The Lattice-Based Digital Signature Scheme qTESLA

Erdem Alkim[1,2], Paulo S. L. M. Barreto[3], Nina Bindel[4(✉)], Juliane Krämer[5], Patrick Longa[6], and Jefferson E. Ricardini[7]

[1] Ondokuz Mayis University, Atakum, Turkey
[2] Fraunhofer SIT, Darmstadt, Germany
erdemalkim@gmail.com
[3] University of Washington Tacoma, Tacoma, USA
pbarreto@uw.edu
[4] University of Waterloo, Waterloo, Canada
nlbindel@uwaterloo.ca
[5] Technische Universität Darmstadt, Darmstadt, Germany
jkraemer@cdc.informatik.tu-darmstadt.de
[6] Microsoft Research, Redmond, USA
plonga@microsoft.com
[7] LG Electronics, Englewood Cliffs, USA
jefferson1.ricardini@lge.com

Abstract. We present qTESLA, a post-quantum provably-secure digital signature scheme that exhibits several attractive features such as simplicity, strong security guarantees against quantum adversaries, and built-in protection against certain side-channel and fault attacks. qTESLA—selected for round 2 of NIST's post-quantum cryptography standardization project—consolidates a series of recent schemes originating in works by Lyubashevsky, and Bai and Galbraith. We provide full-fledged, constant-time portable C implementations consisting of only about 300 lines of C code, which showcases the code compactness of the scheme. Our results also demonstrate that a conservative, provably-secure signature scheme can be efficient and practical, even with a compact and portable implementation. For instance, our C-only implementation executes signing and verification in approximately 0.9 ms on an x64 Intel processor using the proposed level 1 parameter set. Finally, we also provide AVX2-optimized assembly implementations that achieve an additional factor-1.5 speedup.

The work of EA was partially supported by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE, and was partially carried out during his tenure of the ERCIM 'Alain Bensoussan' Fellowship Programme. NB is supported by the NSERC Discovery Accelerator Supplement grant RGPIN-2016-05146. JK is co-funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 – 236615297. JR is partially supported by the joint São Paulo Research Foundation (FAPESP)/Intel Research grant 2015/50520-6 "Efficient Post-Quantum Cryptography for Building Advanced Security Applications".

© Springer Nature Switzerland AG 2020
M. Conti et al. (Eds.): ACNS 2020, LNCS 12146, pp. 441–460, 2020.
https://doi.org/10.1007/978-3-030-57808-4_22

Keywords: Post-quantum cryptography · Lattice-based cryptography · Digital signatures · Provable security · Efficient implementation

1 Introduction

In this work, we introduce a lattice-based digital signature scheme called qTESLA which consolidates a series of recent efforts to design an efficient and provably-(quantum-) secure signature scheme. The security of qTESLA relies on the so-called decisional Ring Learning With Errors (R-LWE) problem [35]. Parameters are generated according to the provided security reduction from R-LWE, i.e., instantiations of the scheme guarantee a certain security level as long as the corresponding R-LWE instances give a certain hardness[1].

The most relevant features of qTESLA are summarized as follows:

Simplicity. qTESLA is designed to be easy to implement with special emphasis on the most used functions in a signature scheme, namely, signing and verification. In particular, Gaussian sampling, arguably the most complex part of traditional lattice-based signature schemes, is relegated exclusively to key generation. qTESLA's simple design makes it straightforward to easily support more than one security level or parameter set with a single and compact portable implementation. For instance, our reference implementation written in portable C and supporting both qTESLA instantiations consists of only ∼300 lines of code[2].

Security Foundation. The security of qTESLA is ensured by a security reduction in the Quantum Random Oracle Model (QROM) [12], i.e., a quantum adversary is allowed to ask the random oracle in superposition. Moreover, the explicitness of the reduction enables choosing parameters according to the reduction, while its tightness enables smaller parameters and, thus, better performance for provably secure instantiations.

Practical Security. qTESLA facilitates realizations that are secure against implementation attacks. For example, it supports *constant-time* implementations (i.e., implementations that are secure against timing and cache side-channel attacks by avoiding secret memory accesses and secret branches), and is inherently protected against certain simple yet powerful fault attacks [13,38]. Moreover, it also comes with a built-in safeguard to protect against Key Substitution (KS) attacks [11,36] (a.k.a. duplicate signature key selection attacks) and, thus, provides improved security in the multi-user setting; see also [27].

Related Work. qTESLA is the result of a long line of research and consolidates the most relevant features of the prior works. The first work in this line is the signature scheme proposed by Bai and Galbraith [7], which is based on the Fiat-Shamir construction of Lyubashevsky [32,33]. The Bai-Galbraith scheme is

[1] It is important to note that the security reduction requires a conjecture, see Sect. 4.

[2] This count excludes the parameter-specific packing functions, header files, NTT constants, and (c)SHAKE functions.

constructed over standard lattices and comes with a (non-tight) security reduction from the LWE and the Short Integer Solution (SIS) problem in the Random Oracle Model (ROM). In [18] improvements and the first implementation of the Bai-Galbraith scheme were presented. The scheme was subsequently studied under the name TESLA [5], and an alternative (tight) reduction from the LWE problem in the QROM was provided. A variant of TESLA over ideal lattices was derived under the name ring-TESLA [1]. `qTESLA` is a direct successor of this scheme, with several modifications aimed at improving its security, correctness, and implementation, the most important of which are: `qTESLA` includes a new *correctness requirement* that prevents occasional rejections of valid signatures during ring-TESLA's verification; `qTESLA`'s *security reduction* is proven in the QROM while ring-TESLA's reduction was only given in the ROM; the *security estimations* of ring-TESLA are not state-of-the-art and are limited to classical algorithms while `qTESLA`'s instantiations are with respect to state-of-the-art classical *and* quantum attacks; the *number of R-LWE samples* in `qTESLA` is flexible, not fixed to two samples as in ring-TESLA, which enables instantiations with better efficiency; our `qTESLA` implementations are protected against several implementation attacks while known implementations of ring-TESLA are not (e.g., do not run in constant-time). In addition, `qTESLA` adopts the next features: following a standard security practice, the public polynomials a_i are freshly generated at each key pair generation; and the hash of the public key is included in the signature computation to protect against KS attacks [11].

Another variant of the Bai-Galbraith scheme is the lattice-based signature scheme Dilithium [21,34] which is constructed over module lattices. While `qTESLA` and Dilithium share several properties such as a tight security reduction in the QROM [29], Dilithium's parameters are not strictly chosen according to it. Also, Dilithium signatures are *deterministic* by default[3], whereas `qTESLA` signatures are *probabilistic* and come with built-in protection against some powerful fault attacks such as the simple and easy-to-implement fault attack in [13,38]. We remark that, arguably, side-channel attacks are more difficult to carry out against probabilistic signatures.

Two other schemes played a major role in the history of Fiat-Shamir lattice signature schemes, namely, GLP [25] and BLISS [20]. These schemes were inspirational for some of `qTESLA`'s building blocks, such as the encoding function.

Software Release. We have released our portable implementations as open source at https://github.com/Microsoft/qTESLA-Library. The implementation software submitted to NIST's Post-Quantum Cryptography Standardization process is available at https://github.com/qtesla/qTesla.

Outline. After describing some preliminary details in Sect. 2, we present the signature scheme in Sect. 3. In Sect. 4, we describe the security foundation of `qTESLA` and the proposed parameter sets. Finally, we give implementation details

[3] Recently, a variant of Dilithium that produces probabilistic signatures was included as a modification for round 2 of the NIST post-quantum project [34]. However, [34] suggests the deterministic version as the default option.

of our C and AVX2-optimized implementations, as well as our experimental results and a comparison with state-of-the-art signature schemes in Sect. 5.

2 Preliminaries

2.1 Notation

Rings. Let q be an odd prime throughout this work. Let $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ denote the quotient ring of integers modulo q, and let $\mathcal{R}$ and $\mathcal{R}_q$ denote the rings $\mathbb{Z}[x]/\langle x^n + 1\rangle$ and $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$, respectively.

Given $f = \sum_{i=0}^{n-1} f_i x^i \in \mathcal{R}$, we define the reduction of f modulo q to be $\sum_{i=0}^{n-1}(f_i \bmod q)x^i \in \mathcal{R}_q$. Let $\mathbb{H}_{n,h} = \{\sum_{i=0}^{n-1} f_i x^i \in \mathcal{R} \mid f_i \in \{-1, 0, 1\}, \sum_{i=0}^{n-1} |f_i| = h\}$, and $\mathcal{R}_{[B]} = \{\sum_{i=0}^{n-1} f_i x^i \in \mathcal{R} \mid f_i \in [-B, B]\}$.

Rounding Operators. 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 c' such that $-m/2 < c' \leq m/2$ (resp. $-\lfloor m/2 \rfloor \leq c' \leq \lfloor m/2 \rfloor$) and $c' = c \bmod m$. We then define the functions $[\cdot]_L : \mathbb{Z} \to \mathbb{Z},\ c \mapsto (c \bmod^{\pm} q) \bmod^{\pm} 2^d$, and $[\cdot]_M : \mathbb{Z} \to \mathbb{Z},\ c \mapsto (c \bmod^{\pm} q - [c]_L)/2^d$. Hence, $c \bmod^{\pm} q = 2^d \cdot [c]_M + [c]_L$ for $c \in \mathbb{Z}$. These definitions are extended to polynomials by applying the operators to each polynomial coefficient, i.e., $[f]_L = \sum_{i=0}^{n-1} [f_i]_L \, x^i$ and $[f]_M = \sum_{i=0}^{n-1} [f_i]_M \, x^i$ for a given $f = \sum_{i=0}^{n-1} f_i x^i \in \mathcal{R}$.

Infinity Norm. Given $f \in \mathcal{R}$, the function $\max_k(f)$ returns the k-th largest absolute coefficient of f. For an element $c \in \mathbb{Z}$, we have that $\|c\|_\infty = |c \bmod^{\pm} q|$, and define the infinity norm for a polynomial $f \in \mathcal{R}$ as $\|f\|_\infty = \max_k \|f_k\|_\infty$.

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

We define the Number Theoretic Transform (NTT) and the R-LWE problem in the full version of this paper [4, Section 2].

3 The Signature Scheme qTESLA

qTESLA is parameterized by λ, κ, n, k, q, σ, L_E, L_S, E, S, B, d, h, and b_{GenA}; see Table 1 in Sect. 3.1 for a detailed description of all the system parameters. The following functions are required for the implementation of the scheme:

- The pseudorandom functions $\mathsf{PRF}_1 : \{0,1\}^\kappa \to \{0,1\}^{\kappa,k+3}$ and $\mathsf{PRF}_2 : \{0,1\}^\kappa \times \{0,1\}^\kappa \times \{0,1\}^{320} \to \{0,1\}^\kappa$.
- The collision-resistant hash function $\mathsf{G} : \{0,1\}^* \to \{0,1\}^{320}$.
- The function $\mathsf{GenA} : \{0,1\}^\kappa \to \mathcal{R}_q$ which takes as input the κ-bit seed seed_a and maps it to k polynomials $a_1, ..., a_k \in \mathcal{R}_q$.

- The Gaussian sampler function $\mathsf{GaussSampler}\colon \{0,1\}^\kappa \times \mathbb{Z} \to \mathcal{R}$, which takes as inputs a κ-bit seed $\mathsf{seed} \in \{\mathsf{seed}_s, \mathsf{seed}_{e_1}, \ldots, \mathsf{seed}_{e_k}\}$ and a nonce $\mathsf{counter} \in \mathbb{Z}_{>0}$, and outputs a polynomial in $\mathcal{R}$ sampled according to $\mathcal{D}_\sigma$.
- The encoding function $\mathsf{Enc}\colon \{0,1\}^\kappa \to \{0,\ldots,n-1\}^h \times \{-1,1\}^h$ encodes a κ-bit hash value c' as a polynomial $c \in \mathbb{H}_{n,h}$. The polynomial c is represented as the two arrays $pos\_list \in \{0,\ldots,n-1\}^h$ and $sign\_list \in \{-1,1\}^h$, containing the positions and signs of its nonzero coefficients, respectively.
- The sampling function $\mathsf{ySampler}\colon \{0,1\}^\kappa \times \mathbb{Z} \to \mathcal{R}_{[B]}$ samples a polynomial $y \in \mathcal{R}_{[B]}$, taking as inputs a κ-bit seed rand and a nonce $\mathsf{counter} \in \mathbb{Z}_{>0}$.
- The hash-based function $\mathsf{H}\colon \mathcal{R}_q^k \times \{0,1\}^{320} \times \{0,1\}^{320} \to \{0,1\}^\kappa$. This function takes as inputs k polynomials $v_1,\ldots,v_k \in \mathcal{R}_q$ and first computes $[v_1]_M, \ldots, [v_k]_M$. The result is then hashed together with the hash $\mathsf{G}(m)$ for a given message m and the hash $\mathsf{G}(t_1,\ldots,t_k)$ to a string κ bits long.
- The correctness check function checkE, which gets an error polynomial e as input and rejects it by returning 1 if $\sum_{k=1}^{h} \max_k(e)$ is greater than some bound $L_E = E$[4]. Otherwise, it accepts it and returns 0. The function checkE guarantees the correctness of the signature scheme by ensuring $\|e_i c\|_\infty \leq L_E$.
- The simplification check function checkS, which gets a secret polynomial s as input and rejects it by returning 1 if $\sum_{k=1}^{h} \max_k(s)$ is greater than some bound $L_S = S$. Otherwise, it accepts it and returns 0. checkS ensures $\|sc\|_\infty \leq L_S$, which is used to simplify the security reduction.

The pseudocode of qTESLA's key generation, signing, and verification is depicted in Algorithms 1, 2, and 3, respectively.

Correctness. To guarantee the correctness of qTESLA it must hold for a signature (z, c') of a message m generated by Algorithm 2 that (i) $z \in \mathcal{R}_{[B-S]}$ and that (ii) the output of the hash-based function H at signing (line 9 of Algorithm 2) is the same as the analogous output at verification (line 6 of Algorithm 3). Requirement (i) is ensured by line 12 of Algorithm 2. To ensure (ii), the correctness check at signing is used (line 18 of Algorithm 2). Essentially, it ensures that for $[a_i z - t_i c]_M = [a_i(y+sc) - (a_i s + e_i)c]_M = [a_i y - e_i c]_M = [a_i y]_M$. A formal correctness proof can be found in the full version of this paper [4, Appendix A].

Design Features. qTESLA's design comes with several built-in security features. First, the public polynomials $a_1,\ldots,a_k$ are freshly generated at each key generation, using the random seed seed_a. This seed is stored as part of both sk and pk so that the signing and verification operations can regenerate $a_1,\ldots,a_k$. This makes the introduction of backdoors more difficult and reduces drastically the scope of all-for-the-price-of-one attacks [6,8]. Moreover, storing only a seed permits to save bandwidth since we only need κ bits to store seed_a instead of the $kn\lceil \log_2(q) \rceil$ bits required to represent the full polynomials.

[4] In an earlier version of this document we needed to distinguish L_S/L_E and S/E. Although this is not necessary in this version, we keep all four values L_S, S, L_E, E for consistency reasons.

Algorithm 1. qTESLA's Key Generation

Require: -

Ensure: key pair (sk, pk) with secret key $sk = (s, e_1, \ldots, e_k, \mathsf{seed}_a, \mathsf{seed}_y, g)$ and public key $pk = (t_1, \ldots, t_k, \mathsf{seed}_a)$

1: $\mathsf{counter} \leftarrow 1$
2: $\mathsf{pre\text{-}seed} \leftarrow_{\$} \{0,1\}^{\kappa}$
3: $\mathsf{seed}_s, \mathsf{seed}_{e_1}, \ldots, \mathsf{seed}_{e_k}, \mathsf{seed}_a, \mathsf{seed}_y \leftarrow \mathsf{PRF}_1(\mathsf{pre\text{-}seed})$ } Generating $a_1, \ldots, a_k$.
4: $a_1, \ldots, a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$
5: **do**
6: $\quad s \leftarrow \mathsf{GaussSampler}(\mathsf{seed}_s, \mathsf{counter})$ } Sampling $s \leftarrow_{\sigma} \mathcal{R}$.
7: $\quad \mathsf{counter} \leftarrow \mathsf{counter} + 1$
8: **while** $\mathsf{checkS}(s) \neq 0$
9: **for** $i = 1, \ldots, k$ **do**
10: $\quad$ **do**
11: $\quad\quad e_i \leftarrow \mathsf{GaussSampler}(\mathsf{seed}_{e_i}, \mathsf{counter})$ } Sampling $e_1, \ldots, e_k \leftarrow_{\sigma} \mathcal{R}$.
12: $\quad\quad \mathsf{counter} \leftarrow \mathsf{counter} + 1$
13: $\quad$ **while** $\mathsf{checkE}(e_i) \neq 0$
14: $\quad t_i \leftarrow a_i s + e_i \mod q$
15: **end for**
16: $g \leftarrow \mathsf{G}(t_1, \ldots, t_k)$
17: $sk \leftarrow (s, e_1, \ldots, e_k, \mathsf{seed}_a, \mathsf{seed}_y, g)$ } Return pk and sk.
18: $pk \leftarrow (t_1, \ldots, t_k, \mathsf{seed}_a)$
19: **return** sk, pk

To protect against KS attacks [11], we include the hash G of the polynomials $t_1, \ldots, t_k$ (which are part of the public key) in the secret key, in order to use it during the hashing operation to derive c'. This guarantees that any attempt by an attacker of modifying the public key will be detected during verification when checking the value c' (line 6 of Algorithm 3).

Also, the seed used to generate the randomness y at signing is produced by hashing the value seed_y that is part of the secret key, some fresh randomness r, and the digest $\mathsf{G}(m)$ of the message m. The use of seed_y makes qTESLA resilient to a catastrophic failure of the Random Number Generator (RNG) during generation of the fresh randomness, protecting against fixed-randomness attacks such as the one demonstrated against Sony's Playstation 3 [14]. Likewise, the random value r guarantees the use of a fresh y at each signing operation, which makes qTESLA's signatures *probabilistic*. Probabilistic signatures are, arguably, more difficult to attack through side-channel analysis. Moreover, the fresh y prevents some easy-to-implement but powerful fault attacks against deterministic signature schemes [13,38]; see [13, Sect. 6] for a relevant discussion.

Another design feature of qTESLA is that discrete Gaussian sampling, arguably the most complex function in many lattice-based signature schemes, is only required during key generation, while signing and verification, the most used functions of digital signature schemes, only use very simple arithmetic operations that are easy to implement. This facilitates the realization of compact and portable implementations that achieve high performance.

Algorithm 2. qTESLA's Signature Generation

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

1: $\mathsf{counter} \leftarrow 1$
2: $r \leftarrow_{\$} \{0,1\}^{\kappa}$
3: $\mathsf{rand} \leftarrow \mathsf{PRF}_2(\mathsf{seed}_y, r, \mathsf{G}(m))$ } Sampling $y \leftarrow_{\$} \mathcal{R}_{[B]}$.
4: $y \leftarrow \mathsf{ySampler}(\mathsf{rand}, \mathsf{counter})$
5: $a_1, \ldots, a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$
6: **for** $i = 1, \ldots, k$ **do**
7: $\quad v_i = a_i y \bmod^{\pm} q$
8: **end for**
9: $c' \leftarrow \mathsf{H}(v_1, \ldots, v_k, \mathsf{G}(m), g)$ } Computing the hash value.
10: $c \triangleq \{pos\_list, sign\_list\} \leftarrow \mathsf{Enc}(c')$ } Generating the sparse polynomial c.
11: $z \leftarrow y + sc$ } Computing the potential signature (z, c').
12: **if** $z \notin \mathcal{R}_{[B-S]}$ **then**
13: $\quad \mathsf{counter} \leftarrow \mathsf{counter} + 1$ } Ensuring security (the "rejection sampling").
14: $\quad$ Restart at step 4
15: **end if**
16: **for** $i = 1, \ldots, k$ **do**
17: $\quad w_i \leftarrow v_i - e_i c \bmod^{\pm} q$
18: $\quad$ **if** $\|[w_i]_L\|_\infty \geq 2^{d-1} - E \vee \|w_i\|_\infty \geq \lfloor q/2 \rfloor - E$ **then**
19: $\quad\quad \mathsf{counter} \leftarrow \mathsf{counter} + 1$ } Ensuring correctness.
20: $\quad\quad$ Restart at step 4
21: $\quad$ **end if**
22: **end for**
23: **return** (z, c') } Returning the signature for m.

3.1 Parameter Description

qTESLA's system parameters and their corresponding bounds are summarized in Table 1.

The parameter λ is defined as the security parameter, i.e., the targeted bit security of a given instantiation. In the standard R-LWE setting, we have $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1 \rangle$, where the dimension n is a power-of-two, i.e., $n = 2^\ell$ for $\ell \in \mathbb{N}$. Depending on the specific function, the parameter κ defines the input and/or output lengths of the hash-based and pseudorandom functions. This parameter is specified to be larger or equal to the security level λ. This is consistent with the use of the hash in a Fiat-Shamir style signature scheme such as qTESLA, for which preimage resistance is relevant while collision resistance is much less. Accordingly, we take the hash size to be enough to resist preimage attacks.

The parameter $b_{\mathsf{GenA}} \in \mathbb{Z}_{>0}$ represents the number of blocks requested in the first call to cSHAKE128 during the generation of the public polynomials $a_1, \ldots, a_k$. The values of b_{GenA} are chosen experimentally such that they maximize performance on the targeted Intel platform; see Sect. 5.

The Modulus q. This parameter is chosen to fulfill several bounds and assumptions that are motivated by efficiency requirements and qTESLA's security reduc-

Algorithm 3. qTESLA's Signature Verification

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

1: $c \triangleq \{pos\_list, sign\_list\} \leftarrow \mathsf{Enc}(c')$
2: $a_1, \ldots, a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$
3: **for** $i = 1, \ldots, k$ **do**
4: $\quad w_i \leftarrow a_i z - t_i c \bmod^{\pm} q$
5: **end for**
6: **if** $z \notin \mathcal{R}_{[B-S]} \vee c' \neq \mathsf{H}(w_1, \ldots, w_k, \mathsf{G}(m), \mathsf{G}(t_1, \ldots, t_k))$ **then**
7: $\quad$ **return** -1 } Reject signature (z, c') for m.
8: **end if**
9: **return** 0 } Accept signature (z, c') for m.

tion. To enable the use of fast polynomial multiplication using the NTT, we choose q to be a prime integer such that $q \bmod 2n = 1$. Moreover, we choose $q > 2B$. To choose parameters according to the security reduction, it is first convenient to simplify our security statement. To this end we ensure that $q^{nk} \geq |\Delta\mathbb{S}| \cdot |\Delta\mathbb{L}| \cdot |\Delta\mathbb{H}|$; see Table 1 for the definition of the respective sets. Then, the following equation (see Theorem 1) has to hold:

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

Following NIST's call for proposals [37, Section 4.A.4], we choose the number of sign queries to be $q_s = \min\{2^{\lambda/2}, 2^{64}\}$ and the number of hash queries to be $q_h = \min\{2^{\lambda}, 2^{128}\}$.

Bound Parameters and Acceptance Probabilities. The values L_S and L_E are used to bound the coefficients of the secret and error polynomials in the evaluation functions checkS and checkE, 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 as explained in Sect. 4. Both bounds, L_S and L_E impact the rejection probability during signature generation as follows. If one increases the values of L_S and L_E, the acceptance probability during key generation, referred to as δ_{keygen}, increases (see lines 8 and 13 in Algorithm 1), while the acceptance probabilities of z and w during signature generation, referred to as δ_z and δ_w resp., decrease (see lines 12 and 18 in Algorithm 2). We determine a good trade-off between the two acceptance probabilities. The values for δ_z, δ_w, δ_{sign} (overall acceptance probability during sign), and δ_{keygen} that were obtained for the proposed parameter sets are displayed in Table 2.

Key and Signature Sizes. The theoretical bitlengths of the signatures and public keys are given by $\kappa + n \cdot (\lceil \log_2(B - S) \rceil + 1)$ and $k \cdot n \cdot (\lceil \log_2(q) \rceil) + \kappa$, respectively. To determine the size of the secret keys we first define t as the number of β-bit entries of the discrete Gaussian sampler's CDT tables (see Table 3) which

Table 1. Description and bounds of all the system parameters.

Param.	Description	Requirement
λ	security parameter	-
q_h, q_s	#hash and sign queries	-
n	dimension	2^ℓ
σ	standard deviation of $\mathcal{D}_\sigma$	-
k	#public polynomials $a_1, \ldots, a_k$	-
q	modulus	$q > 2^{d+1}$, $q^{nk} \geq \lvert\Delta\mathbb{S}\rvert \cdot \lvert\Delta\mathbb{L}\rvert \cdot \lvert\Delta\mathbb{H}\rvert$, $q > 2B$, $q = 1 \bmod 2n$, $q^{nk} \geq 2^{4\lambda+nkd} 4q_s^3(q_s+q_h)^2$
h	#nonzero entries in Enc's output	$2^h \cdot \binom{n}{h} \geq 2^{2\lambda}$
κ	out-/input length of different functions	$\kappa \geq \lambda$
L_E, η_E	bound in checkE	$\lceil \eta_E \cdot h \cdot \sigma \rceil$
L_S, η_S	bound in checkS	$\lceil \eta_S \cdot h \cdot \sigma \rceil$
S, E	rejection parameters	$= L_S, L_E$
M^2	lower bound on the sign acceptance rate	-
B	determines randomness during sign	near a power-of-two, $B \geq \frac{\sqrt[n]{M}+2S-1}{2(1-\sqrt[n]{M})}$
d	#rounded bits	$d > \log_2(B), d \geq \log_2\left(\frac{2E+1}{1-M^{\frac{1}{nk}}}\right)$
b_{GenA}	#blocks requested to SHAKE128	$b_{\mathsf{GenA}} \in \mathbb{Z}_{>0}$
$\lvert\Delta\mathbb{H}\rvert$	$\Delta\mathbb{H} = \{c - c' \ : \ c, c' \in \mathbb{H}_{n,h}\}$	$\sum_{j=0}^{h} \sum_{i=0}^{h-j} \binom{kn}{2i} 2^{2i} \binom{kn-2i}{j} 2^j$
$\lvert\Delta\mathbb{S}\rvert$	$\Delta\mathbb{S} = \{z - z' \ : \ z, z' \in \mathcal{R}_{[B-S]}\}$	$(4(B-S)+1)^n$
$\lvert\Delta\mathbb{L}\rvert$	$\Delta\mathbb{L} = \{x - x' : x, x' \in \mathcal{R}, \ [x]_M = [x']_M\}$	$(2^d+1)^{nk}$
$\lvert\mathsf{sig}\rvert$	theoretical size of signature [bits]	$\kappa + n(\lceil \log_2(B-S) \rceil + 1)$
$\lvert\mathsf{pk}\rvert$	theoretical size of public key [bits]	$kn(\lceil \log_2(q) \rceil) + \kappa$
$\lvert\mathsf{sk}\rvert$	theoretical size of secret key [bits]	$n(k+1)(\lceil \log_2(t-1) \rceil + 1) + 2\kappa + 320$, $t = 78$ or 111

corresponds to the maximum value that can be possibly sampled to generate the coefficients of secret polynomials s. Then, it follows that the theoretical size of the secret key is given by $n(k+1)(\lceil \log_2(t-1) \rceil + 1) + 2\kappa + 320$ bits.

4 Security and Instantiations of qTESLA

4.1 Provable Security in the QROM

The standard security requirement for signature schemes, namely Existential Unforgeability under Chosen-Message Attack (EUF-CMA), dates back to Goldwasser, Micali, and Rivest [24]: The adversary can obtain q_S signatures via signing oracle queries on messages of their own choosing, and must output one valid signature on a message not queried to the oracle.

The EUF-CMA security of qTESLA is supported by a reduction in the QROM [12], in which the adversary is granted access to a quantum random oracle. Namely, Theorem 1 gives a reduction from the R-LWE problem to the EUF-CMA security of qTESLA in the QROM. It is very similar to [5, Theorem 1], which gives the security reduction for qTESLA's predecessor TESLA. It is important to note that to port the reduction idea from TESLA over standard lattices

to qTESLA over ideal lattices, we assume a conjecture to hold. The formal statement of qTESLA's security and a sketch of the proof, together with the required conjecture, are given in the full version of this paper [4, Section 5.1].

Theorem 1 (Security reduction from R-LWE). *Let the parameters be as in Table 1, in particular, let* $q^{nk} \geq 2^{4\lambda+nkd} 4q_s^3(q_s+q_h)^2$. *Assume that [4, Conjecture 1] holds. Assume that there exists a quantum adversary* $\mathcal{A}$ *that forges a* qTESLA *signature in time* t_Σ, *making at most* q_h *(quantum) queries to its quantum random oracle and* q_s *(classical) queries to its signing oracle. Then there exists a reduction* $\mathcal{S}$ *that solves the R-LWE problem in time* t_{LWE} *which is about the same as* t_Σ *in addition to the time to simulate the quantum random oracle and with*

$$\mathrm{Adv}^{\text{EUF-CMA}}_{\texttt{qTESLA}}(\mathcal{A}) \leq \mathrm{Adv}^{\text{R-LWE}}_{k,n,q,\sigma}(\mathcal{S}) + \frac{2^{3\lambda+nkd} \cdot 4 \cdot q_s^3(q_s+q_h)^2}{q^{nk}} + \frac{2(q_h+1)}{\sqrt{2^h\binom{n}{h}}}. \quad (1)$$

With parameters corresponding to Table 1, this security reduction is tight and explicit, allowing for efficient provably-secure parameters as explained next.

4.2 qTESLA's Security and the R-LWE Hardness

Our parameters are chosen such that $\epsilon_{\text{LWE}} \approx \epsilon_\Sigma$ and $t_\Sigma \approx t_{\text{LWE}}$[5], which guarantees that the bit hardness of the R-LWE instance is *theoretically* almost the same as the bit security of our signature scheme, by virtue of the security reduction and its tightness. The reduction provably guarantees that the scheme has the selected security level as long as the corresponding R-LWE instance gives the assumed hardness level and the aforementioned conjecture holds. This approach provides a strong security argument.

We emphasize that our provably secure parameters are chosen according to their security reductions from R-LWE but not according to reductions from underlying existing worst-case to average-case reductions from SIVP or GapSVP to R-LWE [35]. In this work, we propose two provably secure parameter sets called qTESLA-p-I and qTESLA-p-III; see Sect. 4.4.

Remark 1. In practical instantiations of qTESLA, the bit security does not exactly match the bit hardness of R-LWE (see Table 2). This is because the bit security does not only depend on the bit hardness of R-LWE, 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 the key space is reduced by the rejection of polynomials $s, e_1, \ldots, e_k$ with large coefficients via checkE and checkS. In particular, depending on the instantiation, the size of the key space is decreased by $\lceil |\log_2(\delta_{\mathsf{KeyGen}})| \rceil$ bits. We compensate this security loss by choosing an R-LWE instance of larger bit hardness. Hence, the corresponding

[5] To be precise, we assume that the time to simulate the (quantum) random oracle is smaller than the time to forge a signature. This assumption is commonly made in "provably secure" cryptography.

R-LWE instances give at least $\lambda + \lceil |\log_2(\delta_{\mathsf{KeyGen}})| \rceil$ bits of hardness against currently known (classical and quantum) attacks. Finally, we instantiate the encoding function Enc such that it is λ-bit secure.

4.3 Hardness Estimation of Our Instances

Lattice reduction is arguably the most important building block in most efficient attacks against R-LWE instances. As the Block-Korkine-Zolotarev algorithm (BKZ) [15,16] is considered the most efficient lattice reduction in practice, the model used to estimate the cost of BKZ determines the overall hardness estimation. While many different cost models for BKZ exist [2], we decided to adopt the BKZ cost model of $0.265\beta + 16.4 + \log_2(8d)$ for the hardness estimation of our parameters (denoted by `BKZ.qsieve`), where β is the BKZ block size and d is the lattice dimension. It corresponds to solving instances of the shortest vector problem of blocksize β with a quantum sieving algorithm [30,31]. This cost model is conservative since it only takes into account the number of operations needed to solve a certain instance and assumes that the attacker can handle huge amounts of quantum memory. In the full version of this paper [4, Table 3], we compare our chosen hardness estimation for R-LWE with other BKZ models, including the one from [6] (denoted by `BKZ.ADPS16`) and the classical algorithms using sieving [9] (denoted by `BKZ.sieve`).

Since its introduction in [35], it has remained an open question to determine whether the R-LWE problem is as hard as the LWE problem for instances typically used in signature schemes. Several results exist that exploit the structure of some ideal lattices, e.g., [17,23]. However, up to now, these results do not seem to apply to R-LWE instances that are typically used in practice. 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. In particular, we integrated the *LWE-Estimator* [3] with commit-id `3019847` on 2019-02-14 in the sage script that we wrote to perform the security estimation.

4.4 Parameter Sets

We propose two parameter sets called `qTESLA-p-I` and `qTESLA-p-III`, which match the security of NIST levels 1 and 3 [37], respectively; see Table 2.

5 Implementation and Performance Evaluation

5.1 Portable C Implementation

Our compact reference implementation is written exclusively in portable C using approximately 300 lines of code. It exploits the fact that it is straightforward to write a `qTESLA` implementation with a common codebase, since the different parameter set realizations only differ in some packing functions and system constants that can be instantiated at compilation time. This illustrates the simplicity and scalability of software based on `qTESLA`.

Table 2. Proposed parameter sets with $\kappa = 256$.

Parameter	qTESLA-p-I	qTESLA-p-III
λ	95	160
n, k	1 024, 4	2 048, 5
σ	8.5	8.5
q	343 576 577 $\approx 2^{28}$	856 145 921 $\approx 2^{30}$
h	25	40
$L_E = E = L_S = S$	554	901
B	$2^{19} - 1$	$2^{21} - 1$
d	22	24
b_{GenA}	108	180
δ_w, δ_z	0.37, 0.34	0.33, 0.42
δ_{sign}, M	0.13, 0.3	0.14, 0.3
δ_{keygen}	0.59	0.43
sig size [bytes]	2, 592	5, 664
pk size [bytes]	14, 880	38, 432
sk size [bytes]	5, 224	12, 392
Quantum bit hardness	139	279

Protection Against Side-Channel Attacks. Our implementations run in *constant-time*, i.e., they avoid the use of secret address accesses and secret branches and, hence, are protected against timing and cache side-channel attacks. The following functions are implemented securely via constant-time logical and arithmetic operations: H, checkE, checkS, the correctness test for rejection sampling, polynomial multiplication using the NTT, sparse multiplication, and all the polynomial operations requiring modular reductions or corrections. Some of the functions that perform some form of rejection sampling, such as the security test at signing, GenA, ySampler, and Enc, potentially leak the timing of the failure to some internal test, but this information is independent of the secret data. Table lookups performed in our implementation of the Gaussian sampler are done with linear passes over the full table and producing samples via constant-time logical and arithmetic operations.

Extendable Output Functions. Several functions used for the implementation of qTESLA require hashing and pseudorandom bit generation. This functionality is provided by so-called Extendable Output Functions (XOFs). For qTESLA we use the XOF function SHAKE [22] in the realization of the functions G and H, and cSHAKE128 [28] in the realization of the functions GenA and Enc. To implement the functions PRF_1, PRF_2, ySampler, and GaussSampler, implementers are free to pick a cryptographic PRF of their choice. For simplicity purposes, in our implementations we use SHAKE (in the case of PRF_1 and PRF_2) and cSHAKE (in the case of ySampler and GaussSampler). With the exception of GenA and Enc

(which always use cSHAKE128), our level 1 parameter set uses (c)SHAKE128 and our level 3 set uses (c)SHAKE256.

Polynomial Arithmetic. Our polynomial arithmetic, which is dominated by polynomial multiplications based on the NTT, uses a signed 32-bit datatype to represent coefficients. Throughout polynomial computations, intermediate results are let to grow and are only reduced or corrected when there is a chance of exceeding 32 bits of length, after a multiplication, or when a result needs to be prepared for final packing (e.g., when outputting public keys). Accordingly, to avoid overflows the results of additions and subtractions are either corrected or reduced via Barrett reductions whenever necessary. We have performed a careful bound analysis for each of the proposed parameter sets in order to maximize the use of lazy reduction and cheap modular corrections in the polynomial arithmetic. In the case of multiplications, the results are reduced via Montgomery reductions. To minimize the cost of converting to/from Montgomery representation we use the following approach. First, the so-called "twiddle factors" in the NTT are scaled *offline* by multiplying with the Montgomery constant $R = 2^{32} \bmod q$. Similarly, the coefficients of the outputs a_i from GenA are scaled to remainders $r' = rn^{-1}R \bmod q$ by multiplying with the constant $R^2 \cdot n^{-1}$. This enables an efficient use of Montgomery reductions during the NTT-based polynomial multiplication $\mathsf{NTT}^{-1}(\tilde{a} \circ \mathsf{NTT}(b))$, where $\tilde{a} = \mathsf{NTT}(a)$ is the output of GenA which is assumed to be in NTT domain. Multiplications with the twiddle factors during the computation of $\mathsf{NTT}(b)$ naturally cancel out the Montgomery constant. The same happens during the pointwise multiplication with $\tilde{a}$, and finally during the inverse NTT, which naturally outputs values in standard representation without the need for explicit conversions.

To compute the power-of-two NTT in our implementations, we adopt butterfly algorithms that efficiently merge the powers of ϕ and ϕ^{-1} with the powers of ω, and that at the same time avoid the need for the so-called bit-reversal operation which is required by some implementations, e.g., [6]. 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 bit-reversed ordering. The algorithm receives the inputs of a polynomial $\tilde{a}$ in bit-reversed ordering and produces an output in standard ordering. Polished versions of these well-known algorithms, which we follow in our implementations, can be found in [41, Alg. 1 and 2].

While standard polynomial multiplications can be efficiently carried out using the NTT as explained above, *sparse multiplications* with a polynomial $c \in \mathbb{H}_{n,h}$ can be realized more efficiently with a specialized algorithm that exploits the sparseness of the input.

Gaussian Sampling. One of the advantages of `qTESLA` is that Gaussian sampling is only required during key generation. Nevertheless, certain applications might

Table 3. CDT parameters used in qTESLA.

Parameter set	Bit precision		#rows in CDT	Size of CDT [byte]
	Targeted	Implemented		
qTESLA-p-I	64	63	78	624
qTESLA-p-III	128	125	111	1776

still require an efficient and secure implementation of key generation and one that is, in particular, portable and protected against timing and cache side-channel attacks. Accordingly, we employ a *constant-time* Gaussian sampler based on the well-established technique of Cumulative Distribution Table (CDT) of the normal distribution, which consists of precomputing, to a given β-bit precision, a table $\mathsf{CDT}[i] := \lfloor 2^{\beta} \Pr[c \leqslant i \mid c \leftarrow_{\sigma} \mathbb{Z}] \rfloor$, for $i \in [-t+1 \ldots t-1]$ with the smallest t such that $\Pr[|c| \geqslant t \mid c \leftarrow_{\sigma} \mathbb{Z}] < 2^{-\beta}$. To obtain a Gaussian sample, one picks a uniform sample $u \leftarrow_{\$} \mathbb{Z}/2^{\beta}\mathbb{Z}$, looks it up in the table, and returns the value z such that $\mathsf{CDT}[z] \leqslant u < \mathsf{CDT}[z+1]$. In the case of qTESLA, this method is very efficient due to the values of σ being relatively small, as can be seen in Table 2.

In our implementations, the CDT method is implemented by generating a chunk of $c \mid n$ samples at a time, where we fix $c = 512$. Then, to generate each sample in a chunk the precomputed CDT table is fully scanned, using constant-time logical and arithmetic operations to produce a Gaussian sample. For the precomputed CDT tables, the targeted sampling precision β is conservatively set to a value much greater than $\lambda/2$, as can be seen in Table 3.

5.2 AVX2 Optimizations

We wrote an assembly implementation of the polynomial multiplication to speed up its execution with the use of AVX2 vector instructions. Our polynomial multiplication follows the recent approach by Seiler [41], and the realization of the method has some similarities with the implementation from [21]. That is, our implementation processes 32 coefficients loaded in 8 AVX2 registers simultaneously, in such a way that butterfly computations are carried out through multiple NTT levels without the need for storing and loading intermediate results, whenever possible. Although there are 32 coefficients in the AVX2 registers, the butterfly operation needs its inputs have distance bigger than 32 for some levels. Thus, we combine 5 levels whenever possible, and up to 3 levels for the rest. To avoid in-register operations during the combined 5 levels, we shuffle coefficients such that the 4 subsequent butterfly operations are performed in parallel in the registers.

One difference with [21,41] is that our NTT coefficients are represented as 32-bit *signed* integers, which motivates a speedup in the butterfly computation by avoiding the extra additions that are required to make the result of subtractions positive when using an unsigned representation. Our approach reduces the cost

of the portable C polynomial multiplication from 76,300 to 18,400 cycles for $n = 1024$, and from 174,800 to 43,900 cycles for $n = 2048$.

In addition, to speed up the sampling of y we use the AVX2 implementation of SHAKE by [10], which enables sampling of up to 4 coefficients in parallel.

Table 4. Performance (in thousands of cycles) of the portable C and the AVX2 implementations of `qTESLA` on a 3.4 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.

	Scheme	`keygen`	`sign`	`verify`
C	`qTESLA-p-I`	2,358.6 (2,431.9)	2,299.0 (3,089.9)	814.3 (814.5)
	`qTESLA-p-III`	13,151.4 (13,312.4)	5,212.3 (7,122.6)	2,102.3 (2,102.6)
AVX2	`qTESLA-p-I`	2,212.4 (2,285.0)	1,370.4 (1,759.0)	678.4 (678.5)
	`qTESLA-p-III`	12,791.0 (13,073.4)	3,081.9 (4,029.5)	1,745.3 (1,746.4)

5.3 Performance on x64

We evaluated the performance of our implementations on an x64 machine powered by a 3.4 GHz Intel Core i7-6700 (Skylake) processor 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`. The results for the portable C and AVX2 implementations are summarized in Table 4. `qTESLA` computes the combined (median) time of signing and verification on the Skylake platform in approximately 0.92 and 2.15 ms with `qTESLA-p-I` and `qTESLA-p-III`, respectively. This demonstrates that the speed of `qTESLA`, although slower than other lattice-based signature schemes, can still be considered practical for most applications.

The AVX2 optimizations improve the performance by a factor 1.5x, approximately. The speedup is mainly due to the AVX2 implementation of the polynomial multiplication, which is responsible for ∼70% of the total speedup. `qTESLA` computes the combined (median) time of signing and verification on the Skylake platform in approximately 0.60 and 1.42 ms with `qTESLA-p-I` and `qTESLA-p-III`, respectively.

We note that the overhead of including g, i.e., the hash of part of the public key, in the signature computation of c' is between 3–8% of the combined cost of signing and verification.

5.4 Comparison

Table 5 compares `qTESLA` to representative state-of-the-art post-quantum signature schemes in terms of bit security, signature and public key sizes, and performance of portable C reference and AVX2-optimized implementations (if

Table 5. Comparison of different post-quantum signature schemes.

	Scheme	Security [bit]	const. time	Sizes [B]		Cycle counts [k-cycles] Reference	AVX2	CPU
Selected lattice-based signatures	BLISS-BI [19,20]	128	✗	pk: 896 sig: 717	sign: ≈435.2 verify: ≈102.0		- -	U
	FALCON-512[a] [39]	158[b] (103)	✗	pk: 897 sig: 617	sign: 1,368.5 verify: 95.6		1,009.8 81.0	S
	Dilithium-II [34]	122[b] (91)	✓	pk: 1,184 sig: 2,044	sign: 1,378.1 verify: 272.8		410.7 109.0	S
	Dilithium-III [34]	160[b] (125)	✓	pk: 1,472 sig: 2,701	sign: 2,035.9 verify: 375.7		547.2 155.8	S
	qTESLA-p-I [a] (this paper)	95[b]	✓	pk: 14,880 sig: 2,592	sign: 3,089.9 verify: 814.3		1,759.0 678.5	S
	qTESLA-p-III [a] (this paper)	160[b]	✓	pk: 38,432 sig: 5,664	sign: 7,122.6 verify: 2,102.3		4,029.5 1,746.4	S
Others	SPHINCS$^+$-128f-s[a] (SHAKE256) [26]	128[c]	✓	pk: 32 sig: 16,976	sign: 325,311 verify: 13,541		129,137 9,385	H
	MQDSS-31-64 [40]	128[c]	✓	pk: 64 sig: 43,728	sign: 85,268.7 verify: 62,306.1		9,047.1 6,133.0	H

[a] Parameters are chosen according to given security reduction in the ROM/QROM.

[b] Bit security against classical and quantum adversaries with BKZ cost model $0.265\beta + 16.4 + \log_2(8d)$ [2]; (originally stated bit security given in brackets).

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

U: Unknown 3.4GHz Intel Core for BLISS.

S: 3.3GHz Intel Core i7-6567U (Skylake) for FALCON-512 (TurboBoost enabled), 2.6GHz Intel Core i7-6600U (Skylake) for Dilithium, and 3.4GHz Intel Core i7-6700 (Skylake) for qTESLA.

H: 3.5GHz Intel Core i7-4770K (Haswell).

available). If both median and average of cycle counts are provided in the literature, we report the average for signing and the median for verify. To have a fair comparison, we state the bit security of qTESLA, Falcon, and Dilithium assuming the same BKZ cost model of $0.265\beta + 16.4 + \log_2(8d)$ with β being the BKZ blocksize and d being the lattice dimension (for schemes that use other cost models, we write in brackets the bit security stated in the respective papers).

FALCON-512, the only other scheme proposing parameters according to their (tight) security reduction, features the smallest (pk + sig) size among all the post-quantum signature schemes shown in the table. However, Falcon has some shortcomings due to its high complexity. This scheme relies on very complex Fourier sampling methods and requires floating-point arithmetic, which is not supported by many devices. This makes the scheme significantly harder to implement in general, and hard to protect against side-channel and fault attacks in particular. The recent efficient implementation by Pornin [39] makes use of complicated floating-point emulation code to deal with the portability issues, and contains several thousands of lines of C code. Still, the software cannot be labeled as a strictly *constant-time* implementation because some portions of it allow a

limited amount of leakage to happen. This should be contrasted against the simple and compact implementation of `qTESLA`.

Schemes based on other underlying problems, such as SPHINCS$^+$ and MQDSS, offer compact public keys at the expense of having very large signature sizes. In contrast, `qTESLA` has smaller signature sizes, and is significantly faster for signing and verification.

In summary, `qTESLA` offers a good balance between efficiency, accompanied by a simple, compact, and secure design.

Acknowledgments. We are grateful to the anonymous reviewers for their valuable comments on earlier versions of this paper. We thank Vadim Lyubashevsky for pointing out that the heuristic parameters proposed in a previous paper version were lacking security estimates with respect to the SIS problem. We also thank Greg Zaverucha for bringing up the vulnerability of some signature schemes, including a previous version of qTESLA, to KS attacks, and for several fruitful discussions. We are thankful to Edward Eaton for his advice concerning the conjecture used in Theorem 1 and carrying out the supporting experiments, and to Joo Woo for pointing out an incorrectness in the conjecture. Finally, we thank Fernando Virdia, Martin Albrecht and Shi Bai for fruitful discussions and helpful advice on the hardness estimation of SIS for an earlier version of this paper.

References

1. Akleylek, S., Bindel, N., Buchmann, J., Krämer, J., Marson, G.A.: An efficient lattice-based signature scheme with provably secure instantiation. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2016. LNCS, vol. 9646, pp. 44–60. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31517-1_3
2. Albrecht, M.R., et al.: Estimate all the LWE, NTRU schemes!. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 351–367. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_19
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015)
4. Alkim, E., Barreto, P.S.L.M., Bindel, N., Krämer, J., Longa, P., Ricardini, J.E.: The lattice-based digital signature scheme qTESLA. Cryptology ePrint Archive, Report 2019/085 (2019). https://eprint.iacr.org/2019/085
5. Alkim, E., et al.: Revisiting TESLA in the quantum random oracle model. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 143–162. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_9
6. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 2016, pp. 327–343. USENIX Association (2016)
7. Bai, S., Galbraith, S.D.: An improved compression technique for signatures based on learning with errors. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 28–47. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04852-9_2
8. Barreto, P.S.L.M., Longa, P., Naehrig, M., Ricardini, J.E., Zanon, G.: Sharper ring-LWE signatures. Cryptology ePrint Archive, Report 2016/1026 (2016). http://eprint.iacr.org/2016/1026

9. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th SODA, pp. 10–24. ACM-SIAM, January 2016
10. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Assche, G.V., Keer, R.V.: The eXtended Keccak Code Package (XKCP). https://github.com/XKCP/XKCP
11. Blake-Wilson, S., Menezes, A.: Unknown key-share attacks on the station-to-station (STS) protocol. In: Imai, H., Zheng, Y. (eds.) PKC 1999. LNCS, vol. 1560, pp. 154–170. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49162-7_12
12. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_3
13. Bruinderink, L.G., Pessl, P.: Differential fault attacks on deterministic lattice signatures. IACR TCHES **2018**(3), 21–43 (2018). https://tches.iacr.org/index.php/TCHES/article/view/7267
14. Cantero, H., Peter, S., Bushing, S.: Console hacking 2010 - PS3 epic fail. In: 27th Chaos Communication Congress (2010). https://www.cs.cmu.edu/~dst/GeoHot/1780_27c3_console_hacking_2010.pdf
15. Chen, Y.: Réduction de réseau et sécurité concrète du chiffrement com-plètement homomorphe. Ph.D. thesis, Paris, France (2013)
16. Chen, Y., Nguyen, P.Q.: BKZ 2.0: better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_1
17. Cramer, R., Ducas, L., Peikert, C., Regev, O.: Recovering short generators of principal ideals in cyclotomic rings. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 559–585. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_20
18. Dagdelen, Ö., et al.: High-speed signatures from standard lattices. In: Aranha, D.F., Menezes, A. (eds.) LATINCRYPT 2014. LNCS, vol. 8895, pp. 84–103. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16295-9_5
19. Ducas, L.: Accelerating BLISS: the geometry of ternary polynomials. Cryptology ePrint Archive, Report 2014/874 (2014). http://eprint.iacr.org/2014/874
20. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_3
21. Ducas, L., et al.: CRYSTALS-Dilithium: a lattice-based digital signature scheme. IACR TCHES **2018**(1), 238–268 (2018). https://tches.iacr.org/index.php/TCHES/article/view/839
22. Dworkin, M.J.: SHA-3 standard: permutation-based hash and extendable-output functions. Federal Inf. Process. Stds. (NIST FIPS) - 202 (2015). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
23. Elias, Y., Lauter, K.E., Ozman, E., Stange, K.E.: Provably weak instances of ring-LWE. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 63–92. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_4
24. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)

25. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: a signature scheme for embedded systems. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 530–547. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_31
26. Hülsing, A., et al.: SPHINCS+. Technical report, National Institute of Standards and Technology (2019). https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
27. Jackson, D., Cremers, C., Cohn-Gordon, K., Sasse, R.: Seems legit: automated analysis of subtle attacks on protocols that use signatures. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, pp. 2165–2180. ACM, New York (2019)
28. Kelsey, J.: SHA-3 derived functions: cSHAKE, KMAC, TupleHash, and ParallelHash. NIST Special Publication, 800:185 (2016). http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf
29. Kiltz, E., Lyubashevsky, V., Schaffner, C.: A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 552–586. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_18
30. Laarhoven, T.: Search problems in cryptography. Ph.D. thesis, Eindhoven University of Technology (2016)
31. Laarhoven, T., Mosca, M., van de Pol, J.: Solving the shortest vector problem in lattices faster using quantum search. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 83–101. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_6
32. Lyubashevsky, V.: Fiat-Shamir with aborts: applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_35
33. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_43
34. Lyubashevsky, V., et al.: CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology (2019). https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
35. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
36. Menezes, A., Smart, N.P.: Security of signature schemes in a multi-user setting. Des. Codes Cryptogr. **33**(3), 261–274 (2004)
37. 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. Accessed 23 July 2018
38. Poddebniak, D., Somorovsky, J., Schinzel, S., Lochter, M., Rösler, P.: Attacking deterministic signature schemes using fault attacks. Cryptology ePrint Archive, Report 2017/1014 (2017). http://eprint.iacr.org/2017/1014
39. Pornin, T.: New efficient, constant-time implementations of Falcon (2019). https://falcon-sign.info/falcon-impl-20190918.pdf. Accessed 11 Oct 2019

40. Samardjiska, S., Chen, M.-S., Hülsing, A., Rijneveld, J., Schwabe, P.: MQDSS. Technical report, National Institute of Standards and Technology (2019). https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
41. Seiler, G.: Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. Cryptology ePrint Archive, Report 2018/039 (2018). https://eprint.iacr.org/2018/039