invalidated the "provably-secure" property of the parameter sets originally submitted to NIST on November 2017 (version 1.0 of this document).

SA and EA are partially supported by TÜBITAK under grant no. EEEAG-116E279. NB is partially supported by the German Research Foundation (DFG) as part of project P1 within the CRC 1119 CROSSING. JR is supported by the joint São Paulo Research Foundation (FAPESP)/Intel Research grant 2015/50520-6 "Efficient Post-Quantum Cryptography for Building Advanced Security Applications".

2 Specification

Next, we give an informal description of the basic scheme that is used to specify qTESLA. A formal specification of qTESLA's key generation, signing and verification algorithms then follows in Section 2.2. The correctness of the scheme is discussed in Section 2.3. We describe the implementation of the functions required by qTESLA in Section 2.4, and explain all the system parameters and the proposed parameter sets in Section 2.5.

2.1 Basic signature scheme

Informal descriptions of the algorithms that give rise to the signature scheme qTESLA are shown in Algorithms 1, 2 and 3. These algorithms require two basic terms, namely, *B-short* and *well-rounded*, which are defined below.

Let q, L_E, L_S, and d be system parameters that denote the modulus, the bound constant for error polynomials, the bound constant for the secret polynomial, and the rounding value, respectively. An integer polynomial y is *B-short* if each coefficient is at most B in absolute value. We call an integer polynomial w *well-rounded* if w is $(\lfloor q/2 \rceil - L_E)$-short and $[w]_L$ is $(2^{d-1} - L_E)$-short, where $[w]_L$ denotes the unique integer in $(-2^{d-1}, 2^{d-1}] \subset \mathbb{Z}$ such that $w = [w]_L$ modulo 2^d. Likewise, $[w]_M$ is the value represented by all but the d least significant bits of $(w - [w]_L)$. Let $\mathcal{R} = \mathbb{Z}[x]/\langle x^n + 1 \rangle$ and $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1 \rangle$. For simplicity we assume that the hash oracle $\mathsf{H}(\cdot)$ maps from $\{0,1\}^*$ to $\mathbb{H}$, where $\mathbb{H}$ denotes the set of polynomials $c \in \mathcal{R}$ with coefficients in $\{-1, 0, 1\}$ with exactly h nonzero entries, i.e., we ignore the encoding function F introduced in Section 2.2.

Because of the random generation of the polynomial y (see line 1 of Alg. 2), Algorithm 2 is described as a *non-deterministic* algorithm. This property implies that different randomness is required for each signature. For the formal specification of qTESLA we incorporate an additional improvement: qTESLA requires a combination of fresh randomness and a fixed value for the generation of y (see Section 2.2). This design feature is added in order to pre-

Algorithm 1 Informal description of the key generation

```
Require: -
Ensure: Secret key $sk = (s, e_1, ..., e_k, a_1, ..., a_k)$, and public key $pk = (a_1, ..., a_k, t_1, ..., t_k)$
$a_1, ..., a_k \leftarrow \mathcal{R}_q$ invertible ring elements.
Choose $s \in \mathcal{R}$ with entries from $\mathcal{D}_\sigma$. Repeat step if the $h$ largest entries of $s$ sum to $L_S$.
For $i = 1, ..., k$: Choose $e_i \in \mathcal{R}$ with entries from $\mathcal{D}_\sigma$. Repeat step at iteration $i$ if the $h$ largest entries of $e_i$ sum to $L_E$.
For $i = 1, ..., k$: Compute $t_i \leftarrow a_i s + e_i \in \mathcal{R}_q$.
Return $sk = (s, e_1, ..., e_k, a_1, ..., a_k)$ and $pk = (a_1, ..., a_k, t_1, ..., t_k)$.
```

Algorithm 2 Informal description of the signature generation

```
Require: Message $m$, secret key $sk = (s, e_1, ..., e_k, a_1, ..., a_k)$
Ensure: Signature $(z, c)$
1: Choose $y$ uniformly at random among $B$-short polynomials in $\mathcal{R}_q$.
2: $c \leftarrow \mathsf{H}([a_1 y]_M, ..., [a_k y]_M, m)$.
3: Compute $z \leftarrow y + sc$.
4: If $z$ is not $(B - L_S)$-short then retry at step 1.
5: For $i = 1, ..., k$: If $a_i y - e_i c$ is not well-rounded then retry at step 1.
6: Return $(z, c)$.
```

Algorithm 3 Informal description of the signature verification

```
Require: Message $m$, public key $pk = (a_1, ..., a_k, t_1, ..., t_k)$, and signature $(z, c)$
Ensure: “Accept” or “reject” signature
1: If $z$ is not $(B - L_S)$-short then return reject.
2: For $i = 1, ..., k$: Compute $w_i \leftarrow a_i z - t_i c \in \mathcal{R}_q$.
3: If $c \neq \mathsf{H}([w_1]_M, ..., [w_k]_M, m)$ then return reject.
4: Return accept.
```

vent some implementation pitfalls and, at the same time, protect against some simple but devastating fault attacks. We discuss the advantages of our approach in Section 5.3.

2.2 Formal description of qTESLA

Below, we define all the necessary functions, sets, and system parameters in qTESLA.

The description of the scheme depends on the following system parameters: λ, κ, n, k, q, σ, L_E, L_S, B, d, h, and b_{GenA}. Let $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$, $\mathcal{R} = \mathbb{Z}[x]/\langle x^n + 1\rangle$, $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$, $\mathcal{R}_{q,[I]} = \{f \in \mathcal{R}_q \mid f = \sum_{i=0}^{n-1} f_i x^i,\ f_i \in [-I, I]\}$, and $\mathbb{H}_{n,h} = \{f \in \mathcal{R}_q \mid f = \sum_{i=0}^{n-1} f_i x^i,\ f_i \in \{-1, 0, 1\},\ \sum_{i=0}^{n-1} |f_i| = h\}$. Let $\mathcal{R}$ be a ring, and let inverse elements in this ring be represented by $\mathcal{R}^{\times}$. Let $f = \sum_{i=0}^{n-1} f_i x^i \in \mathcal{R}$. Then we define the reduction of f modulo q to be $(f \bmod q) = \sum_{i=0}^{n-1} (f_i \bmod q) x^i \in \mathcal{R}_q$. Let $d \in \mathbb{N}$ and $c \in \mathbb{Z}$. For an even (odd) modulus $m \in \mathbb{Z}_{\geq 0}$, define $c' = c \bmod^{\pm} m$ as the unique element $-m/2 < c' \leq m/2$ (resp. $-\lfloor m/2 \rfloor < c' \leq \lfloor m/2 \rfloor$) such that $c' = c \bmod m$. As defined before, we denote by $[c]_L$ the unique integer in $(-2^{d-1}, 2^{d-1}] \subset \mathbb{Z}$ such that $c = [c]_L$ modulo 2^d; that is, $[c]_L = c \bmod^{\pm} 2^d$. Let $[\cdot]_M$ be the function $[\cdot]_M : \mathbb{Z} \rightarrow \mathbb{Z}, c \mapsto (c \bmod^{\pm} q - [c]_L)/2^d$. Furthermore, let $f = \sum_{i=0}^{n-1} f_i x^i \in \mathcal{R}_q$, for which $[f]_L = \sum_{i=0}^{n-1} [f_i]_L x^i$ and $[f]_M = \sum_{i=0}^{n-1} [f_i]_M x^i$. Let $f \in \mathcal{R}_q$ be a polynomial with coefficients being ordered (without losing any generality) as $|f_1| \geq |f_2| \geq ... \geq |f_n|$. Then we define $\max_i(f) = f_i$. Parameter $b_{\mathsf{GenA}} \in \mathbb{Z}_{>0}$ represents the number of blocks requested to cSHAKE128 in a first call during generation of the public polynomials $a_1, \ldots, a_k$.

In the remainder, we write $f = \sum_{i=0}^{n-1} f_i x^i$ to denote a polynomial in $\mathcal{R}_q$ (i.e., a polynomial of degree at most $n-1$ with coefficients from the ring $\mathbb{Z}_q$). We use the same symbol f to also denote the coefficient vector $f = (f_0, f_1, \ldots, f_{n-1}) \in \mathbb{Z}_q^n$. In some instances we represent polynomials as f_j (e.g., to represent polynomials $a_1, \ldots, a_k$). In these cases, we write $f_j = \sum_{i=0}^{n-1} f_{j,i} x^i$, and the corresponding vector representation is given by $f_j = (f_{j,0}, f_{j,1}, \ldots, f_{j,n-1}) \in \mathbb{Z}_q^n$. Likewise, s-bit strings $r \in \{0,1\}^s$ and $r' \in \{-1,0,1\}^s$ are written as vectors over the sets $\{0,1\}$ and $\{-1,0,1\}$ (resp.), in which an element in the i-th position is represented by r_i (r'_i, resp.). Multiple instances of the same set are represented by appending an additional superscript. For example, $\{0,1\}^{s,t}$ corresponds to t s-bit strings each defined over the set $\{0,1\}$.

The centered discrete Gaussian distribution for $x \in \mathbb{Z}$ with standard deviation σ is defined to be $\mathcal{D}_\sigma = \rho_\sigma(x)/\rho_\sigma(\mathbb{Z})$, where $\sigma > 0$, $\rho_\sigma(x) = \exp(\frac{-x^2}{2\sigma^2})$, and $\rho_\sigma(\mathbb{Z}) = 1 + 2\sum_{x=1}^{\infty} \rho_\sigma(x)$. We write $c \leftarrow_\sigma \mathbb{Z}$ to denote sampling of a value c with distribution $\mathcal{D}_\sigma$. For a polynomial $c \in \mathcal{R}$, we write $c \leftarrow_\sigma \mathcal{R}$ to denote sampling each coefficient of c with distribution $\mathcal{D}_\sigma$. For a finite set S, we denote sampling the element s uniformly from S with $s \leftarrow_\$ S$.

qTESLA's algorithms for key generation, signing and signature verification are given in Algorithms 4, 7, and 8, respectively. The two subroutines checkE and checkS that are called during key generation are depicted in Algorithms 6 and 5, respectively.

Remark 1. *In an earlier description of* qTESLA*, we described the algorithms in a less general way using only one sample of the ring learning with errors problem. In particular, the public key consisted of* seed_a *(corresponding to a) and t, and the secret key consisted of the polynomials s and e, together with* seed_y *and* seed_a*.*

We define the following functions that are required in the implementation of qTESLA (refer to the specified sections for explicit details about their realization):

- The pseudorandom function $\mathsf{PRF}_1 : \{0,1\}^\kappa \to \{0,1\}^{\kappa,k+3}$. This function takes as input a seed pre-seed that is κ bits long and maps it to $(k+3)$ seeds of κ bits each (cf. Section 2.4.2).
- The pseudorandom function $\mathsf{PRF}_2 : \{0,1\}^\kappa \times \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$. This function takes as inputs the seed seed_y and the random value r, each κ bits long, and a message m and maps them to the seed rand of κ bits (cf. line 3 of Algorithm 7).
- The generation function of the public polynomials $a_1, ..., a_k$ $\mathsf{GenA} : \{0,1\}^\kappa \to \mathcal{R}_q^k$. This function takes as input the seed seed_a that is κ bits long and maps it to k polynomials $a_i \in \mathcal{R}_q$ (cf. Section 2.4.3).
- The Gaussian sampler function $\mathsf{GaussSampler} : \{0,1\}^\kappa \times \mathbb{Z} \to \mathbb{Z}$. This function

samples the polynomials $s, e_1, ..., e_k$ according to the Gaussian distribution $\mathcal{D}_\sigma$ taking as inputs a κ-bit seed rand and a nonce $S \in \mathbb{Z}_{>0}$ (cf. Section 2.4.4).

- The encoding function $\mathsf{Enc} : \{0,1\}^\kappa \rightarrow \{-1,0,1\}^{h,2}$. This function encodes a κ-bit hash value c' as a polynomial $c \in \mathbb{H}_{n,h}$. The polynomial c is in turn encoded as the two lists *post_list* and *sign_list* $\in \{-1,0,1\}^h$ containing the positions and signs of its nonzero coefficients, respectively (cf. Section 2.4.5).
- The sampling function of the polynomial y, $\mathsf{ySampler} : \{0,1\}^\kappa \times \mathbb{Z} \rightarrow \mathcal{R}_{q,[B]}$. This function samples a polynomial $y \in \mathcal{R}_{q,[B]}$ taking as inputs a κ-bit seed rand and a nonce $S \in \mathbb{Z}_{>0}$ (cf. Section 2.4.6).
- The hash function $\mathsf{H} : \mathcal{R}_q^k \times \{0,1\}^* \rightarrow \{0,1\}^\kappa$. This function takes as inputs k polynomials $v_1, ..., v_k \in \mathcal{R}_q$ and computes $[v_1]_M, ..., [v_k]_M$. The result is then hashed together with a message m to a string κ bits long (cf. Section 2.4.7).

Algorithm 4 qTESLA's key generation

```
Require: -
Ensure: secret key $sk = (s, e_1, ..., e_k, \mathsf{seed}_a, \mathsf{seed}_y)$, and public key $pk = (\mathsf{seed}_a, t_1, ..., t_k)$
pre-seed $\leftarrow_\$ \{0,1\}^\kappa$
seed$_s$, seed$_{e_1}$, ..., seed$_{e_k}$, seed$_a$, seed$_y$ $\leftarrow \mathsf{PRF}_1$(pre-seed)      [Algorithm 9]
$a_1, ..., a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$                                     [Algorithm 10]
do
    $s \leftarrow_\sigma \mathcal{R}$                                                    [Algorithm 11]
while checkS($s$) $\neq 0$                                                 [Algorithm 5]
for $i = 1, ..., k$ do
    do
        $e_i \leftarrow_\sigma \mathcal{R}$                                              [Algorithm 11]
    while checkE($e_i$) $\neq 0$                                           [Algorithm 6]
    $t_i \leftarrow a_i s + e_i \mod q$
end for
$sk \leftarrow (s, e_1, ..., e_k, \mathsf{seed}_a, \mathsf{seed}_y)$
$pk \leftarrow (\mathsf{seed}_a, t_1, ..., t_k)$
return $sk$, $pk$
```

Algorithm 5 checkS: simplifies the security reduction by ensuring that $\|sc\|_\infty \leq L_S$.

Require: $s \in \mathcal{R}$
Ensure: $\{0,1\}$ ▷ true, false

```
if $\sum_{i=1}^{h} \max_i(s) > L_S$ then
    return 1
end if
return 0
```

Algorithm 6 checkE: ensures correctness of the scheme by checking that $\|ec\|_\infty \leq L_E$.

Require: $e \in \mathcal{R}$
Ensure: $\{0,1\}$ ▷ true, false

```
if $\sum_{i=1}^{h} \max_i(e) > L_E$ then
    return 1
end if
return 0
```

Algorithm 7 `qTESLA`'s signature generation

Require: message m, and secret key $sk = (s, e_1, ..., e_k, \mathsf{seed}_a, \mathsf{seed}_y)$
Ensure: signature (z, c')

```
 1: counter $\leftarrow 0$
 2: $r \leftarrow_\$ \{0,1\}^\kappa$
 3: $\mathsf{rand} \leftarrow \mathsf{PRF}_2(\mathsf{seed}_y, r, m)$
 4: $y \leftarrow \mathsf{ySampler}(\mathsf{rand}, \text{counter})$                    [Algorithm 14]
 5: $a_1, ..., a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$                        [Algorithm 10]
 6: for $i = 1, ..., k$ do
 7:     $v_i = a_i y \bmod^{\pm} q$
 8: end for
 9: $c' \leftarrow \mathsf{H}([v_1]_M, ..., [v_k]_M, m)$                   [Algorithm 15]
10: $c \leftarrow \mathsf{Enc}(c')$                                       [Algorithm 13]
11: $z \leftarrow y + sc$
12: if $z \notin \mathcal{R}_{q,[B-L_S]}$ then
13:     counter++
14:     Restart at step 4
15: end if
16: for $i = 1, ..., k$ do
17:     $w_i \leftarrow v_i - e_i c \bmod^{\pm} q$
18:     if $\|[w_i]_L\|_\infty \geq 2^{d-1} - L_E \vee \|w_i\|_\infty \geq \lfloor q/2 \rfloor - L_E$ then
19:         counter++
20:         Restart at step 4
21:     end if
22: end for
23: return $(z, c')$
```

Algorithm 8 `qTESLA`'s signature verification

Require: message m, signature (z, c'), and public key $pk = (\mathsf{seed}_a, t_1, ..., t_k)$
Ensure: $\{0, -1\}$ ▷ accept, reject signature

```
1: $c \leftarrow \mathsf{Enc}(c')$                                        [Algorithm 13]
2: $a_1, ..., a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$               [Algorithm 10]
3: for $i = 1, ..., k$ do
4:     $w_i \leftarrow a_i z - t_i c \bmod^{\pm} q$
5: end for
6: if $z \notin \mathcal{R}_{q,[B-L_S]} \vee c \neq \mathsf{H}([w_1]_M, ..., [w_k]_M, m)$ then
7:     return $-1$
8: end if
9: return 0
```

2.3 Correctness of the scheme

To establish the correctness of `qTESLA` we need to prove that the nonce input to the hash function H at signing (line 9 of Algorithm 7) is the same as the nonce input to the hash function H at verification (line 6 of Algorithm 8). That is, we need to prove that, for genuine signatures, $[ay \bmod^{\pm} q]_M = [az - tc \bmod^{\pm} q]_M = [a(y+sc) - (as+e)c \bmod^{\pm} q]_M = [ay + asc - asc - ec \bmod^{\pm} q]_M = [ay - ec \bmod^{\pm} q]_M$. From the definition of $[\cdot]_M$, this means proving that $(ay \bmod^{\pm} q - [ay \bmod^{\pm} q]_L)/2^d = (ay - ec \bmod^{\pm} q - [ay - ec \bmod^{\pm} q]_L)/2^d$, or simply $[ay \bmod^{\pm} q]_L = ec + [ay - ec \bmod^{\pm} q]_L$.

The above equality must hold component-wise, so let us prove the corresponding property for individual integers.

Assume that for integers α and ε it holds that $|[\alpha - \varepsilon \bmod^{\pm} q]_L| < 2^{d-1} - L_E$, $|\varepsilon| \leq L_E < \lfloor q/2 \rfloor$, $|\alpha - \varepsilon \bmod^{\pm} q| < \lfloor q/2 \rfloor - L_E$, and $-\lfloor q/2 \rfloor < \alpha \leq \lfloor q/2 \rfloor$ (i.e., $\alpha \bmod^{\pm} q = \alpha$). Then, we need to prove that

$$[\alpha]_L = \varepsilon + [\alpha - \varepsilon \bmod^{\pm} q]_L. \tag{1}$$

Proof. To prove equation (1), start by noticing that $|\varepsilon| \leq L_E < 2^{d-1}$ implies $[\varepsilon]_L = \varepsilon$. Thus, from $-2^{d-1} + L_E < [\alpha - \varepsilon \bmod^{\pm} q]_L < 2^{d-1} - L_E$ and $-L_E \leq [\varepsilon]_L \leq L_E$ it follows that

$$-2^{d-1} = -2^{d-1} + L_E - L_E < [\varepsilon]_L + [\alpha - \varepsilon \bmod^{\pm} q]_L < 2^{d-1} - L_E + L_E = 2^{d-1},$$

and therefore

$$[[\varepsilon]_L + [\alpha - \varepsilon \bmod^{\pm} q]_L]_L = [\varepsilon]_L + [\alpha - \varepsilon \bmod^{\pm} q]_L = \varepsilon + [\alpha - \varepsilon \bmod^{\pm} q]_L. \tag{2}$$

Next we prove that

$$[[\varepsilon]_L + [\alpha - \varepsilon \bmod^{\pm} q]_L]_L = [\alpha]_L. \tag{3}$$

We note that since $|\varepsilon| \leq L_E < \lfloor q/2 \rfloor$ it holds that $[\varepsilon]_L = [\varepsilon \bmod^{\pm} q]_L$. It holds further that

$$
\begin{aligned}
& [[\varepsilon \bmod^{\pm} q]_L + [\alpha - \varepsilon \bmod^{\pm} q]_L]_L && (4)\\
= \; & ((\varepsilon \bmod^{\pm} q) \bmod^{\pm} 2^d + (\alpha - \varepsilon \bmod^{\pm} q) \bmod^{\pm} 2^d) \bmod^{\pm} 2^d && (5)\\
& \text{by the definition of } [\cdot]_L && \\
= \; & (\varepsilon \bmod^{\pm} q + (\alpha - \varepsilon \bmod^{\pm} q)) \bmod^{\pm} 2^d. && (6)
\end{aligned}
$$

Since $|\varepsilon| \leq L_E$ and $|\alpha - \varepsilon \bmod^{\pm} q| < \lfloor q/2 \rfloor - L_E$, it holds that $|\alpha - \varepsilon| + |\varepsilon| < (\lfloor q/2 \rfloor - L_E) + L_E = \lfloor q/2 \rfloor$. Hence, equation (6) is the same as

$$
\begin{aligned}
= \; & (\varepsilon + \alpha - \varepsilon \bmod^{\pm} q) \bmod^{\pm} 2^d = (\alpha \bmod^{\pm} q) \bmod^{\pm} 2^d = \alpha \bmod^{\pm} 2^d \\
= \; & [\alpha]_L.
\end{aligned}
$$

Combining equations (2) and (3) we deduce that $[\alpha]_L = \varepsilon + [\alpha - \varepsilon \bmod^{\pm} q]_L$, which is the equation we needed to prove. □

Now define $\alpha := (ay)_i$ and $\varepsilon := (ec)_i$. From line 18 of Algorithm 7, we know that $\|[ay - ec]_L\|_\infty < 2^{d-1} - L_E$ and $\|ay - ec\|_\infty < \lfloor q/2 \rfloor - L_E$ for a valid signature, and that Algorithm 4 (line 10) guarantees $\|ec\|_\infty \leq L_E$. Likewise, by definition it holds that $L_E < \lfloor q/2 \rfloor$; see Section 2.5. Finally, $v = ay$ is reduced $\bmod^{\pm} q$ in line 7 of Algorithm 7 and, hence, v is in the centered range $-\lfloor q/2 \rfloor < ay \leq \lfloor q/2 \rfloor$.

In conclusion, we get the desired condition for ring elements, $[ay]_L = ec + [ay - ec]_L$, which in turn means $[az - tc]_M = [ay]_M$ as argued above.

2.4 Realization of the required functions

2.4.1 Hash and pseudorandom functions

In addition to the hash function H and the pseudorandom functions PRF_1 and PRF_2, several functions that are used for the implementation of `qTESLA` require pseudorandom bit generation. This functionality is provided by so-called extendable output functions (XOF).

For the remainder, the format that we use to call a XOF is given by $\mathsf{XOF}(\mathsf{X}, \mathsf{L}, \mathsf{S})$, where X is the input string, L specifies the output length in bytes, and S specifies an optional domain separator [1].

Next, we summarize how XOFs are instantiated using SHAKE [31] and cSHAKE [42] in the different functions requiring hashing or pseudorandom bit generation.

[1] Specifically, the domain separator S is used with cSHAKE, but ignored when SHAKE is used.

- PRF_1: the XOF is instantiated with SHAKE128 (resp. SHAKE256) for parameter sets qTESLA-I and qTESLA-p-I (resp. for other parameter sets); cf. Algorithm 9.
- PRF_2: the same as PRF_1.
- GenA: the XOF is instantiated with cSHAKE128 (cf. Algorithm 10).
- GaussSampler: the XOF is instantiated with cSHAKE128 (resp. cSHAKE256) for parameter sets qTESLA-I and qTESLA-p-I (resp. for other parameter sets); cf. Algorithm 11.
- Enc: the XOF is instantiated with cSHAKE128 (cf. Algorithm 13).
- ySampler: the XOF is instantiated with cSHAKE128 (resp. cSHAKE256) for parameter sets qTESLA-I and qTESLA-p-I (resp. for other parameter sets); cf. Algorithm 14.
- Hash H: the hash is instantiated with cSHAKE128 (resp. cSHAKE256) for parameter sets qTESLA-I and qTESLA-p-I (resp. for other parameter sets); cf. Algorithm 15.

In the cases of the functions GenA, Enc, and H, implementations of qTESLA need to follow strictly the XOF specifications based on SHAKE/cSHAKE given above in order to be fully compatible. However, for the rest of the cases (i.e., PRF_1, PRF_2, and ySampler) users can opt for a different cryptographic PRF.

2.4.2 Pseudorandom bit generation of seeds

qTESLA requires the generation of seeds during key generation; see line 2 of Algorithm 4. These seeds are then used to produce the polynomials s, e_i, a_i and y. Specifically, these seeds are:

- seed_s, which is used to generate the polynomial s,
- seed_{e_i}, which are used to generate the polynomials e_i for $i = 1, \ldots, k$,
- seed_a, which is used to generate the polynomials a_i for $i = 1, \ldots, k$, and
- seed_y, which is used to generate the polynomial y.

The size of each of these seeds is κ bits. In the accompanying implementations, the seeds are generated by first calling the system random number generator (RNG) to produce a pre-seed of size κ bits at line 1 of Algorithm 4, and then expanding this pre-seed through Algorithm 9. As explained in Section 2.4.1, in this case the XOF function is instantiated with SHAKE in our implementations.

Algorithm 9 Seed generation

```
Require: $\mathsf{pre\text{-}seed} \in \{0,1\}^\kappa$
Ensure: $(\mathsf{seed}_s, \mathsf{seed}_{e_1}, ..., \mathsf{seed}_{e_k}, \mathsf{seed}_a)$, where each seed is $\kappa$ bits long
$\langle \mathsf{seed}_s \rangle \| \langle \mathsf{seed}_{e_1} \rangle \| \dots \| \langle \mathsf{seed}_{e_k} \rangle \| \langle \mathsf{seed}_a \rangle \| \langle \mathsf{seed}_y \rangle \leftarrow \mathsf{XOF}(\mathsf{pre\text{-}seed}, \kappa \cdot (k+3)/8)$, where each $\langle \mathsf{seed} \rangle \in \{0,1\}^\kappa$
return $(\mathsf{seed}_s, \mathsf{seed}_{e_1}, ..., \mathsf{seed}_{e_k}, \mathsf{seed}_a)$
```

2.4.3 Generation of $\mathbf{a_1, ..., a_k}$

In qTESLA, the polynomials $a_1, ..., a_k$ are freshly generated per secret/public keypair using the seed seed_a during key generation; see line 3 of Algorithm 4. This seed is then stored as part of both the private and public keys so that the signing and verification operations can regenerate $a_1, ..., a_k$.

The approach above permits to save bandwidth since we only need κ bits to store seed_a instead of the $k \cdot n \cdot \lceil \log_2 q \rceil$ bits that are required to represent the full polynomials. Moreover, the use of fresh $a_1, ..., a_k$ per keypair makes the introduction of backdoors more difficult and reduces drastically the scope of all-for-the-price-of-one attacks [10, 16].

The procedure depicted in Algorithm 10 to generate $a_1, ..., a_k$ is as follows. The seed seed_a obtained from Algorithm 9 is expanded to $(\text{rate}_{\mathsf{XOF}} \cdot b_{\mathsf{GenA}})$ bytes using cSHAKE128, where $\text{rate}_{\mathsf{XOF}}$ is the SHAKE128 rate constant (i.e., the value 168 [31]) and b_{GenA} is a qTESLA parameter that represents the number of blocks requested in the first XOF call. The values of b_{GenA} for the different parameter sets were chosen as to allow the generation of slightly more bytes than are necessary to fill out all the coefficients of the polynomials (see Tables 2 and 3). Then, the algorithm proceeds to do rejection sampling over each $8b$-bit string of the cSHAKE output, discarding every package that has a value greater than the modulus q, where b is the number of bytes needed to represent q. Since there is a possibility that the cSHAKE output is exhausted before all the $k \cdot n$ coefficients are filled out, the algorithm permits successive (and as many as necessary) calls to the function requesting $\text{rate}_{\mathsf{XOF}}$ bytes each time (lines 9–12). The particular form of the expression to evaluate if more cSHAKE128 calls are necessary at line 9 facilitates high-performance implementations that evaluate up to *four* coefficients at a time in the if-loop of lines 5–8. It should be noted that the first call to cSHAKE128 uses the value $S = 0$ as domain separator. This value is incremented by one at each subsequent call.

Algorithm 10 Generation of public polynomials a_i, GenA

```
Require: $\mathsf{seed}_a \in \{0,1\}^{\kappa}$. Set $b = \lceil (\log_2 q)/8 \rceil$ and the SHAKE128 rate constant $\mathrm{rate}_{\mathsf{XOF}} = 168$
Ensure: $a_i \in \mathcal{R}_q$ for $i = 1, \ldots, k$

$S \leftarrow 0$
$\langle c_0 \rangle \| \langle c_1 \rangle \| \ldots \| \langle c_T \rangle \leftarrow$ cSHAKE128($\mathsf{seed}_a, \mathrm{rate}_{\mathsf{XOF}} \cdot b_{\mathsf{GenA}}, S$), where each $\langle c_t \rangle \in \{0,1\}^{8b}$
$i \leftarrow 0$, $pos \leftarrow 0$
while $i < k \cdot n$ do
    if $q > c_{pos} \bmod 2^{\lceil \log_2 q \rceil}$ then
        $a_{\lfloor i/n \rfloor + 1, i - n \cdot \lfloor i/n \rfloor} \leftarrow c_{pos} \bmod 2^{\lceil \log_2 q \rceil}$, where a polynomial $a_x$ is interpreted as a vector of coefficients $(a_{x,0}, a_{x,1}, \ldots, a_{x,n-1})$
        $pos \leftarrow pos + 1$, $i \leftarrow i + 1$
    end if
    if $pos > (\mathrm{rate}_{\mathsf{XOF}} \cdot b_{\mathsf{GenA}} - 4 \cdot b)$ then
        $S \leftarrow S + 1,\ pos \leftarrow 0$
        $\langle c_0 \rangle \| \langle c_1 \rangle \| \ldots \| \langle c_T \rangle \leftarrow$ cSHAKE128($\mathsf{seed}_a, \mathrm{rate}_{\mathsf{XOF}}, S$), where each $\langle c_t \rangle \in \{0,1\}^{8 \cdot b}$
    end if
end while
return $(a_1, \ldots, a_k)$
```

The procedure to generate $a_1, ..., a_k$ produces polynomials with uniformly random coefficients. Thus, following a standard practice, qTESLA assumes that the resulting polynomials $a_1, ..., a_k$ from Algorithm 10 are in the NTT domain. This permits an important speedup of some of the polynomial arithmetic operations. We remark that, although this assumption does not affect the security of the scheme, it does affect the correctness. In other words, implementations of the scheme need to follow this specification in order to be compatible with the qTESLA design. In particular, polynomial multiplications with these polynomials need to consider that they are expressed in NTT domain. For example, in the accompanying implementations multiplications in line 11 of Algorithm 4, line 7 of Algorithm 7 and line 4 of Algorithm 8 are done by computing $a_i \cdot b = \mathsf{NTT}^{-1}(a_i \circ \mathsf{NTT}(b))$ for some polynomial $b \in \mathcal{R}_q$ (see Section 2.4.8 for details about the NTT computations).

2.4.4 Gaussian sampling

One of the advantages of qTESLA is that Gaussian sampling is only required during key generation to sample $e_1, ..., e_k$, and s (see Alg. 4). Nevertheless, certain applications might still require an efficient and secure implementation of key generation and one that is, in particular, protected against timing and cache side-channel attacks. In that direction, we adopt the "constant-time" Gaussian sampler proposed in [16], which is an improvement upon the sampler proposed by Ducas *et al.* [27, Section 6]. We give the pseudocode of the proposed Gaussian sampler in Algorithms 11 and 12.

Algorithm 11 Sampling D_σ, GaussSampler

Require: seed rand $\in \{0,1\}^\kappa$, and nonce $S \in \mathbb{Z}_{>0}$. Set $\xi \in \mathbb{Z}$, the computer wordsize w, and the precomputed CDT table.

Ensure: $z \in \mathbb{Z}$ according to D_σ

```
 1: $S \leftarrow S \cdot 2^8$
 2: do
 3:     do
 4:         $y \leftarrow \mathsf{XOF}(\mathsf{rand}, \mathrm{sizeof}(\xi), S)$
 5:         $S \leftarrow S + 1$
 6:     while $y < \xi - 1$
 7:     $r \leftarrow \mathsf{XOF}(\mathsf{rand}, w/8, S)$
 8:     $S \leftarrow S + 1$
 9:     $x \leftarrow 0$
10:     while $r < \mathrm{CDT}[x]$ do
11:         $x \leftarrow x + 1$
12:     end while
13:     $z \leftarrow \xi \cdot x + y$
14:     $r \leftarrow \mathsf{XOF}(\mathsf{rand}, w/8, S)$
15:     $S \leftarrow S + 1$
16: while $(\mathsf{BerSampler}(r, y(y + 2\xi x)) = 0)$
17: Generate a random bit $b$
18: if $z = 0 \wedge b = 0$ then
19:     Restart at step 3
20: end if
21: Generate a random bit $b$
22: return $(-1)^b \cdot z$
```

In our implementations we target $w = 64$. The precomputed CDT table for this case is generated by the Magma script `gaussSigma2Sample_table.magma` found in the folder `\Supporting_Documentation\Scripts_for_Gaussian_sampler`.

Algorithm 12 Sampling $\mathcal{B}_{exp(-t/(2\sigma^2))}$ for $t \in [0, 2^w)$, BerSampler

Require: $r \in \{0,1\}^w$ and $t \in [0, 2^w)$. Set the computer wordsize w, and the precomputed Bernoulli table $\mathcal{B}$, which consists of $\mathcal{B}_{tables}$ sub-tables with $\mathcal{B}_{rows}$ rows each.

```
$c \leftarrow 2^{w-2}$
$s \leftarrow t$
for $i = 0, 1, \ldots, \mathcal{B}_{tables} - 1$ do
    $c \leftarrow c \cdot \mathcal{B}_{i, s \bmod \mathcal{B}_{rows}}$, where an element at the $j$-th row of the $i$-th sub-table is represented by $\mathcal{B}_{i,j}$
    $s \leftarrow s/\mathcal{B}_{rows}$
end for
if $r \bmod (w-1) \geq \lfloor c \rceil$ then
    return 0
else
    return 1
end if
```

In our implementations we target $w = 64$. For our parameters, we use $\mathcal{B}_{tables} = 3$ and $\mathcal{B}_{rows} = 32$. The precomputed Bernoulli table $\mathcal{B}$ for this case is generated by the Magma script `gaussBernoulliSample_table.magma` found in the folder `\Supporting_Documentation\Scripts_for_Gaussian_sampler`.

The basic idea behind the Gaussian sampler by Ducas *et al.* [27, Algorithms 10–12] is to start from a distribution that approximates the desired Gaussian distribution. Namely, Ducas *et al.* use a stepwise uniform distribution where the steps have width ξ and height distributed according to a ξ-scaled binary Gaussian that can be implemented efficiently. The binary Gaussian sampler, which was introduced by Ducas *et al.*, is a discrete Gaussian with specific variance $\sigma_2 = \frac{1}{\sqrt{2 \ln 2}} \approx 0.849$.

We follow the original Gaussian sampler, which focuses only on the positive half of D_{σ_2} denoted by $D^+_{\sigma_2} = \{x \leftarrow D_{\sigma_2} : x \geq 0\}$. From there, a Gaussian distribution D_σ with the desired quality is obtained by rejection sampling guided by Bernoulli distributions $\mathcal{B}_\rho$ with parameter ρ related to the standard deviation σ of the desired Gaussian distribution. Ducas *et al.* implement those Bernoulli distributions by decomposing them into ℓ certain base distributions $(\mathcal{B}_{\rho_0}, \mathcal{B}_{\rho_1}, \ldots, \mathcal{B}_{\rho_{\ell-1}})$ where the ρ constants are precomputed to the desired accuracy, and then sampling from those base distributions to that accuracy. Even though this Bernoulli decomposition is reportedly quite efficient, its running time highly depends on the private bits. Besides that, each $\mathcal{B}_{c_\rho}$ must be sampled to the same precision as the target distribution, which is why the total amount of entropy needed to obtain one Gaussian sample is much higher than theoretically necessary, roughly $O(\ell\lambda)$ bits rather than $O(\lambda)$ for security level λ.

Since `qTESLA` only needs a basic Gaussian sampler for key generation, it is possible to use the much simpler construction explained in [16]. In particular, only one kind of Bernoulli distribution $\mathcal{B}_\rho$ is needed in our case, namely, a distribution $\mathcal{B}_{\exp(-t/2\sigma^2)}$ where t is an ℓ-bit integer.[2] Hence, we can simplify the sampler by Ducas *et al.* by just computing the exponential bias $\rho = \exp(-t/2\sigma^2)$ using well-known exponentiation techniques, where ρ is an approximation of a real number in the interval $[0, 1]$ to the desired precision according to the security level λ. Specifically, we need to perform one uniform sample $y \leftarrow_\$ \mathbb{Z}$ in $\{0, \ldots, \xi - 1\}$ (lines 3–6 of Algorithm 11), one binary Gaussian sample $x \in \mathbb{Z}$ according to $D^+_{\sigma_2}$ (lines 7–12 of Algorithm 11) and, finally, one uniform sample to the desired precision based on the bias ρ with $t = y(y+2\xi x)$ (Algorithm 12 invoked from line 16 of Algorithm 11). At the end of this process we have the desired Gaussian sample with $\sigma = \xi\sigma_2 = \frac{\xi}{\sqrt{2\ln 2}}$, and the total entropy consumption is $O(\lambda)$ bits.

In the accompanying implementations, we implement an optimized version of Algorithm 11 that has been finely tuned for a computer wordsize $w = 64$. The precomputed tables that are used can be generated with the scripts provided in the folder `\Supporting_Documentation\Scripts_for_Gaussian_sampler`. In order to make the Gaussian sampler constant-time in our implementations, we make sure that basic operations such as comparisons are not implemented with conditional jumps that depend on secret data, and that lookup tables are always fully scanned at each pass. These modifications are straightforward in the case of `qTESLA`, given the significantly simpler Gaussian sampler algorithm. As explained in Section 2.4.1, our implementations use cSHAKE as the XOF function.

2.4.5 Encoding function

In the signature generation we need to map the hash input $([v_1]_M, ..., [v_k]_M, m)$ to a polynomial $c \in \mathbb{H}_{n,h} \subset \mathcal{R}_q$ (cf. line 9 and 10 of Algorithm 7). In order to obtain smaller signatures $(z, c') \in \{0,1\}^\kappa \times \mathcal{R}_q$, we break up this operation into $\mathsf{Enc}(\mathsf{H}([v_1]_M, ..., [v_k]_M, m)) = \mathsf{Enc}(c') = c$, where the encoding function Enc takes the output of the hash function H and maps it to a vector with entries in $\{-1, 0, 1\}$ of length n and weight h (i.e., it has h entries that are either 1 or -1). The vector represents a polynomial of degree $n - 1$.

We implement Enc as in [1] which in turn follows an algorithm originally proposed in [30, Section 4.4]; see Algorithm 13. Basically, the algorithm maps the output of the hash function H to a vector c by using values generated uniformly at random to fix the positions and signs of the h nonzero entries. The outputs are the two list arrays *pos_list* and *sign_list*, which contain the positions and signs of the nonzero entries of c, respectively. Accordingly, the algorithm first requests $\text{rate}_{\mathsf{XOF}}$ bytes from a XOF, and the output stream is interpreted as an array of 3-byte packets in little endian format. The lowest two bytes in every packet

[2] The Gaussian sampler originally designed for the signature scheme BLISS has the additional complication that Bernoulli distributions with inverse hyperbolic cosine biases $\mathcal{B}_{1/\cosh(t/f)}$ are required as well.

is used to calculate a position *pos* of a nonzero element in the n-dimension vector c. The third byte is used to determine its sign as follows. If the (pos)-th element of c is “empty” (indicated by the value 0), then the sign entry is filled out with 1 (+) if the least significant bit of the third byte is 0 which is interpreted as a positive element 1 of c. Otherwise, if the least significant bit of the third byte is 1 then the sign entry is set to -1 (–), which means that a bit 1 is interpreted as a negative coefficient -1 of c. These procedure is executed until h nonzero entries are filled out. If the XOF output is exhausted before completing the task then additional calls are invoked, requesting $\text{rate}_{\mathsf{XOF}}$ bytes each time. It should be noted that the first call to the XOF uses the value $S = 0$ as domain separator. This value is incremented by one at each subsequent call. As explained in Section 2.4.1, `qTESLA` uses cSHAKE128 as the XOF function.

Algorithm 13 Encoding function, Enc

Require: $c' \in \{0,1\}^{\kappa}$
Ensure: lists $pos\_list \in \{0, ..., n-1\}^h$ and $sign\_list \in \{-1,1\}^h$ containing the positions and signs, resp., of the nonzero elements of $c \in \mathbb{H}_{n,h}$

```
$S \leftarrow 0$, $cnt \leftarrow 0$
$\langle r_0 \rangle \| \langle r_1 \rangle \| \ldots \| \langle r_T \rangle \leftarrow \text{cSHAKE128}(c', \text{rate}_{\mathsf{XOF}}, S)$, where each $\langle r_t \rangle \in \{0,1\}^8$
$i \leftarrow 0$
Set all coefficients of $c$ to 0
while $i < h$ do
    $pos \leftarrow (r_{cnt} \cdot 2^8 + r_{cnt+1}) \bmod 2^n$
    $cnt \leftarrow cnt + 2$
    if $c_{pos} = 0$ then
        if $r_{cnt} \bmod 2 = 1$ then
            $c_{pos} \leftarrow -1$
        else
            $c_{pos} \leftarrow 1$
        end if
        $pos\_list_i \leftarrow pos$
        $sign\_list_i \leftarrow c_{pos}$
        $i \leftarrow i + 1$, $cnt \leftarrow cnt + 1$
    end if
    if $cnt > (\text{rate}_{\mathsf{XOF}} - 3)$ then
        $S \leftarrow S + 1$, $cnt \leftarrow 0$
        $\langle r_0 \rangle \| \langle r_1 \rangle \| \ldots \| \langle r_T \rangle \leftarrow \text{cSHAKE128}(c', \text{rate}_{\mathsf{XOF}}, S)$, where each $\langle r_t \rangle \in \{0,1\}^8$
    end if
end while
return $c$
```

2.4.6 Sampling of y

The sampling of the polynomial y at line 4 of Algorithm 7 can be simply performed by generating n $(\log_2 B + 1)$-bit values uniformly at random, and then correcting each value to the range $[-B, B]$. Algorithm 14 depicts the procedure used in the accompanying implementations based on a XOF. In this case, the value $S \cdot 2^8$, for $S \in \mathbb{Z} > 0$, is used as domain separator or nonce. In order to provide proper domain separation, the first invocation to sample y in Algorithm 7 is done with S initialized at 1, and then each subsequent invocation increases S by 1. This means that the successive calls to the sampler use as nonces $2^8, 2 \cdot 2^8, 3 \cdot 2^8$, and so on, providing domain separation between the sampling of y and other uses of the XOF in the signing algorithm. As explained in Section 2.4.1, our implementations use cSHAKE as the XOF function.

Algorithm 14 receives as input a seed rand which is the result of hashing seed_y, a random value r and the message m, as can be seen at line 3 of Algorithm 7.

Algorithm 14 Sampling y, $\mathsf{ySampler}$

Require: seed $\mathsf{rand} \in \{0,1\}^\kappa$ and nonce $S \in \mathbb{Z}_{>0}$. Set $b = \lceil(\log_2 B + 1)/8\rceil$
Ensure: $y \in \mathcal{R}_{q,[B]}$

```
$pos \leftarrow 0$
$\langle y_0 \rangle \| \langle y_1 \rangle \| \ldots \| \langle y_T \rangle \leftarrow \mathsf{XOF}(\mathsf{rand}, b \cdot n, S \cdot 2^8)$, where each $\langle y_t \rangle \in \{0,1\}^{8b}$
for $i = 0, 1, \ldots, n-1$ do
    $y_i \leftarrow y_i \bmod 2^{\lceil \log_2 B \rceil + 1}$
    $y_i \leftarrow y_i - B$
    $pos \leftarrow pos + b$
end for
return $y = (y_0, y_1, \ldots, y_{n-1}) \in \mathcal{R}_{q,[B]}$
```

2.4.7 Hash H

The hash H required by the signing and verification algorithms takes as inputs k polynomials $v_1, \ldots, v_k \in \mathcal{R}_q$ and computes $[v_1]_M, \ldots, [v_k]_M$. The result is hashed together with a message m to a string c' that is κ bits long. The procedure is depicted in Algorithm 15. Let each polynomial v_i be interpreted as a vector of coefficients $(v_{i,0}, v_{i,1}, \ldots, v_{i,n-1})$. We first compute $[v_{i,j}]_L$ for each coefficient $v_{i,j} \in (-q/2, q/2]$ following the details from Section 2.2: each coefficient is reduced modulo 2^d if it is greater than 2^{d-1}. This guarantees a result in the range $(-2^{d-1}, 2^{d-1}]$, as required by the definition of $[\cdot]_L$. Next, we compute $[v_{i,j}]_M$ as $(v_{i,j} - [v_{i,j}]_L)/2^d$ and store it as a byte, which in the end makes up a full string of $k \cdot n$ bytes. Finally, SHAKE is used to hash the resulting string concatenated with the message m to the κ-bit string c'.

Algorithm 15 Hash H

Require: polynomials $v_1, \ldots, v_k \in \mathcal{R}_q$, where $v_{i,j} \in (-q/2, q/2]$, for $i = 1, \ldots, k$ and $j = 0, \ldots, n-1$, and a message m of length $mlen$ bytes.
Ensure: $c' \in \{0,1\}^\kappa$

```
for $i = 1, 2, \ldots, k$ do
    for $j = 0, 1, \ldots, n-1$ do
        $\text{val} \leftarrow v_{i,j} \bmod 2^d$
        if $\text{val} > 2^{d-1}$ then
            $\text{val} \leftarrow \text{val} - 2^d$
        end if
        $w_{(i-1)\cdot n+j} \leftarrow (v_{i,j} - \text{val})/2^d$
    end for
end for
for $i = 0, 1, \ldots, mlen-1$ do
    $w_{k\cdot n+i} \leftarrow m_i$, where $m$ is interpreted as the byte array $\langle m_0\rangle \| \langle m_1\rangle \| \ldots \| \langle m_{mlen-1}\rangle$
end for
$c' \leftarrow \text{SHAKE}(w, \kappa/8)$, where $w$ is the byte array $\langle w_0\rangle \| \langle w_1\rangle \| \ldots \| \langle w_{k\cdot n+mlen-1}\rangle$
return $c' \in \{0,1\}^\kappa$
```

2.4.8 Polynomial multiplication and the number theoretic transform

Polynomial multiplication over a finite field is one of the fundamental operations in R-LWE based schemes such as qTESLA. In this setting, this operation can be efficiently carried out by satisfying the condition $q \equiv 1 \pmod{2n}$ and, thus, enabling the use of the Number Theoretic Transform (NTT).

Since qTESLA specifies the generation of the polynomials $a_1, \ldots, a_k$ directly in the NTT domain for efficiency purposes (see Section 2.4.3), we need to define polynomials in such a domain. Let ω be a primitive n-th root of unity in $\mathbb{Z}_q$, i.e., $\omega^n \equiv 1 \mod q$, and let ϕ be a primitive $2n$-th root of unity in $\mathbb{Z}_q$ such that $\phi^2 = \omega$. Then, given a polynomial $a = \sum_{i=0}^{n-1} a_i x^i$ the forward transform is defined as

$$\mathsf{NTT} : \mathbb{Z}_q[x]/\langle x^n + 1\rangle \to \mathbb{Z}_q^n, \;\; a \mapsto \tilde{a} = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} a_j \phi^j \omega^{ij} \right) x^i,$$

where $\tilde{a} = \mathsf{NTT}(a)$ is said to be in *NTT domain*. Similarly, the inverse transformation of a polynomial $\tilde{a}$ in NTT domain is defined as

$$\mathsf{NTT}^{-1} : \mathbb{Z}_q^n \to \mathbb{Z}_q[x]/\langle x^n + 1\rangle, \;\; \tilde{a} \mapsto a = \sum_{i=0}^{n-1} \left(n^{-1}\phi^{-i} \sum_{j=0}^{n-1} \tilde{a}_j \omega^{-ij} \right) x^i.$$

It then holds that $\mathsf{NTT}^{-1}(\mathsf{NTT}(a)) = a$ for all polynomials $a \in \mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n - 1\rangle$. The polynomial multiplication of a and $b \in \mathcal{R}_q$ can be performed as $a \cdot b = \mathsf{NTT}^{-1}(\mathsf{NTT}(a) \circ$

$\mathsf{NTT}(b)$), where $\cdot$ is the polynomial multiplication in $\mathcal{R}_q$ and $\circ$ is the coefficient wise multiplication in $\mathbb{Z}_q^n$.

In the accompanying implementations, we adopt butterfly algorithms to compute the NTT that efficiently merge the powers of ϕ and ϕ^{-1} with the powers of ω, and that at the same time avoid the need of a so-called bit-reversal operation which is required by some implementations [10, 51, 52]. Specifically, we use an algorithm that computes the forward NTT based on the Cooley-Tukey butterfly that absorbs the products of the root powers in bit-reversed ordering. This algorithm receives the inputs of a polynomial a in standard ordering and produces a result in bit-reversed ordering. Similarly, for the inverse NTT we use an algorithm based on the Gentleman-Sande butterfly that absorbs the inverses of the products of the root powers in the bit-reversed ordering. The algorithm receives the inputs of a polynomial $\tilde{a}$ in the bit-reversed ordering and produces an output in standard ordering. Efficient versions of these algorithms, which we follow for our implementations, can be found in [54, Algorithms 1 and 2].

Sparse multiplication. In our implementations, standard polynomial multiplications are carried out using the NTT as explained above. However, `qTESLA` also requires specialized multiplications with the polynomial c which by definition only contains h nonzero coefficients (cf. lines 11 and 17 in Algorithm 7, and line 4 in Algorithm 8). These *sparse multiplications* can be realized efficiently with the specialized polynomial algorithm depicted in Algorithm 16.

Algorithm 16 Sparse Polynomial Multiplication

Require: $a = \sum_{i=0}^{n-1} a_i x^i \in \mathcal{R}_q$ with $a_i \in \mathbb{Z}_q$, and list arrays $pos\_list \in \{0, ..., n-1\}^h$ and $sign\_list \in \{-1, 1\}^h$ containing the positions and signs, resp., of the nonzero elements of a polynomial $c \in \mathbb{H}_{n,h}$

Ensure: $f = a \cdot c \in \mathcal{R}_q$

```
Set all coefficients of $f$ to 0
for $i = 0, ..., h-1$ do
    $pos \leftarrow pos\_list_i$
    for $j = 0, ..., pos-1$ do
        $f_j \leftarrow f_j - sign\_list_i \cdot a_{j+n-pos}$
    end for
    for $j = pos, ..., n-1$ do
        $f_j \leftarrow f_j + sign\_list_i \cdot a_{j-pos}$
    end for
end for
return $f$
```

2.5 System parameters and parameter selection

In this section, we describe `qTESLA`'s system parameters and our explicit choice of parameter sets.

Parameter sets. Herein, we propose *five* parameter sets which were derived according to *two* different approaches (i) following a "heuristic" parameter generation, and (ii) following a "provably-secure" parameter generation according to a security reduction. The proposed parameter sets are displayed in Table 1 together with their targeted security category, as defined by NIST in [56].

Table 1: Parameter sets and their targeted security.

Heuristic	Provably-secure	Security category
`qTESLA-I`	`qTESLA-p-I`	NIST's category 1
`qTESLA-III-speed`, `qTESLA-III-size`	`qTESLA-p-III`	NIST's category 3

The proposed provably-secure parameter sets, namely `qTESLA-p-I` and `qTESLA-p-III`, were chosen according to the security reduction provided in Theorem 6, Section 5.1. This implies the following: by virtue of our security reduction, these parameters strictly correspond to an instance of the R-LWE problem. That is, the reduction provably guarantees that our scheme has the selected security level as long as the corresponding R-LWE instance is intractable. In other words, hardness statements for R-LWE instances have a provable consequence for the security levels of our scheme. Moreover, since the presented reduction is tight, the tightness gap of our reduction is equal to 1 for our choice of parameters and, hence, the concrete bit security of our signature scheme is essentially the same as the bit hardness of the underlying R-LWE instance.

Choosing parameters following the security statements, as described above, implies to follow specific security requirements and to take a reduction loss into account. This affects the performance and signature/key sizes of the scheme. In order to offer a more efficient approach, we also propose three parameter sets, namely `qTESLA-I`, `qTESLA-III-speed`, and `qTESLA-III-size`, which are chosen *heuristically*. In this case, we assume that the security level of an instantiation of the scheme by a certain parameter set directly corresponds to the hardness level of the instance of the underlying lattice problem that corresponds to those parameters, without taking into account the security reduction. The assumption is that Theorem 6 still holds for these concrete parameter sets.

The sage script that was used to generate the various parameters is included in the submission package (see the file `parameterchoice.sage` found in the submission folder `\Supporting_Documentation\Script_to_choose_parameters`).

Table 2: Description and bounds of all the system parameters.

Param.	Description	Requirement
λ	security parameter	-
q_h, q_s	number of hash and sign queries	-
n	dimension ($n-1$ is the poly. degree)	power-of-two
σ, ξ	standard deviation of centered discrete Gaussian distribution	$\sigma = \frac{\xi}{\sqrt{2 \ln 2}}$
k	#R-LWE samples	-
q	modulus	$q \equiv 1 \bmod 2n$, $q > 4B$ For provably secure parameters: $q^{nk} \geq \lvert\Delta\mathbb{S}\rvert \cdot \lvert\Delta\mathbb{L}\rvert \cdot \lvert\Delta\mathbb{H}\rvert$, $q^{nk} \geq 2^{4\lambda+nkd} 4 q_s^3 (q_s + q_h)^2$
h	# of nonzero entries of output elements of Enc	$2^h \cdot \binom{n}{h} \geq 2^{2\lambda}$
κ	output length of hash function H and input length of GenA, PRF_1, PRF_2, Enc and $\mathsf{ySampler}$	$\kappa \geq \lambda$
L_E, η_E	bound in checkE	$\eta_E \cdot h \cdot \sigma$
L_S, η_S	bound in checkS	$\eta_S \cdot h \cdot \sigma$
B	determines interval the randomness is chosen from during signing	$B \geq \frac{\sqrt[k \cdot n]{M} + 2L_S - 1}{2(1 - \sqrt[k \cdot n]{M})}$, near a power-of-two
d	number of rounded bits	$\left(1 - \frac{2 \cdot L_E + 1}{2^d}\right)^{k \cdot n} \geq 0.3$, $d > \log_2(B)$
b_{GenA}	number of blocks requested to SHAKE128 for GenA	$b_{\mathsf{GenA}} \in \mathbb{Z}_{>0}$
$\lvert\Delta\mathbb{H}\rvert$		$\sum_{j=0}^{h} \sum_{i=0}^{h-j} \binom{kn}{2i} 2^{2i} \binom{kn-2i}{j} 2^j$
$\lvert\Delta\mathbb{S}\rvert$	see definition in the text	$(4(B - L_S) + 1)^n$
$\lvert\Delta\mathbb{L}\rvert$		$(2^d + 1)^{nk}$
δ_z	acceptance probability of z in line 12 during signing	experimentally
δ_w	acceptance probability of w in line 18 during signing	experimentally
δ_{keygen}	acceptance probability of key pairs during key generation	experimentally
sig size	theoretical size of signature [byte]	$\kappa + n(\lceil \log_2(B - L_S) \rceil + 1)$
pk size	theoretical size of public key [byte]	$kn(\lceil \log_2(q) \rceil) + \kappa$
sk size	theoretical size of secret key [byte]	$n(k+1)(\lceil \log_2(t \cdot \sigma + 1) \rceil) + 2\kappa$ with $t = 11.6$ or 15

System parameters. Table 2 summarizes all the system parameters, including all relevant bounds. Concrete parameter values for each of the proposed parameter sets are compiled in Table 3.

Let λ be the security parameter, i.e., the targeted bit security of a given instantiation. Let $n \in \mathbb{Z}_{>0}$ be the dimension, such that $n-1$ is the polynomial degree. In the targeted R-LWE setting with $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1 \rangle$, the polynomial degree is set to a power of two, i.e., $n = 2^l$ for $l \in \mathbb{N}$. Let σ be the standard deviation of the centered discrete Gaussian distribution that is used to sample the coefficients of the secret and error polynomials. To use the fast Gaussian sampler described in Section 2.4.4, we choose $\sigma = \frac{\xi}{\sqrt{2 \ln 2}}$ for some $\xi \in \mathbb{Z}_{>0}$. Let $k \in \mathbb{Z}_{>0}$ be the number of ring learning with errors samples. For our *heuristic* parameter sets `qTESLA-I`, `qTESLA-III-speed`, and `qTESLA-III-size`, we fix

Table 3: Parameters for each of the proposed *heuristic* and *provably-secure* parameter sets with $q_h = 2^{128}$ and $q_s = 2^{64}$; we choose $M = 0.3$.

Param.	`qTESLA-I`	`qTESLA-III-speed`	`qTESLA-III-size`	`qTESLA-p-I`	`qTESLA-p-III`
λ	95	160	160	95	160
κ	256	256	256	256	256
n	512	1 024	1 024	1 024	2 048
σ, ξ	23.78, 27.9988	10.2, 12	8.49, 9.9962	8.5, 10	8.5, 10
k	1	1	1	4	5
q	4 205 569 $\approx 2^{22}$	8 404 993 $\approx 2^{23}$	4 206 593 $\approx 2^{22}$	485 978 113 $\approx 2^{29}$	1 129 725 953 $\approx 2^{30}$
h	30	48	48	25	40
L_E, η_E L_S, η_S	1 586, 2.223 1 586, 2.223	1 147, 2.34 1 233, 2.52	910, 2.23 910, 2.23	554, 2.61 554, 2.61	901, 2.65 901, 2.65
B	$2^{20} - 1$	$2^{21} - 1$	$2^{20} - 1$	$2^{21} - 1$	$2^{23} - 1$
d	21	22	21	22	24
b_{GenA}	19	38	38	108	180
$\lvert\Delta\mathbb{H}\rvert$ $\lvert\Delta\mathbb{S}\rvert$ $\lvert\Delta\mathbb{L}\rvert$	- - -	- - -	- - -	$\approx 2^{435.8}$ $\approx 2^{23551.6}$ $\approx 2^{94208.0}$	$\approx 2^{750.9}$ $\approx 2^{51199.7}$ $\approx 2^{256000.0}$
δ_w δ_z δ_{sign} δ_{keygen}	0.31 0.44 0.14 0.45	0.38 0.56 0.21 0.60	0.25 0.37 0.09 0.39	0.33 0.78 0.26 0.59	0.34 0.81 0.28 0.44
sig size pk size sk size	1 376 1 504 1 216	2 848 3 104 2 112	2 720 2 976 2 112	2 848 14 880 4 576	6 176 39 712 12 320
classical bit hardness quantum bit hardness	104 97	178 164	188 169	132 123	247 270

$k = 1$, whereas for our *provably-secure* parameter sets `qTESLA-p-I` and `qTESLA-p-III`, we choose $k > 1$. The latter choice allows us to reduce the size of the modulus q, as explained later. Depending on the specific function, the parameter κ defines the outputs or inputs of the hash function and pseudorandom functions described in Section 2.4.1. The parameter h defines the number of nonzero elements in the output of the encoding function described in Section 2.4.5.

The values L_E and L_S are used to bound the coefficients in the error and secret polynomials in the evaluation functions checkE and checkS, respectively. Bounding the size of those polynomials restricts the size of the key space; accordingly we compensate the security loss by choosing a larger bit hardness. Both bounds, L_E and L_S, impact the rejection probability during the signature generation as follows. If one increases the values of L_E

and L_S, the acceptance probability during key generation increases (lines 6 and 10 in Algorithm 4), while the acceptance probability during signature generation decreases (lines 12 and 18 in Algorithm 7). We determine the best trade-off between these two acceptance probabilities experimentally. We start choosing $L_E = \eta_E \cdot h \cdot \sigma$ (resp., $L_S = \eta_S \cdot h \cdot \sigma$) with $\eta_E = \eta_S = 2.8$ and try different values for $\eta_E, \eta_S \in [2.0, 3.0]$. Let $M = 0.3$ be a value of our choosing that determines (together with L_S and B) the acceptance probability of the rejection sampling in line 12 of Algorithm 7. The parameter B defines the interval of the random polynomial y (cf. line 4 of Algorithm 7) and it is determined by M and the parameter L_S as follows:

$$\left(\frac{2B - 2L_S + 1}{2B + 1}\right)^{k \cdot n} \geq M \Leftrightarrow B \geq \frac{\sqrt[k \cdot n]{M} + 2L_S - 1}{2(1 - \sqrt[k \cdot n]{M})}.$$

We select the rounding value d to be larger than $\log_2(B)$ and such that the acceptance probability of the check $\| [w]_L \|_\infty \geq 2^{d-1} - L_E$ in line 18 of Algorithm 7 is upper bounded by 0.7 when using the sage script to choose parameters. Changing the value L_E as described above, impacts the rejection probability of w as well. Experimentally, we determine the acceptance probability δ_z of z and δ_w of w during signing and the acceptance probability of key pairs δ_{keygen}. The results are summarized in Table 3.

Finally, $b_{\mathsf{GenA}} \in \mathbb{Z}_{>0}$ represents the number of blocks requested to cSHAKE128 in a first call during the generation of the public polynomials $a_1, \ldots, a_k$ (cf. Algorithm 10). The values of b_{GenA} for the different parameter sets were chosen as to allow the generation of (slightly) more bytes than are necessary to fill out all the coefficients of the polynomials $a_1, \ldots, a_k$.

The modulus q. The parameter q is the modulus in the R-LWE instance. Depending on whether parameters are chosen heuristically or according to our security reduction in Theorem 6, q is chosen to fulfill several bounds and assumptions that are motivated by the security reduction or efficient implementation requirements. To be able to use fast polynomial multiplication we choose q to be a prime integer such that $q \mod 2n = 1$.
To choose parameter sets according to the security reduction, it is first convenient to simplify our security statement. To this end we ensure that $q^{nk} \geq |\Delta\mathbb{S}| \cdot |\Delta\mathbb{L}| \cdot |\Delta\mathbb{H}|$ with the following definition of sets: $\mathbb{S}$ is the set of polynomials $z \in \mathcal{R}_{q,[B-U]}$ and $\Delta\mathbb{S} = \{z - z' \ : \ z, z' \in \mathbb{S}\}$, $\mathbb{H}$ is the set of polynomials $c \in \mathcal{R}_{q,[1]}$ with exactly h nonzero coefficients and $\Delta\mathbb{H} = \{c - c' \ : \ c, c' \in \mathbb{H}\}$, and $\Delta\mathbb{L} = \{x - x' : x, x' \in \mathcal{R} \text{ and } [x]_M = [x']_M\}$. Then, the following equation (cf. Theorem 6) has to hold:

$$\frac{2^{3\lambda + nkd} \cdot 4 \cdot q_s^3 (q_s + q_h)^2}{q^{nk}} \leq 2^{-\lambda} \Leftrightarrow q \geq \left(2^{4\lambda + nkd} \cdot 4 \cdot q_s^3 (q_s + q_h)^2\right)^{1/nk}.$$

Following the NIST's call for proposals [56, Section 4.A.4], we choose the number of classical queries to the sign oracle to be $q_s = 2^{64}$ for all our parameter sets. Moreover, we choose the number of queries of a hash function to be $q_h = 2^{128}$.

Key and signature sizes We are now in position to determine the key and signature sizes. The theoretical bit length of the signatures and public keys are given by $\kappa + n \cdot (\lceil \log_2(B - L_S) \rceil + 1)$ bits and $k \cdot n \cdot (\lceil \log_2(q) \rceil) + \kappa$ bits, respectively. To determine the size of the secret key we note that for $t > 0$ it holds that $Pr_{x \leftarrow_\sigma \mathbb{Z}} [|x| > t\sigma] \leq 2e^{-t^2/2}$. For example, for $t = 11.6$ and $t = 15$, the probability $Pr_{x \leftarrow_\sigma \mathbb{Z}} [|x| > t\sigma]$ is less or equal to 2^{-95} and 2^{-160}, respectively. Therefore, the theoretical size of the secret key is given by $n(k+1)(\lceil \log_2(t \cdot \sigma + 1) \rceil) + 2\kappa$ bits with $t = 11.6$ for `qTESLA-I` and `qTESLA-p-I`, and with $t = 15$ for `qTESLA-III-speed`, `qTESLA-III-size` and `qTESLA-p-III`. Table 3 details the key and signature sizes according to these theoretical estimates.

3 Performance analysis

3.1 Associated implementations

This document comes accompanied by simple yet efficient reference implementations written exclusively in portable C.

An important feature of `qTESLA` is that it enables very efficient implementations that can work for different security levels with minor changes. For example, our implementations of the `heuristic` `qTESLA` parameter sets `qTESLA-I`, `qTESLA-III-speed` and `qTESLA-III-size` share most of their codebase, and only differ in some packing functions and system constants that can be instantiated at compilation time. This feature is also shared between our implementations for the `provably-secure` `qTESLA` parameter sets `qTESLA-p-I` and `qTESLA-p-III`. This highlights the simplicity and scalability of software based on `qTESLA`.

Furthermore, since `provably-secure` `qTESLA` uses a generalization of the scheme with $k > 1$, it is possible to merge all the implementations into one. For our current implementations, we separate both approaches in order to maximize the efficiency of `heuristic qTESLA`. However, we envision applications in which this feature could be exploited with a relatively small performance overhead.

All our implementations avoid the use of secret address accesses and secret branches and, hence, are protected against timing and cache side-channel attacks.

3.2 Performance on x64 Intel

To evaluate the performance of the provided implementations, we ran our benchmarking suite on two machines powered by: (i) a 3.40 GHz Intel Core i7-6700 (Skylake) processor and (ii) a 3.40 GHz Intel Core i7-4770 (Haswell) processor, both running Ubuntu 16.04.3 LTS. As is standard practice, TurboBoost was disabled during the tests. For compilation we used gcc version 7.2.0 with the command `gcc -O3 -march=native -fomit-frame-pointer`.

Scheme	keygen	sign	verify	**total** (sign + verify)
qTESLA-I	1,582.8 (1,727.3)	467.4 (626.4)	98.8 (99.3)	566.2 (725.7)
qTESLA-III-speed	3,575.5 (3,873.1)	662.5 (865.6)	201.7 (202.0)	864.2 (1,067.6)
qTESLA-III-size	6,056.5 (6,284.4)	1,236.2 (1,714.3)	204.1 (204.5)	1,440.3 (1,918.8)
qTESLA-p-I	6,678.3 (6,880.1)	1,258.6 (1,590.2)	504.6 (505.2)	1,763.2 (2,095.4)
qTESLA-p-III	30,597.2 (32,572.8)	5,056.8 (6,241.5)	2,556.3 (2,555.9)	7,613.1 (8,797.4)

Table 4: Performance (in thousands of cycles) of the reference implementation of qTESLA on a 3.40 GHz Intel Core i7-6700 (Skylake) processor. Results for the median and average (in parenthesis) are rounded to the nearest 10^2 cycles. Signing is performed on a message of 59 bytes.

The results in Table 4 showcase the high performance of heuristic qTESLA: the (median) time of signing and verification on the Intel Skylake platform is of approximately 166.5, 254.2 and 423.6 microseconds for qTESLA-I, qTESLA-III-speed and qTESLA-III-size, respectively. Likewise, provably-secure qTESLA computes the same operations in approximately 0.52 and 2.24 milliseconds with qTESLA-p-I and qTESLA-p-III, respectively. This demonstrates that the speed of provably-secure qTESLA, although slower, is still practical for many applications. Similar results were observed in our tests on the Intel Haswell platform (see Table 5).

We highlight that these results are only obtained with generic C implementations. Future work includes the development of optimized implementations exploiting assembly and vector instructions to get further performance improvements.

Scheme	keygen	sign	verify	total (sign + verify)
qTESLA-I	1, 656.7 (1, 851.2)	512.9 (714.2)	106.3 (106.7)	619.2 (820.9)
qTESLA-III-speed	3, 706.8 (4, 120.4)	804.3 (1, 109.8)	238.0 (241.2)	1, 042.3 (1, 351.0)
qTESLA-III-size	6, 161.8 (6, 553.7)	1, 402.2 (1, 992.9)	221.1 (226.7)	1, 623.3 (2, 219.6)
qTESLA-p-I	6, 857.1 (7, 193.6)	1, 348.4 (1, 704.5)	544.2 (544.8)	1, 892.6 (2, 249.3)
qTESLA-p-III	31, 903.5 (34, 488.4)	5, 488.1 (7, 127.2)	2, 715.4 (2, 783.9)	8, 203.5 (9, 911.1)

Table 5: Performance (in thousands of cycles) of the reference implementation of qTESLA on a 3.40 GHz Intel Core i7-4770 (Haswell) processor. Results for the median and average (in parenthesis) are rounded to the nearest 10^2 cycles. Signing is performed on a message of 59 bytes.

4 Known answer values

The submission includes KAT values with tuples that contain message size (mlen), message (msg), public key (pk), secret key (sk), signature size (smlen) and signature (sm) values for all the proposed parameter sets.

The KAT files can be found in the media folder:

- qTESLA-I: \KAT\PQCsignKAT_qTesla-I.rsp,
- qTESLA-III-speed: \KAT\PQCsignKAT_qTesla-III-speed.rsp,
- qTESLA-III-size: \KAT\PQCsignKAT_qTesla-III-size.rsp,
- qTESLA-p-I: \KAT\PQCsignKAT_qTesla-p-I.rsp, and
- qTESLA-p-III: \KAT\PQCsignKAT_qTesla-p-III.rsp.

5 Expected security strength

It this section we discuss the expected security strength of and possible attacks against qTESLA. This includes two statements about the theoretical security and the parameter choices depending on them. To this end we first define the hardness assumptions qTESLA is based on. This includes the ring short integer solution (R-SIS) problem and the *decisional* ring learning with errors (decisional R-LWE) problem.

Definition 2 (Ring short integer solution problem R-SIS$_{n,k,q,\beta}$)**.** *Let* $a_1, ..., a_k \leftarrow_{\$} \mathcal{R}_q$. *The ring short integer solution problem* $R\text{−}SIS_{n,k,q,\beta}$ *is to find solutions* $u_1, ..., u_{k+1} \in \mathcal{R}_q$, *where* $u_i \neq 0$ *for at least one* i, *such that* $(a_1, ..., a_k, 1) \cdot (u_1, ..., u_{k+1})^T = a_1u_1 + ... + a_ku_k + u_{k+1} = 0 \mod q$ *and* $\|u_1\|, ..., \|u_{k+1}\| \leq \beta$.

Definition 3 (Learning with Errors Distribution)**.** *Let* $n, q > 0$ *be integers,* $s \in \mathcal{R}$, *and* χ *be a distribution over* $\mathcal{R}$. *We define by* $\mathcal{D}_{s,\chi}$ *the 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 4 (Decisional Ring Learning with Errors Problem R-LWE$_{n,k,q,\chi}$)**.** *Let* $n, q > 0$ *be integers and* χ *be a distribution over* $\mathcal{R}$. *Moreover, let* $s \leftarrow \chi$ *and* $\mathcal{D}_{s,\chi}$ *be the learning with errors distribution. Given* k *tuples* $(a_1, t_1), ..., (a_k, t_k)$, *the decisional ring learning with errors problem R-LWE*$_{n,k,q,\chi}$ *is to distinguish whether* $(a_i, t_i) \leftarrow \mathcal{U}(\mathcal{R}_q \times \mathcal{R}_q)$ *or* $(a_i, t_i) \leftarrow \mathcal{D}_{s,\chi}$ *for all* i.

5.1 Provable security in the (quantum) random oracle model

The security of qTESLA instantiated with the *provably-secure* parameters sets (see Section 2.5) is supported by *two* statements reducing the hardness of lattice-based assumptions to the security of our proposed signature scheme in the (quantum) random oracle model. In this subsection, we describe these two statements. Note that formal security proofs are not included in this document because these are very close to the original results. The interested reader is referred to [9, 14] for more details.

The first reduction (cf. Theorem 5), which follows closely the approach proposed by Bai and Galbraith [14], gives a non-tight reduction from R-LWE and R-SIS to the existentially unforgeability under chosen-message attack (EUF-CMA) of qTESLA in the random oracle model.

Theorem 5. *Let* $2^n \cdot \binom{n}{h} \geq 2^\lambda$, $(2R + 1)^{k+1} \geq kq^n2^\kappa$, *and* $q > 4B$. *If there exists an adversary* A *that forges a signature of the signature scheme* qTESLA *described in Section 2.2 in time* t_Σ *and with success probability* ϵ_Σ, *then there exists a reduction* R *that solves either*

- *the* $R\text{−}LWE_{n,k,q,\sigma}$ *problem in time* $t_{LWE} \approx t_\Sigma$ *with* $\epsilon_{LWE} \geq \epsilon_\Sigma/2$, *or*
- *the* $R\text{−}SIS_{n,k,q,\beta}$ *problem with* $\beta = \max\{k2^{d-1}, 2(B - L_E)\} + 2hR$ *in time* $t_{SIS} \approx 2t_\Sigma$ *with* $\epsilon_{SIS} \geq \frac{1}{2}(\epsilon_\Sigma - \frac{1}{2^\kappa})\left(\frac{(\epsilon_\Sigma - \frac{1}{2^\kappa})}{q_h} - \frac{1}{2^\kappa}\right) + \epsilon_\Sigma/2$ *with our choice of parameters.*

The second security reduction (cf. Theorem 6) gives a tight reduction in the *quantum* random oracle model from R-LWE to EUF-CMA of `qTESLA`. In our opinion, this second theorem is much stronger since it guarantees security against adversaries that have quantum access to a quantum random oracle. Accordingly, we always refer to Theorem 6 when discussing the security of the scheme. We emphasize that Theorem 6 gives a reduction from the decisional R-LWE problem only, while in Theorem 5 the decisional R-SIS problem is additionally used. Currently, Theorem 6 holds assuming a conjecture, as explained below.

Theorem 6. *Let the parameters be as in Table 2. Furthermore, assume that Conjecture 7 holds. If there exists an adversary A that forges a signature of the signature scheme* `qTESLA` *described in Section 2.2 in time t_Σ and with success probability ϵ_Σ, then there exists a reduction R that solves the $R{-}LWE_{n,k,q,\sigma}$ problem in time $t_{LWE} \approx t_\Sigma$ with $\epsilon_\Sigma \le \frac{2^{3\lambda+nkd} \cdot 4 \cdot q_s^3(q_s+q_h)^2}{q^{nk}} + \frac{2q_h+5}{2^\lambda} + \epsilon_{LWE}$ with parameters as in Table 2.*

The proof follows the approach proposed in [9] except for the computation of the two probabilities $\mathrm{coll}(a,e)$ and $\mathrm{nwr}(a,e)$ that we explain in the following. For simplicity we assume that the randomness is sampled uniformly random in $\mathcal{R}_{q,[B]}$ as in Algorithm 2. We define $\Delta\mathbb{L}$ to be the set $\{x - x' : x, x' \in \mathcal{R} \text{ and } [x]_M = [x']_M\}$. Furthermore, we call a polynomial w *well-rounded* if w is in $\mathcal{R}_{q,[\lfloor q/2 \rfloor - L_E]}$ and $[w]_L \in \mathcal{R}_{q,[(2^{d-1}-L_E)]}$. We define the following quantities for keys $(a_1, ..., a_k, t_1, ..., t_k)$, $(s, e_1, ..., e_k)$, where we denote $\overrightarrow{a} = (a_1, ..., a_k)$ and $\overrightarrow{e} = (e_1, ..., e_k)$:

$$\mathrm{nwr}(\overrightarrow{a}, \overrightarrow{e}) \overset{\mathrm{def}}{=} \Pr_{(y,c)\in\mathbb{Y}\times\mathbb{H}} [a_i y - e_i c \text{ not well-rounded for at least one } i \in \{1, ..., k\}] \tag{7}$$

$$\mathrm{coll}(\overrightarrow{a}, \overrightarrow{e}) \overset{\mathrm{def}}{=} \max_{(w_1,...,w_k)\in\mathbb{W}^k} \left\{ \Pr_{(y,c)\in\mathbb{Y}\times\mathbb{H}} [[a_1 y - e_1 c]_M = w_1, ..., [a_k y - e_k c]_M = w_k] \right\}. \tag{8}$$

Informally speaking $\mathrm{nwr}(\overrightarrow{a}, \overrightarrow{e})$ refers to the probability over random (y, c) that $a_i y - e_i c$ is not well-rounded for some i. This quantity varies as a function of $a_1, ..., a_k, e_1, ..., e_k$. In contrast to [9], we cannot upper bound this in general in the ring setting. Hence, we first assume that $\mathrm{nwr}(\overrightarrow{a}, \overrightarrow{e}) < 3/4$ and afterwards check experimentally that this holds true. As our acceptance probability of w_i in line 18 of Algorithm 7 (signature generation) is at least 1/4 for all parameter sets (cf. δ_w in Table 2), the bound $\mathrm{nwr}(\overrightarrow{a}, \overrightarrow{e}) < 3/4$ holds.

Secondly, we need to bound the probability $\mathrm{coll}(\overrightarrow{a}, \overrightarrow{e})$. In [9, Lemma 4] the corresponding probability $\mathrm{coll}(A, E)$ for standard lattices is upper bounded. Unfortunately, we were not able to transfer the proof to the ring setting for the following reason. In the proof of [9, Lemma 4], it is used that if the randomness y is not equal to 0 the vector Ay is uniformly random distributed over $\mathbb{Z}_q$ and hence also $Ay - Ec$ is uniformly random distributed over $\mathbb{Z}_q$. This does not necessarily hold if the *polynomial* y is chosen uniformly in $\mathcal{R}_{q,[B]}$. Moreover, in Equation (99) in [9], ψ denotes the probability that a random vector

$x \in \mathbb{Z}_q^m$ is in $\Delta\mathbb{L}$:

$$\psi \overset{\text{def}}{=} \Pr_{x \in \mathbb{Z}_q^m} [x \in \Delta\mathbb{L}] \leq \left(\frac{2^d+1}{q}\right)^m . \tag{9}$$

The quantity ψ is a function of the TESLA parameters q, m, d, and it is negligibly small.

We cannot prove a similar statement for the signature scheme `qTESLA` over ideals. Instead, we need to *conjecture* the following.

Conjecture 7. *Let I be a nonzero ideal in $\mathcal{R}_q$ and let $r \in \mathcal{R}_q$ be a fixed choice of ring elements. Then, it holds that the probability that $x + r \in \Delta\mathbb{L}$ for a uniformly distributed element $x \leftarrow_{\$} I$ is negligibly small.*

The intuition behind our conjecture is as follows. Let ψ_I denote the probability that a random element from the ideal I lands in $\Delta\mathbb{L}$. We know that ψ_I is small when the ideal $I = \mathcal{R}_q$, i.e., a negligibly small fraction of elements from $\mathcal{R}_q$ are in $\Delta\mathbb{L}$. Furthermore, the set $\Delta\mathbb{L}$ appears to have no relationship with the ideal structure of the ring, so it seems reasonable to view each ideal as a "random" subset of $\mathcal{R}_q$ in the following sense: no larger or smaller portion of elements in the ideal I is in $\Delta\mathbb{L}$ than that portion of elements of $\mathcal{R}_q$ that is in $\Delta\mathbb{L}$.

Hence, the corresponding statement described above and needed in [9, Lemma 4] translates for `qTESLA` to the following. If $y \neq 0$ then $a_i y$ is a uniformly random element of some nonzero ideal I for all i. The polynomial c is fixed and the polynomials $e_1, ..., e_k$ are independent of the polynomials $a_1, ..., a_k$, and y. Hence, by our conjecture (with $x = a_i y$ and $r = e_i c$) it holds that the probability of Equation (107) in [9] is negligibly small. Thus, assuming that our conjecture holds true, [9, Lemma 4] and hence the security reduction in [9] holds for `qTESLA` as well.

5.2 Bit security of our proposed parameter sets

In the following, we describe how we estimate the concrete security of the proposed parameters described in Section 2.5. To this end, we first describe how the security of our scheme depends on the hardness of R-LWE and afterwards we describe how we derive the bit hardness of the underlying R-LWE instance.

5.2.1 Correspondence between security and hardness

The security reduction given by Theorem 6, in Section 5.1, provides a reduction from the hardness of the decisional ring learning with errors problem and bounds *explicitly* the forging probability with the success probability of the reduction. More formally, let ϵ_Σ and

t_Σ denote the success probability and the runtime (resp.) of a forger against our signature scheme, and let ϵ_{LWE} and t_{LWE} denote analogous quantities for the reduction presented in the proof of Theorem 6. We say that R-LWE is η-*bit hard* if $t_{LWE}/\epsilon_{LWE} \geq 2^\eta$; and we say that the signature scheme is λ-*bit secure* if $t_\Sigma/\epsilon_\Sigma \geq 2^\lambda$.

For our *provably-secure* parameter sets qTESLA-p-I and qTESLA-p-III, we choose parameters such that $\epsilon_{LWE} \approx \epsilon_\Sigma$ and $t_\Sigma \approx t_{LWE}$, that is, the bit hardness of the R-LWE instance is *theoretically* the same as the bit security of our signature scheme. Hence, the security reduction provably guarantees that our scheme instantiated with the two provably-secure parameter sets has the selected security level as long as the corresponding R-LWE instance is intractable.

For our *heuristic* parameter sets qTESLA-I, qTESLA-III-speed, and qTESLA-III-size, we *assume* that the bit security of our scheme instantiated with these three parameter sets is *theoretically* the same as the bit hardness of the corresponding R-LWE instance. So far no attack that exploits this heuristic is known. However, this is a somewhat less trustworthy approach that trades provable security assurance for performance.

Remark 8. *In practical instantiations of* qTESLA*, the bit security does not exactly match the bit hardness of R-LWE (see Table 3). This is because the bit security does not only depend on the bit hardness of R-LWE (as explained above), but also on the probability of rejected/accepted key pairs and on the security of other building blocks such as the encoding function* Enc*. First, in all our parameter sets, heuristic and provably-secure, the key space is reduced by the rejection of polynomials $s, e_1, ..., e_k$ with large coefficients via* checkE *and* checkS*. We compensate this security loss by choosing an R-LWE instance of larger bit hardness. Moreover, we instantiate the encoding function* Enc *such that it is λ-bit secure; cf. Section 2.5.*

5.2.2 Estimation of the R-LWE hardness

Since the introduction of the learning with errors problem over rings [49], it has remained an open question to determine whether the R-LWE problem is as hard as the LWE problem. Several results exist that exploit the structure of some ideal lattices [22,25,33,35]. However, up to now, these results do not seem to apply to R-LWE instances that are typically used in signature schemes and, therefore, do not apply to the proposed qTESLA instances. Consequently, we assume that the R-LWE problem is as hard as the LWE problem, and estimate the hardness of R-LWE using state-of-the-art attacks against LWE.

Albrecht, Player, and Scott [8] presented the *LWE-Estimator*, a software to estimate the hardness of LWE given the matrix dimension n, the modulus q, the relative error rate $\alpha = \frac{\sqrt{2\pi}\sigma}{q}$, and the number of given LWE samples. The LWE-Estimator estimates the hardness against the fastest LWE solvers currently known, i.e., it outputs an upper (conservative) bound on the number of operations an attack needs to break a given LWE instance. In particular, the following attacks are considered in the LWE-Estimator: the meet-in-the-middle exhaustive search, the coded Blum-Kalai-Wassermann algorithm [40], the dual lattice attacks recently published in [3], the enumeration approach by Linder and Peikert [46], the primal attack described in [6, 15], the Arora-Ge algorithm [11] using Gröbner bases [4], and the latest analysis to compute the block sizes used in the lattice basis reduction BKZ recently published by Albrecht *et al.* [2]. Moreover, quantum speed-ups for the sieving algorithm used in BKZ [44, 45] are also considered.

We note that another recent quantum attack, called quantum hybrid attack, by Göpfert, van Vredendaal, and Wunderer [38] is not considered in our analysis and the LWE-Estimator. This hybrid attack is most efficient on the learning with errors problem with very small secret and error, e.g., binary or ternary. Since the coefficients of the secret and error of `qTESLA` are chosen Gaussian distributed, the attack is not efficient for our instances.

The LWE-Estimator is the result of many different contributions and contributors. It is open source and hence easily checked and maintained by the community. Hence, we find the LWE-Estimator to be a suitable tool to estimate the hardness of our chosen LWE instances. We integrated the LWE-Estimator with commit-id `9302d42` on 2017-09-27 in our sage script.

In the following we describe very briefly the most efficient LWE solvers for our instances, i.e., the decoding attack and the embedding approach, following closely the description of [18]. The Blum-Kalai-Wasserman algorithm [5, 43] is omitted since it requires exponentially many samples.

The embedding attack. The standard embedding attack solves LWE via reduction to the unique shortest vector problem (uSVP). During the reduction an $(m+1)$-dimensional lattice that contains the error vector e is created. Since e is very short for typical LWE instances, this results in a uSVP instance that is usually solved by applying basis reduction.

Let $(A, c = As + e \mod q)$ and t be the distance $\mathrm{dist}(c, L(A)) = \|c - x\|$ where $x \in L(A)$, such that $\|c - x\|$ is minimized. Then the lattice $L(A)$ can be embedded in the lattice $L(A')$, with $A' = \begin{pmatrix} A & c \\ 0 & t \end{pmatrix}$. If $t < \frac{\lambda_1(L(A))}{2\gamma}$, the higher-dimensional lattice $L(A')$ has a unique shortest vector $c' = (-e, t) \in Z_q^{m+1}$ with length $\|c'\| = \sqrt{m\alpha^2 q^2/(2\pi) + |t|^2}$ [26, 48]. In the LWE-Estimator $t = 1$ is used. Therefore, e can be extracted from c', the value As is known, and s can be solved for. Based on Albrecht *et al.* [7], Göpfert shows [37, Section 3.1.3] that the standard embedding attack succeeds with non-negligible probability if $\delta_0 \leq$

$\left(\frac{q^{1-\frac{n}{m}}\sqrt{\frac{1}{e}}}{\tau\alpha q}\right)^{\frac{1}{m}}$, where m is the number of LWE samples. The value τ is experimentally determined to be $\tau \leq 0.4$ for a success probability of $\epsilon = 0.1$ [7].

The efficiency of the embedding attack highly depends on the number of samples. In the case of LWE instances with limited number of samples, the lattice $\Lambda_q^{\perp}(A_o) = \{v \in \mathbb{Z}^{m+n+1} | A_o \cdot v = 0 \bmod q\}$ with $A_o =[A|I|b]$ can be used as the embedding lattice.

The decoding attack. The decoding attack treats an LWE instance as an instance of the bounded distance decoding problem (BDD). The attack can be divided into two phases: basis reduction and finding the closest vector to the target vector. In the first phase, basis reduction algorithms like BKZ [53] are applied. Afterwards, in the second phase, the nearest plane algorithm [13] (or its variants) is applied to find the closest vector to As and thereby eliminate the error vector e of the LWE instance. The secret can then be accessed, as the closest vector equals the value As of an LWE instance.

5.3 Resistance to implementation attacks

Besides the theoretical security against computational attacks, such as lattice reduction, it is important for a cryptographic scheme to be secure against implementation attacks. These attacks come in two flavors: side-channel and fault analysis attacks.

5.3.1 Side-channel analysis

These attacks exploit physical information such as timing or power consumption, electromagnetic emanation, etc., that is correlated to some secret information during the execution of a cryptographic scheme. Simple and differential side-channel attacks that rely on power and electromagnetic emanations are very powerful but typically require physical access (or close proximity) to the targeted device. Protecting lattice-based schemes against this class of attacks is a very active area of research.

In contrast, attacks that exploit timing leakage, such as timing and cache attacks, are easier to carry out remotely. Hence, these attacks represent a more immediate danger for most applications and, consequently, it has become a minimum security requirement for a cryptographic implementation to be secure against this class of attacks. One effective approach to provide such a protection is by guarantying so-called *constant-time* execution. In practice, this means that an implementation should avoid the use of secret address accesses and conditional branches based on secret information.

One of the main advantages of qTESLA is that the Gaussian sampler, arguably the most complex part of the scheme, is restricted to key generation. This reduces drastically the attack surface to carry out a timing and cache attack against qTESLA. Still, we remark that qTESLA's Gaussian sampler is relatively simple and can be implemented securely in a constant-time manner, as can be observed in the accompanying implementations. Other functions of qTESLA, such as polynomial arithmetic operations, are easy to implement in constant-time.

We note that, recently, the scheme ring-TESLA [1] was analyzed with respect to cache side channels with the software tool CacheAudit [20]. It was the first time that a post-quantum scheme was analyzed with program analysis. The authors found potential cache side channels, proposed countermeasures, and showed the effectiveness of their mitigations with CacheAudit. In our implementations, we apply similar techniques to those proposed in [20] with some additional optimizations. For example, our implementation of the rejection sampling in the signing algorithm (line 12 of Algorithm 7) protects the sign of each coefficient of z by doing a masked conversion to obtain its absolute value before checking against a unique positive bound. The function leaks, however, the position of the coefficient that fails the test, but this information is independent of the secret data.

Notably, qTESLA has implicitly an additional line of defense, thanks to its non-deterministic nature. To generate the polynomial y, qTESLA hashes together fresh randomness, a value seed_y passed through the secret key and the message m. The use of seed_y makes qTESLA resilient to a catastrophic failure of the RNG during generation of the fresh randomness (e.g., a similar failure of ECDSA signatures was demonstrated in [23]). The randomness guarantees the use of a fresh y at each signing operation, which makes a timing attack (or even more generally, any side-channel attack) more difficult to carry out. More importantly, this implicitly protects against some powerful and easy-to-carry out fault attacks, as explained next.

5.3.2 Fault analysis

Recently, some studies have exposed the vulnerability of lattice-based schemes to fault attacks. We describe a simple yet powerful attack that falls in this category of attacks [21].

Assume that line 3 of Algorithm 7 is computed without the random value r, i.e., as $\mathsf{rand} \leftarrow \mathsf{PRF}_2(\mathsf{seed}_y, m)$. Assume that a signature (z, c) is generated for a given message m. Afterwards, a signature is requested again for the same message m, but this time, a fault is injected on the computation of the hash value c yielding the value c_{faulted}. This second signature is $(z_{\text{faulted}}, c_{\text{faulted}})$. Computing $z - z_{\text{faulted}} = sc - sc_{\text{faulted}} = s(c - c_{\text{faulted}})$, reveals the secret s since $c - c_{\text{faulted}}$ is known to the attacker. As stated in [50], this attack has broad implications since it is generically applicable to deterministic *Schnorr-like* signatures.

It is easy to see that, to prevent this (and other similar) fault attacks, every signing operation should be injected with fresh randomness, as precisely specified in line 3 of Algorithm 7. This makes qTESLA implicitly resilient to this line of attacks.

Remark 9. *In an earlier description of* qTESLA*, the scheme was specified as a deterministic signature scheme and, hence, was susceptible to the fault attacks described in [21].*

6 Advantages and limitations

In this section, we summarize some advantages and limitations of the proposed signature scheme qTESLA. In addition, we compare our scheme with other post-quantum and classical signatures.

Security of our signature scheme. qTESLA comes accompanied by a *tight* security proof (cf. Theorem 6) in the quantum random oracle model, i.e., a quantum adversary is allowed to ask the random oracle in superposition. This reduction is based on a variant of our scheme over standard lattices [9]. To port the reduction given in [9], we use a heuristic argument as explained in Section 5.1. The tightness in the proof enables the use of smaller parameters and, thus, better performance when choosing parameters according to security reductions.

Our signature scheme instantiated with *provably-secure* parameters is provably EUF-CMA secure by virtue of the provided security reduction from the hardness of the decisional R-LWE problem to EUF-CMA security. Moreover, since our security reduction is *explicit*, we can explicitly give the relation between the success probabilities of solving the R-LWE problem and forging qTESLA signatures. This provides a high-level of security assurance to the proposed signature scheme.

Flexible choice of parameters. We offer *two* approaches to instantiate qTESLA: using "heuristic" parameters and using "provably-secure" parameters that follow existing security reductions.

The heuristic approach identifies the security level of an instantiation of a scheme by a certain parameter set with the hardness level of the instance of the underlying lattice problem that corresponds to these parameters, regardless of the tightness gap of the provided security reduction. This approach supports implementations that achieve very high-speed execution while requiring relatively compact keys and signatures.

The parameter choice according to a reduction can be considered as providing a *stronger* security argument since it provably guarantees that the scheme has the selected security level as long as the corresponding R-LWE instance is intractable. This approach supports implementations that are slower and require larger public keys, but that can still be considered practical for many high-security applications.

In summary, the choice between both options represent a trade-off between provably security assurance (provided by the provably-secure parameter sets) and greater efficiency (provided by the heuristic parameter sets).

The bit-security of the proposed parameters was estimated against known state-of-the-art classical and quantum algorithms that solve the learning with errors problem. The proposed parameters, especially in the case of `provably-secure qTESLA`, provide a comfortable margin between the targeted and the estimated bit security in order to deal with future or unknown LWE solvers as well.

Finally, the parameter generation of `qTESLA` is easy to audit: the choice of parameters and their relation are clearly explained; and the generation procedure is simple and easy to follow. In order to facilitate the generation of new parameters (if needed), and also for transparency, we make our sage script for parameter generation available[3].

Ease of implementation and scalability. `qTESLA` has a very compact structure consisting of a few, easy-to-implement functions. The Gaussian sampler, arguably the most complex function in `qTESLA`, is only required during key generation. Therefore, even if the fast Gaussian sampler included in this document is not used, most applications will not be impacted by the use of a slower Gaussian sampler.

`qTESLA` exhibits great scalability to support different security levels with a unique, efficient implementation. The simplicity of `qTESLA` makes it easy to support any of the proposed security levels with a common codebase. For example, our implementations for `qTESLA-I`, `qTESLA-III-speed` and `qTESLA-III-size` share about 325 lines of C code, differing in only about 100 lines of C code corresponding to easy-to-implement packing functions, and a few NTT constants and system parameters which are instantiated at compilation time. Furthermore, the packing functions can be optionally implemented in a more generic way, which would virtually eliminate any differences in the C code (beside the NTT constants and system parameters). This also highlights the compactness of realizations of `qTESLA`; for example, our reference implementations of `heuristic qTESLA` require about 430 lines of C code [4], which is significantly shorter than other schemes based on ideal lattices.

[3] The parameter generation script can be found at `\Supporting_Documentation\Script_to_choose_parameters\parameterchoice.sage`.

[4] This count excludes header files, the NTT constants and the code for the additional symmetric primitives.

High-speed and key compactness. Despite the compactness of the scheme, qTESLA (especially with "heuristic" parameters) supports high-speed implementations with relatively compact keys and signatures. The superb performance of heuristic qTESLA is specifically achieved in the computation of signing and verification, and is obtained at the expense of a relatively more expensive key generation. This approach suits perfectly most scenarios in which key generation is executed less frequently or is done offline.

qTESLA signatures are also relatively compact, making them ideal for applications in which signature size is prioritized over public key size. We note that the combined size of signature and public key for heuristic qTESLA is still competitive with other efficient schemes based on ideal lattices.

As previously stated, provably-secure qTESLA implementations are slower and require larger public keys, but they are still practical for many applications and come with provably secure guarantees that other signature schemes do not provide.

Security against implementation attacks. qTESLA's simplicity and compactness facilitates its implementation in constant-time and, arguably, its protection against more powerful attacks such as differential power analysis. As stated before, Gaussian sampling is only required during key generation, which reduces significantly the attack surface over this function. Moreover, qTESLA requires a *simpler* Gaussian sampler which further eases implementation and protection against side-channel attacks.

qTESLA comes equipped with built-in measures that protect against some side-channel and fault attacks. Since qTESLA generates a fresh y per signing operation, some simple side-channel attacks are more difficult to mount against the scheme. More importantly, this feature immediately renders some powerful and easy-to-carry out fault attacks unfeasible. At the same time, qTESLA is resilient to a catastrophic failure of the RNG during generation of the fresh randomness that is required to generate y.

Comparison with selected state-of-the-art signature schemes. In this subsection we compare qTESLA with other post-quantum and classical signature schemes.

Table 6 summarizes the most relevant features of selected signature schemes, including the underlying computational assumption, the bit security against classical and quantum attacks, and key and signature sizes.

As can be seen, qTESLA is among the most compact post-quantum schemes with regard to signature size. In particular, the signature size of qTESLA is several magnitudes smaller than hash-based and multivariate alternatives. Among post-quantum lattice based schemes, heuristic qTESLA offers the smallest signatures, and represents a very competitive option when the combined size of public keys and signatures is taken into account. The lattice-based signature scheme BLISS has noticeably smaller signatures, but the bit security is not estimated against quantum adversaries. Likewise, if one takes into account the provided security reduction, `provably-secure qTESLA` offers very competitive key and signature sizes, and is quantum-resistant.

We note, however, that classical schemes such as RSA and ECDSA feature significantly smaller signature and public key sizes than all the available post-quantum alternatives.

Table 6: Overview of selected post-quantum and classical signature schemes; signature and key sizes are given in bytes; we write "–" if no corresponding data is available

	Software/ Scheme	Comp. Assum.	Bit Security	Key Size (bytes)	Sig. Size (bytes)
Selected lattice-based signatures	`qTESLA-I` (this document)	R-LWE	95[b]	pk: 1 504 sk: 1 216	1 376
	`qTESLA-III-speed` (this document)	R-LWE	160[b]	pk: 3 104 sk: 2 112	2 848
	`qTESLA-III-size` (this document)	R-LWE	160[b]	pk: 2 976 sk: 2 112	2 720
	`qTESLA-p-I` (this document)	R-LWE[a]	95[b]	pk: 14 880 sk: 4 576	2 848
	`qTESLA-p-III` (this document)	R-LWE[a]	160[b]	pk: 39 712 sk: 12 320	6 176
	Dilithium-II (medium) [28]	module SIS module LWE	91[b]	pk: 1 184 sk: –	2 044
	Dilithium-II (recommended) [28]	module SIS module LWE	125[b]	pk: 1 472 sk: –	2 700
	Dilithium-III (very high) [28]	module SIS module LWE	158[b]	pk: 1 760 sk: –	3 366
	GPV-poly [32, 36]	R-SIS[a]	96[c]	pk: 55 705 sk: 26 316	32 972
	BLISS-B-IV [29, 57]	R-SIS, NTRU	182[c]	pk: 896 sk: 384	812
Other selected post-quantum signatures	gravity-SPHINCS [12]	Hash collisions, 2nd preimage	128[b]	pk: 32 sk: 64	22 304
	SPHINCS-256 [17]	Hash collisions, 2nd preimage	128[b]	pk: 1 056 sk: 1 088	41 000
	MQDSS-31-64 [24]	Multivariate Quadratic system	128[b]	pk: 72 sk: 64	40 952
Selected classical signatures	RSA-3072 [55]	Integer Factorization	128[d]	pk: 384 sk: 1 728	384
	ECDSA (P-256) [41]	Elliptic Curve Discrete Logarithm	128[d]	pk: 64 sk: 96	64

[a] Parameters are chosen according to given security reduction in the quantum random oracle model.

[b] Bit security analyzed against classical and quantum adversaries.

[c] Bit security analyzed against classical adversaries.

[d] Broken by quantum computers (bit security analyzed against classical adversaries).

References

[1] Sedat Akleylek, Nina Bindel, Johannes A. Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. An efficient lattice-based signature scheme with provably secure instantiation. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2016 - 8th International Conference on Cryptology in Africa*, volume 9646 of *LNCS*, pages 44–60. Springer, 2016.

[2] Martin Albrecht, Florian Göpfert, Fernando Vidria, and Thomas Wunderer. Revisiting the Expected Cost of Solving uSVP and Applications to LWE. In *ASIACRYPT 2017 - Advances in Cryptology, to appear*. Springer, 2017.

[3] Martin R. Albrecht. On Dual Lattice Attacks Against Small-Secret LWE and Parameter Choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *LNCS*, pages 103–129, 2017.

[4] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Algebraic algorithms for LWE problems. *ACM Comm. Computer Algebra*, 49(2):62, 2015.

[5] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Designs, Codes and Cryptography*, 74(2):325–354, 2015.

[6] Martin R. Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving LWE by reduction to unique-svp. In Hyang-Sook Lee and Dong-Guk Han, editors, *Information Security and Cryptology - ICISC 2013*, volume 8565 of *LNCS*, pages 293–310. Springer, 2013.

[7] Martin R. Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In Hyang-Sook Lee and Dong-Guk Han, editors, *ICISC 13: 16th International Conference on Information Security and Cryptology*, volume 8565 of *Lecture Notes in Computer Science*, pages 293–310, Seoul, Korea, November 27–29, 2014. Springer, Heidelberg, Germany.

[8] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[9] Erdem Alkim, Nina Bindel, Johannes A. Buchmann, Özgür Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krämer, and Filip Pawlega. Revisiting TESLA in the quantum random oracle model. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum*

Cryptography - 8th International Workshop, PQCrypto 2017, volume 10346 of *LNCS*, pages 143–162. Springer, 2017.

[10] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16*, pages 327–343. USENIX Association, 2016.

[11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *Automata, Languages and Programming*, volume 6755 of *LNCS*, pages 403–415. Springer, 2011.

[12] Jean-Philippe Aumasson and Guillaume Endignoux. Improving stateless hash-based signatures. Cryptology ePrint Archive, Report 2017/933, 2017. https://eprint.iacr.org/2017/933.

[13] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. In K. Mehlhorn, editor, *STACS 1985*. Springer, 1985.

[14] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *Lecture Notes in Computer Science*, pages 28–47, San Francisco, CA, USA, February 25–28, 2014. Springer, Heidelberg, Germany.

[15] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *ACISP 14: 19th Australasian Conference on Information Security and Privacy*, volume 8544 of *Lecture Notes in Computer Science*, pages 322–337, Wollongong, NSW, Australia, July 7–9, 2014. Springer, Heidelberg, Germany.

[16] Paulo S. L. M. Barreto, Patrick Longa, Michael Naehrig, Jefferson E. Ricardini, and Gustavo Zanon. Sharper ring-LWE signatures. Cryptology ePrint Archive, Report 2016/1026, 2016. http://eprint.iacr.org/2016/1026.

[17] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 9056 of *LNCS*, pages 368–397. Springer, 2015.

[18] Nina Bindel, Johannes Buchmann, Florian Göpfert, and Markus Schmidt. Estimation of the hardness of the learning with errors problem with a restricted number of samples. Cryptology ePrint Archive, Report 2017/140, 2017. https://eprint.iacr.org/2017/140.

[19] Nina Bindel, Johannes Buchmann, and Juliane Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. In *2016 Workshop on Fault Diagnosis*

and Tolerance in Cryptography, FDTC 2016, pages 63–77. IEEE Computer Society, 2016.

[20] Nina Bindel, Johannes Buchmann, Juliane Krämer, Heiko Mantel, Johannes Schickel, and Alexandra Weber. Bounding the cache-side-channel leakage of lattice-based signature schemes using program semantics. In *Proceedings of the 10th International Symposium on Foundations & Practice of Security (FPS)*, 2017. To appear.

[21] Leon Groot Bruinderink and Peter Pessl. Differential fault attacks on deterministic lattice signatures. Cryptology ePrint Archive, Report 2018/355, 2018. https://eprint.iacr.org/2018/355.

[22] Peter Campbell, Michael Groves, and Dan Shepherd. SOLILOQUY: A cautionary tale. ETSI 2nd Quantum-Safe Crypto Workshop, 2014. http://docbox.etsi.org/Workshop/2014/201410_CRYPTO/S07_Systems_and_Attacks/S07_Groves_Annex.pdf.

[23] H.M. Cantero, S. Peter, Bushing, and Segher. Console hacking 2010 – PS3 epic fail. 27th Chaos Communication Congress, 2010. https://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf.

[24] Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. From 5-pass MQ-based identification to MQ-based signatures. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security*, volume 10032 of *LNCS*, pages 135–165, 2016.

[25] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 9666 of *LNCS*, pages 559–585. Springer, 2016.

[26] Özgür Dagdelen, Rachid El Bansarkhani, Florian Göpfert, Tim Güneysu, Tobias Oder, Thomas Pöppelmann, Ana Helena Sánchez, and Peter Schwabe. High-speed signatures from standard lattices. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology – LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 84–103. Springer, 2015.

[27] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference*, volume 8042 of *LNCS*, pages 40–56. Springer, 2013.

[28] Leo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS – Dilithium: Digital Signatures from Module Lattices. Cryptology ePrint Archive, Report 2017/633, 2017. http://eprint.iacr.org/2017/633.

[29] Léo Ducas. Accelerating bliss: the geometry of ternary polynomials. Cryptology ePrint Archive, Report 2014/874, 2014. http://eprint.iacr.org/2014/874/.

[30] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. Cryptology ePrint Archive, Report 2013/383, 2013. https://eprint.iacr.org/2013/383.

[31] Morris J. Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. *Federal Inf. Process. Stds. (NIST FIPS) – 202*, 2015. Available at https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[32] Rachid El Bansarkhani and Jan Sturm. An efficient lattice-based multisignature scheme with applications to bitcoins. In Sara Foresti and Giuseppe Persiano, editors, *CANS 2016*, pages 140–155, Cham, 2016. Springer International Publishing.

[33] Yara Elias, Kristin E. Lauter, Ekin Ozman, and Katherine E. Stange. Provably weak instances of ring-lwe. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference*, volume 9215 of *LNCS*, pages 63–92. Springer, 2015.

[34] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Loop-abort faults on lattice-based fiat-shamir and hash-and-sign signatures. In Roberto Avanzi and Howard M. Heys, editors, *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference*, volume 10532 of *Lecture Notes in Computer Science*, pages 140–158. Springer, 2017.

[35] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[36] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing (STOC 2008)*, pages 197–206. ACM, 2008.

[37] Florian Göpfert. *Securely Instantiating Cryptographic Schemes Based on the Learning with Errors Assumption.* PhD thesis, Darmstadt University of Technology, Germany, 2016.

[38] Florian Göpfert, Christine van Vredendaal, and Thomas Wunderer. A hybrid lattice basis reduction and quantum search attack on LWE. In Tanja Lange and Tsuyoshi

Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, volume 10346 of *LNCS*, pages 184–202. Springer, 2017.

[39] Shay Gueron and Fabian Schlieker. Optimized implementation of ring-TESLA. GitHub at https://github.com/fschlieker/ring-TESLA, 2016.

[40] Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-bkw: Solving LWE using lattice codes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9215 of *LNCS*, pages 23–42. Springer, 2015.

[41] James Howe, Thomas Pöppelmann, Máire O'neill, Elizabeth O'sullivan, and Tim Güneysu. Practical lattice-based digital signature schemes. *ACM Trans. Embed. Comput. Syst.*, 14, 2015.

[42] John Kelsey. SHA-3 derived functions: cSHAKE, KMAC, TupleHash, and ParallelHash. *NIST Special Publication*, 800:185, 2016. Available at http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf.

[43] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for lwe with applications to cryptography and lattices. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9215 of *LNCS*, pages 43–62. Springer, 2015.

[44] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2016.

[45] Thijs Laarhoven, Michele Mosca, and Joop Pol. Solving the Shortest Vector Problem in Lattices Faster Using Quantum Search. In Philippe Gaborit, editor, *Post-Quantum Cryptography*, volume 7932 of *Lecture Notes in Computer Science*, pages 83–101. Springer Berlin Heidelberg, 2013.

[46] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, 2011.

[47] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[48] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference*, pages 577–594. Springer, 2009.

[49] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

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

[51] T. Pöppelmann and T. Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 2014.

[52] S. Sinha Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. Compact Ring-LWE cryptoprocessor. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 371–391. Springer, 2014.

[53] Claus P. Schnorr and Taras Shevchenko. Solving subset sum problems of densioty close to 1 by randomized BKZ-reduction. Cryptology ePrint Archive, Report 2012/620, 2012. http://eprint.iacr.org/2012/620.

[54] Gregor Seiler. Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. Cryptology ePrint Archive, Report 2018/039, 2018. https://eprint.iacr.org/2018/039.

[55] Mikael Sjöberg. *Post-quantum algorithms for digital signing in Public Key Infrastructures*. PhD thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2017.

[56] The National Institute of Standards and Technology (NIST). Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, December, 2016. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[57] Thomas Wunderer. Revisiting the hybrid attack: Improved analysis and refined security estimates. Cryptology ePrint Archive, Report 2016/733, 2016. https://eprint.iacr.org/2016/733.