An Efficient and Provable Masked Implementation of qTESLA

François Gérard[1](✉) and Mélissa Rossi[2,3,4]

[1] Université libre de Bruxelles, Brussels, Belgium
fragerar@ulb.ac.be
[2] École normale supérieure, CNRS, PSL University, Paris, France
[3] Thales, Gennevilliers, France
[4] Inria, Paris, France
melissa.rossi@ens.fr

Abstract. Now that the NIST's post-quantum cryptography competition has entered in its second phase, the time has come to focus more closely on practical aspects of the candidates. While efficient implementations of the proposed schemes are somewhat included in the submission packages, certain issues like the threat of side-channel attacks are often lightly touched upon by the authors. Hence, the community is encouraged by the NIST to join the war effort to treat those peripheral, but nonetheless crucial, topics. In this paper, we study the lattice-based signature scheme `qTESLA` in the context of the masking countermeasure. Continuing a line of research opened by Barthe et al. at Eurocrypt 2018 with the masking of the GLP signature scheme, we extend and modify their work to mask `qTESLA`. Based on the work of Migliore et al. in ACNS 2019, we slightly modify the parameters to improve the masked performance while keeping the same security. The masking can be done at any order and specialized gadgets are used to get maximal efficiency at order 1. We implemented our countermeasure in the original code of the submission and performed tests at different orders to assess the feasibility of our technique.

Keywords: Lattice based signatures · Side-channels · Masking

1 Introduction

Following NIST's call for proposals a few years ago, the practical aspects of post-quantum cryptography have lately been studied more closely in the scientific literature. Many researchers tried to optimize parameters of cryptosystems to achieve reasonable practicality while still resisting state-of-the-art cryptanalysis. Once the design phase was over, a lot of implementations flourished on various platforms, proving that those cryptosystems can hope to achieve something useful outside of academia. Nevertheless, everyone is now well aware that having a fast and correct implementation of some functionality is seldom sufficient to get a secure system. In practice, side-channel attacks should not be overlooked and the capability of a cryptosystem to be easily protected against this kind of threats may be a strong argument to decide what will be the reigning algorithm in a post-quantum world.

In this work, we focus on applying the masking countermeasure to qTESLA [1], a Fiat-Shamir lattice-based signature derived from the original work of Lyubashevsky [22]. This signature is, with Dilithium [14], one of the most recent iteration of this line of research and a candidate for the NIST's competition. In 2018, Barthe et al. [3] described and implemented a proof of concept for a masked version of an ancestor of Dilithium/qTESLA called GLP [18]. Their goal was to prove that it is possible to mask the signature procedure at any order. This work led to a concrete masked implementation of Dilithium with experimental leakage tests [23]. In the latter, Migliore *et al.* noticed that replacing the prime modulus by a power of two allows to obtain a considerably more efficient masked scheme, by a factor of 7.3 to 9 for the most timeconsuming masking operations. Our work is in the same spirit. Similarly, we slightly modify the signature and parameters to ease the addition of the countermeasure while keeping the original security. In addition, we provide a detailed proof of masking for the whole signature process taking public outputs into account. Indeed, similarly to the masking of GLP in [3], several elements of qTESLA may be securely unmasked, like, for example, the number of rejections. Besides, we propose an implementation for which we have focused on *performance and reusability*. Our masked signature implementation still keeps the property of being compatible with the original verifying procedure of qTESLA and has been directly implemented within the code of the submission. Even if we target high order masking, we also implemented specialized gadgets for order 1 masking to provide a lightweight version of the masking scheme with reasonable performance fitting nicely on embedded systems. We finally provide extensive performance data and show that the cost of provable masking can be reasonable at least for small orders. Our code is publicly available at https://github.com/fragerar/Masked_qTESLA.

Parameter Sets Removal. While this paper was under peer review, the heuristic parameter sets on which our experiments are based were removed by the qTESLA team. We emphasis that the parameters we use were *not* broken but are not part of the standardization process anymore. Furthermore, our theoretical work is somewhat oblivious to the underlying parameter set used to instanciate the signature and the code can be adapted to implement the provably-secure sets as well.

2 Preliminaries

2.1 Notations

For any integers q, n and $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$, we denote by $\mathcal{R}_q$ the ring $\mathbb{Z}_q[X]/(X^n+1)$. Polynomials are written with bold lower case, e.g. $\mathbf{y} \in \mathcal{R}_q$. Note that, in our study, we do not need to introduce a notation for vectors of polynomials. Let B be an integer, we write $\mathcal{R}_{q,[B]}$ to denote the subset of polynomials in $\mathcal{R}_q$ with coefficients in $[-B, B]$. The usual norm operators are extended to polynomials by interpreting them as a vector of their coefficients. For a polynomial $\mathbf{v} = \sum_{i=0}^{n-1} v_i \cdot \mathbf{x}^i$, $||\mathbf{v}||_1 = \sum_{i=0}^{n-1} |v_i|$ and $||\mathbf{v}||_\infty = \mathsf{max}_i |v_i|$. For a modulus q and an integer x, we write $x \bmod q$ to denote the unique integer $x_{cn} \in [0, \ldots, q-1]$ such that $x_{cn} \equiv x \pmod q$. We call this integer the *canonical representative* of x modulo q. We also write $x \bmod^{\pm} q$ to denote the unique integer $x_{ct} \in (-q/2, \ldots, q/2]$ (where the lower bound is included if q is odd) such that $x_{ct} \equiv x \pmod q$. We call this integer the *centered representative* of x modulo q. For integers w, d, the function $[\cdot]_L : \mathbb{Z} \to \mathbb{Z}, w \mapsto w \bmod^{\pm} 2^d$ denotes the signed extraction of the d last bits of w. We use this function to define $[\cdot]_M : \mathbb{Z} \to \mathbb{Z}, w \mapsto (w \bmod^{\pm} q - [w]_L)/2^d$. Those two functions are extended to polynomials by applying them separately on each coefficient.

2.2 Masking

Side channel attacks are a family of cryptanalytic attacks where the adversary is able access several physical parameters of the device running the algorithm. These physical attacks include, for instance, cache attacks, simple and correlation electromagnetic analysis or fault injections. Modelling and protecting the information leaked though physical parameters has been an important research challenge since the original attack warning in [20].

The *probing model* or *ISW model* from its inventors [19] is the most studied leakage model. It has been introduced in order to theoretically define the vulnerability of implementations exposed to side-channel attacks. In a nutshell, a cryptographic implementation is N-probing secure iff any set of at most N intermediate variables is statisctically independent of the secrets. This model can be applied to practical leakages with the reduction established in [13] and tightened in [17]. The *masking* countermeasure performs computations on secret-shared data. It is the most deployed countermeasure in this landscape. Basically, each input secret x is split into $N+1$ variables $(x_i)_{0 \le i \le N}$ referred to as shares. N of them are generated uniformly at random whereas the last one is computed such that their combination reveals the secret value x. The integer N is called *masking order* and represents the security level of an implementation with respect to side channels. Let us introduce two types of additive combination in the following definition.

Definition 1 (Arithmetic and Boolean Masking). *A sensitive value x is shared with mod q arithmetic masking if it is split into $N+1$ shares $(x_i)_{0 \le i \le N}$ such that*

$$x = x_0 + \cdots + x_N \pmod q. \qquad \text{(Arithmetic masking mod } q\text{)}$$

It is shared with Boolean masking if it is split into $N+1$ shares $(x_i)_{0 \le i \le N}$ such that

$$x = x_0 \oplus \cdots \oplus x_N. \qquad \text{(Boolean masking)}$$

For lattice-based cryptography where most operations are linear for mod q addition, arithmetic masking seems the best choice. However, for certain operations like the randomness generation and comparisons, Boolean masking is better fit. Fortunately, some conversions exist [3,9,11] and allow to switch from one masking to another.

Proofs by Composition. To achieve N-probing security, Barthe et al. formally defined two security properties in [4], namely *non-interference* and *strong non-interference*, which (1) ease the security proofs for small gadgets (see Definition 2), and (2) allows to securely combine secure gadgets together.

Definition 2. *A (u, v)-gadget is a probabilistic algorithm that takes as inputs u shared values, and returns distributions over v-tuples of shared values.*

Definition 3. *A gadget is N-non-interfering (N-NI) iff any set of at most N observations can be perfectly simulated from at most N shares of each input.*

Definition 4. *A gadget is N-strong non-interfering (N-SNI) iff any set of at most N observations whose N_{int} observations on the internal data and N_{out} observations on the outputs can be perfectly simulated from at most N_{int} shares of each input.*

It is easy to check that N-SNI implies N-NI which implies N-probing security. The strong non-interference only appears in the proofs for subgadgets inside the signature and key generation algorithm. An additional notion was introduced in [3] to reason on the security of lattice-based schemes in which some intermediate variables may be revealed to the adversary.

Definition 5. *A gadget with public outputs X is N-non-interfering with public outputs (N-NIo) iff every set of at most N intermediate variables can be perfectly simulated with the public outputs and at most N shares of each input.*

Table 1. Parameters for qTESLA-I and qTESLA-III

Parameters	qTESLA-I	qTESLA-III	Description
n	512	1024	Dimension of the ring
q	$4\,205\,569 \approx 2^{22}$	$8\,404\,993 \approx 2^{23}$	Modulus
σ	22.93	10.2	Standard deviation
h	30	48	Nonzero entries of $\mathbf{c}$
E	1586	1147	Rejection parameter
S	1586	1233	Rejection parameter
B	$2^{20} - 1$	$2^{21} - 1$	Bound for $\mathbf{y}$
d	21	22	Bits dropped in $[\cdot]_M$

2.3 The qTESLA Signature

Let us now describe qTESLA [1], a (family of) lattice-based signature based on the RLWE problem and round 2 candidate for the NIST's post-quantum competition. The signature stems from several iterations of improvements over the original scheme of Lyubashevsky [22]. It is in fact a concrete instantiation of the scheme of Bai and Galbraith [2] over ideal lattices. Its direct contender in the competition is Dilithium [14] which is also based on this same idea of having a lattice variant of Schnorr signature. The security of Dilithium rely on problems over module lattices instead of ideal lattices, in the hope of increasing security by reducing algebraic structure, at the cost of a slight performance penalty.

To avoid overloading the paper, we will not describe in details all the subroutines and subtleties of qTESLA and sometimes simplify some aspects of the signature not required to understand our work.

Parameters
We store in Table 1 the set of selected parameters that are relevant for the rest of the paper. For the sake of practicability, we focus on the heuristic version of qTESLA in this work. More specifically, we implement our countermeasure in qTESLA-I and qTESLA-III even though the techniques we used are not specific to any parameter set.

Scheme
The key generation and signature procedures are formally recalled in Algorithms 1 and 2. They are similar to the corresponding ones in other Fiat-Shamir lattice-based signatures. We redirect the interested reader to [1] or the NIST submission [5] for a detailed description. In the following, PRF is a pseudorandom function, GenA generates a uniformly random polynomial, GaussSampler samples a polynomial according to a Gaussian distribution, CheckS and CheckE verifies that a secret polynomial does not have too large coefficients, ySampler samples a uniformly random polynomial $\mathbf{y} \in \mathcal{R}_{q,[B]}$, H is a collision resistant hash function and Enc encodes a bitstring into a sparse polynomial $\mathbf{c} \in \mathcal{R}_{q,[1]}$ with $||\mathbf{c}||_1 = h$.

3 Masked qTESLA

3.1 Masking-Friendly Design

In the process of masking qTESLA, we decided to make slight modifications in the signing procedure in order to facilitate masking. The idea is that some design elements providing small efficiency gains may be really hard to carry on to the masked version and actually do even more harm than good. Our two main modifications are the modulus which is chosen as the closest power of two of the original parameter set and the removal of the PRF to generate the polynomial $\mathbf{y}$.

Power of Two Modulus. Modular arithmetic is one of the core component of plenty of cryptographic schemes. While, in general, it is reasonably fast for any

Algorithm 1. qTESLA key generation

```
Result: $sk = (\mathbf{s}, \mathbf{e}, \text{seed}_a, \text{seed}_y)$,
$pk = (seed_a, \mathbf{t})$

1: counter $\leftarrow 1$
2: pre-seed $\overset{r}{\leftarrow} \{0,1\}^{\kappa}$
3: $\text{seed}_{s,e,a,y} \leftarrow \mathsf{PRF}(\text{pre-seed})$
4: $\mathbf{a} \leftarrow \mathsf{GenA}(\text{seed}_a)$
5: do
6:     $\mathbf{s} \leftarrow \mathsf{GaussSampler}(\text{seed}_s, \text{counter})$
7:     counter $\leftarrow$ counter + 1
8: while ($\mathsf{CheckS}(\mathbf{s}) \neq 0$)
9: do
10:    $\mathbf{e} \leftarrow \mathsf{GaussSampler}(\text{seed}_e, \text{counter})$
11:    counter $\leftarrow$ counter + 1
12: while ($\mathsf{CheckE}(\mathbf{e}) \neq 0$)
13: $\mathbf{t} \leftarrow \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \bmod q$
14: $sk \leftarrow (\mathbf{s}, \mathbf{e}, \text{seed}_a, \text{seed}_y)$
15: $pk \leftarrow (seed_a, \mathbf{t})$
16: return $sk, pk$
```

Algorithm 2. qTESLA sign

```
Data: $sk = (\mathbf{s}, \mathbf{e}, \text{seed}_a, \text{seed}_y)$
Result: $\Sigma = (\mathbf{z}, \mathbf{c})$
1: counter $\leftarrow 1$
2: $r \overset{r}{\leftarrow} \{0,1\}^{\kappa}$
3: rand $\leftarrow \mathsf{PRF}(\text{seed}_y, r, \mathsf{H}(m))$
4: $\mathbf{y} \leftarrow \mathsf{ySampler}(\text{rand, counter})$
5: $\mathbf{a} \leftarrow \mathsf{GenA}(\text{seed}_a)$
6: $\mathbf{v} \leftarrow \mathbf{a} \cdot \mathbf{y} \bmod^{\pm} q$
7: $\mathbf{c} \leftarrow \mathsf{Enc}(\mathsf{H}([\mathbf{v}]_M, m))$
8: $\mathbf{z} \leftarrow \mathbf{y} + \mathbf{s} \cdot \mathbf{c}$
9: if $\mathbf{z} \notin \mathcal{R}_{q,[B-S]}$ then
10:     counter $\leftarrow$ counter + 1
11:     goto 4
12: end if
13: $\mathbf{w} \leftarrow \mathbf{v} - \mathbf{e} \cdot \mathbf{c} \bmod^{\pm} q$
14: if $||[\mathbf{w}]_L||_\infty \geq 2^{d-1} - E$
15:  or $||\mathbf{w}||_\infty \geq \lfloor q/2 \rfloor - E$ then
16:     counter $\leftarrow$ counter + 1
17:     goto 4
18: end if
19: return $(\mathbf{z}, \mathbf{c})$
```

modulus (but not necessarily straightforward to do in constant time), modular arithmetic in masked form is very inefficient and it is often one of the bottlenecks in terms of running time. In [3], a gadget SecAddModp is defined to add two integers in boolean masked form modulo p. The idea is to naively perform the addition over the integers and to subtract p if the value is larger than p. While this works completely fine, the computational overhead is large in practice and avoiding those reductions would drastically enhance execution time. The ideal case is to work over $\mathbb{Z}_{2^n}$. In this case, almost no reductions are needed throughout the execution of the algorithm and, when needed, can be simply performed by applying a mask on boolean shares. The reason why working with a power of two modulus is not the standard way to instanciate lattice-based cryptography is that it removes the possibility to use the number theoretic transform (NTT) to perform efficient polynomial multiplication in $\mathcal{O}(n \log n)$. Instead, multiplication of polynomial has to be computed using the Karatsuba/Toom-Cook algorithm which is slower for parameters used in state-of-the-art algorithms. Nevertheless, in our case, not having to use the heavy SecAddModp gadget largely overshadows the penalty of switching from NTT to Karatsuba. Since modulus for both parameter sets were already close to a power of two, we rounded to the closest one, i.e. 2^{22} for qTESLA-I and 2^{23} for qTESLA-III. This modification does not change the security of the scheme. Indeed, security-wise, for the heuristic version of the scheme that we study, we need a q such that $q > 4B$[1] and the corresponding decisional LWE instance is still hard. Yet, the form of q does not impact the hardness of the problem as shown in [21] and, since q was already extremely close to a power of two for both parameters sets, the practical bit hardness of the corresponding instance is not sensibly changed.

Removal of the PRF. It is well known that in Schnorr-like signatures, a devastating attack is possible if the adversary gets two different signatures using the same $\mathbf{y}$. Indeed, they can simply compute the secret $\mathbf{s} = \frac{\mathbf{z}-\mathbf{z}'}{\mathbf{c}-\mathbf{c}'}$. While such a situation is very unlikely due to the large size of $\mathbf{y}$, a technique to create a deterministic version of the signature was introduced in [24]. The idea is to compute $\mathbf{y}$ as $\mathsf{PRF}(secret\_seed, m)$ such that each message will have a different value for $\mathbf{y}$ unless a collision is found in PRF. This modification acts as a protection against very weak entropy sources but is not necessary to the security of the signature and was not present in ancestors of `qTESLA`. Unfortunately, adding this determinism also enabled some side-channel attacks [8,25]. Hence, the authors of `qTESLA` decided to take the middle ground by keeping the deterministic design but also seeding the oracle with a fresh random value r[2].

While those small safety measures certainly make sense if they do not incur a significant performance penalty, we decided to drop it and simply sample $\mathbf{y}$ at random at the beginning of the signing procedure. The reason is twofold. First, keeping deterministic generation of $\mathbf{y}$ implied masking the hash function evaluation itself which is really inefficient if not needed and would unnecessarily complicate the masking scheme. Second, implementing a masking countermeasure is, in general, making the hypothesis that a reasonable source of randomness (or at least not weak to the point of having a nonce reuse on something as large as $\mathbf{y}$) is available to generate shares and thus can be also used for the signature itself.

3.2 Existing Gadgets

First, let us describe gadgets already existing in the literature. Since they are not part of our contribution, we decided to only recall their functionalities without formally describing them.

- SecAnd: Computes the logical and between two values given in boolean masked form, output also in boolean masked form. Order 1 algorithm: [12]. Order n algorithm [3].
- SecAdd: Computes the arithmetic add between two values given in boolean masked form, output also in boolean masked form. Order 1 algorithm: [12]. Order n algorithm [3].
- SecArithBoolModq: Converts a value in arithmetic masked form to a value in boolean masked form. Order 1 algorithm: [16]. Order n: [11]. We slightly

[1] The other condition on q in the parameters table of the submission is to enable the NTT.

[2] Note that the fault attacks is still possible in case of failure of the RNG picking r.

Algorithm 3. Absolute Value - AbsVal

Data: A boolean masking $(x_i)_{0\le i\le N}$ of some integer x and an integer k
Result: A boolean masking $(|x|_i)_{0\le i\le N}$ corresponding to the absolute value of $x \bmod^{\pm} 2^k$

```
1: $(mask_i)_{0\le i\le N} \leftarrow ((x_i)_{0\le i\le N} << (\text{RADIX} - k)) >> (\text{RADIX} - 1))$
2: $(x'_i)_{0\le i\le N} \leftarrow \mathsf{Refresh}((x_i)_{0\le i\le N})$
3: $(x_i)_{0\le i\le N} \leftarrow \mathsf{SecAdd}((x'_i)_{0\le i\le N}, (mask_i)_{0\le i\le N}))$
4: $(|x|_i)_{0\le i\le N} \leftarrow ((x_i)_{0\le i\le N} \oplus (mask_i)_{0\le i\le N}) \wedge (2^k - 1)$
```

modify it to an algorithm denoted GenSecArithBoolModq taking into account non power of two number of shares.

– SecBoolArith: Converts a value in boolean masked form to a value in arithmetic masked form. Order 1 algorithm: [16]. Order n algorithm: [9]. This gadget does not explicitly appear in the following but is used inside DataGen.
– DataGen: Takes as input an integer B and outputs a polynomial $\mathbf{y} \in \mathcal{R}_{q,[B]}$ in arithmetic masked form. Uses the boolean to arithmetic conversion.
– FullXor: Merges shares of a value in boolean masked form and output the unmasked value.
– FullAdd: Merges shares of a value in arithmetic masked form and output the unmasked value.
– Refresh: Refreshes a boolean sharing using fresh randomness [19]. We use its N-SNI version, sometimes denoted FullRefresh ([10] Algorithm 4), which is made of a succession of $N+1$ linear refresh operations.

3.3 New Gadgets

To comply with the specifications of qTESLA, our signature scheme includes new components to be masked that were not covered or different than in [3, 23]. In all the following, RADIX refers to the size of the integer datatype used to store the shares.

Absolute Value (Algorithm 3): The three checks during the signing procedure are: $\mathbf{z} \notin \mathcal{R}_{q,[B-S]}$, $||[\mathbf{w}]_L||_\infty \ge 2^{d-1} - E$ and $||\mathbf{w}||_\infty \ge \lfloor q/2 \rfloor - E$. They all involve going through individual coefficients (or their low bits) of a polynomial and checking a bound on their absolute value. In the first version of our work, we were actually making two comparisons on each signed coefficients before realizing that it was actually less intensive to explicitly compute the absolute value and do only one comparison. The gadget takes as input any integer x masked in boolean form and outputs $|x \bmod^{\pm} 2^k|$. Since computers are performing two's complement arithmetic, the absolute value of x can be computed as follows:

1. $m \leftarrow x \gg RADIX - 1$
2. $|x| \leftarrow (x + m) \oplus m$

Algorithm 4. Masked rounding - MaskedRound

Data: An arithmetic masking $(a_i)_{0\le i\le N}$ of some integer a
Result: An integer r corresponding to the modular rounding of a

```
1: $(\text{MINUS\_Q\_HALF}_i)_{0\le i\le N} \leftarrow (-q/2-1, 0, ..., 0)$
2: $(\text{CONST}_i)_{0\le i\le N} \leftarrow (2^{d-1}-1, 0, ..., 0)$
3: $(a'_i)_{0\le i\le N} \leftarrow \mathsf{GenSecArithBoolModq}(a_i)_{0\le i\le N}$
4: $(b_i)_{0\le i\le N} \leftarrow \mathsf{SecAdd}((a'_i)_{0\le i\le N}, (\text{MINUS\_Q\_HALF}_i)_{0\le i\le N})$
5: $b_0 = \neg b_0$
6: $(b_i)_{0\le i\le N} \leftarrow ((b_i)_{0\le i\le N} >> \text{RADIX}-1) << \log_2 q$
7: $(a'_i)_{0\le i\le N} \leftarrow (a'_i)_{0\le i\le N} \oplus (b_i)_{0\le i\le N}$
8: $(a'_i)_{0\le i\le N} \leftarrow \mathsf{SecAdd}((a'_i)_{0\le i\le N}, (\text{CONST}_i)_{0\le i\le N})$
9: $(a'_i)_{0\le i\le N} \leftarrow (a'_i)_{0\le i\le N} >> d$
10: return $t := \mathsf{FullXor}((a'_i)_{0\le i\le N})$
```

As we work on signed integers, one can note that the $\gg$ in the first step is an arithmetic shift and actually writes the sign bit in the whole register. If x is negative then $m=-1$ (all ones in the register) and if x is positive then $m=0$. The gadget AbsVal is using the same technique to compute $|x \bmod^{\pm} 2^k|$. The small difference is that the sign bit is in position k instead of position RADIX. This is why line 1 is moving the sign bit (modulo 2^k) in first position before extending it to the whole register to compute the mask.

Masked Rounding (Algorithm 4): In [2], a compression technique was introduced to reduce the size of the signature. It implies rounding coefficients of a polynomial. Revealing the polynomial before rounding would allow an adversary to get extra information on secret values and thus, this operation has to be done on the masked polynomial. Recall that the operation to compute is $[v]_M = (v \bmod^{\pm} q - [v]_L)/2^d$.

The first step is to compute the centered representative of v, i.e. subtract q from v if $v > q/2$. Taking advantage of our power of two modulus, this operation would be really easy to do if the centered representative was defined as the integer congruent to v in the range $[-q/2, q/2)$ since it would be equivalent to copying the q^{th} bit of v in the most significant part, which can be performed with simple shift operations on shares. Unfortunately, the rounding function of qTESLA works with representatives in $(-q/2, q/2]$. As we wanted compatibility with the original scheme, we decided to stick with their design. Nevertheless, we were still able to exploit our power of two modulus. Indeed, in this context, switching from positive to negative representative modulo q is merely setting all the high bits to one. Hence, we subtract $q/2+1$ from v, extract the sign bit b and copy $\neg b$ to all the high bits of v.

The second step is the computation of $(v-[v]_L)/2^d$. We used a small trick here. Subtracting the centered representative modulo 2^d is actually equivalent to the application of a rounding to the closest multiple of 2^d with ties rounded down. Hence we first computed $v + 2^{d-1} - 1$ and dropped the d least significant bits. This is analogous to computing $\lfloor x \rceil = \lfloor x + 0.499\ldots \rfloor$ to find the closest integer to a real value.

Algorithm 5. Masked well-rounded - MaskedWR

```
Data: Integer $a \in \mathbb{Z}_q$ in arithmetic masked form $(a_i)_{0 \le i \le N}$
Result: A boolean masking $r$ of $(\|a\| \le q/2 - E) \wedge (\|[a]_L\| \le 2^{d-1} - E)$
 1: $(\mathrm{SUP\_Q}_i)_{0 \le i \le N} \leftarrow (-q/2 + E, 0, ..., 0)$
 2: $(\mathrm{SUP\_D}_i)_{0 \le i \le N} \leftarrow (-2^{d-1} + E, 0, ..., 0)$
 3: $(a'_i)_{0 \le i \le N} \leftarrow \mathsf{GenSecArithBoolModq}(a_i)_{0 \le i \le N}$
 4: $(x_i)_{0 \le i \le N} \leftarrow \mathsf{AbsVal}((a'_i)_{0 \le i \le N}, \log_2 q)$
 5: $(x_i)_{0 \le i \le N} \leftarrow \mathsf{SecAdd}((x_i)_{0 \le i \le N}, (\mathrm{SUP\_Q}_i)_{0 \le i \le N}))$
 6: $(b_i)_{0 \le i \le N} \leftarrow (x_i)_{0 \le i \le N} >> (\mathrm{RADIX} - 1)$
 7: $(a'_i)_{0 \le i \le N} \leftarrow \mathsf{Refresh}((a'_i)_{0 \le i \le N})$
 8: $(a'_i)_{0 \le i \le N} \leftarrow (a'_i)_{0 \le i \le N} \wedge 2^d - 1$
 9: $(y_i)_{0 \le i \le N} \leftarrow \mathsf{AbsVal}((a'_i)_{0 \le i \le N}, d)$
10: $(y_i)_{0 \le i \le N} \leftarrow \mathsf{SecAdd}((y_i)_{0 \le i \le N}, (\mathrm{SUP\_D}_i)_{0 \le i \le N}))$
11: $(b'_i)_{0 \le i \le N} \leftarrow (y_i)_{0 \le i \le N} >> (\mathrm{RADIX} - 1)$
12: $(b_i)_{0 \le i \le N} \leftarrow \mathsf{SecAnd}((b_i)_{0 \le i \le N}, (b'_i)_{0 \le i \le N})$
13: return $r \coloneqq \mathsf{FullXor}((b_i)_{0 \le i \le N})$
```

Algorithm 6. Rejection Sampling - MaskedRS

```
Data: A value $a$ to check, in arithmetic masked form $(a_i)_{0 \le i \le N}$
Result: 1 if $|a| \le B - S$ else 0
 1: $(\mathrm{SUP}_i)_{0 \le i \le N} \leftarrow (-B + S - 1, 0, ..., 0)$
 2: $(a'_i)_{0 \le i \le N} \leftarrow \mathsf{GenSecArithBoolModq}((a_i)_{0 \le i \le N})$
 3: $(x_i)_{0 \le i \le N} \leftarrow \mathsf{AbsVal}((a'_i)_{0 \le i \le N}, \log_2 q)$
 4: $(x_i)_{0 \le i \le N} \leftarrow \mathsf{SecAdd}((x_i)_{0 \le i \le N}, (\mathrm{SUP}_i)_{0 \le i \le N})$
 5: $(b_i)_{0 \le i \le N} \leftarrow ((x_i)_{0 \le i \le N} >> \mathrm{RADIX} - 1)$
 6: return $rs \coloneqq \mathsf{FullXor}((b_i)_{0 \le i \le N})$
```

Masked Well-Rounded (Algorithm 5): Unlike GLP, the signature scheme can fail to verify and may have to be restarted even if the rejection sampling test has been successful. This results from the fact that the signature acts as a proof of knowledge only on the **s** part of the secret key and not on the error **e**. Nonetheless, thanks to rounding, the verifier will be able to feed correct input to the hash function if the commitment is so called 'well-rounded'. Since not well-rounded signatures would leak information on the secret key, this verification has to be performed in masked form.

The MaskedWR gadget has to perform the two checks $||[\mathbf{w}]_L||_\infty < 2^{d-1} - E$ and $||\mathbf{w}||_\infty < \lfloor q/2 \rfloor - E$. While the cost of this rather simple operation is negligible compared to polynomial multiplication in the unprotected signature, this test is fairly expensive in masked form. Indeed, it requires four comparisons in addition to the extraction of the low bits of $\mathbf{w}$.

After trying the four comparisons method, we realized that the best strategy was actually to compute both absolute values with the AbsVal gadget. While comparisons only require one SecAdd and one shift, which is less than AbsVal, the cost of all SecAnd operations between the results of those comparisons makes our approach of computing the absolute value slightly better.

Rejection Sampling (Algorithm 6): The rejection sampling procedure consists in ensuring that the absolute value of all coefficients of a polynomial $\mathbf{z}$ are smaller than a bound B. In [3], a gadget verifying that the centered representative of a masked integer is greater than $-B$ was applied to both $\mathbf{z}$ and $-\mathbf{z}$. In [23], a less computationally intensive approach was taken: their rejection sampling gadget takes as input an arithmetic masking of a coefficient $a \in \mathbb{Z}_q$ identified by its canonical representative and check directly that either $a - B$ is negative or $a - q + B$ is positive. This can be easily done using precomputed constants $(-B-1, 0, ..., 0)$ and $(-q+B, 0, ..., 0)$. Our approach is similar but we use instead the same technique as in the MaskedWR algorithm, that is to first compute the absolute value of a and perform the masked test $||a|| \leq B$. This saves the need for a masked operation to aggregate both tests.

Algorithm 7. Masked signature

```
Data: message $m$, secret key $sk = ((\mathbf{s}_i)_{0\leq i\leq N}, (\mathbf{e}_i)_{0\leq i\leq N})$, seed $sd$
Result: Signature $(\mathbf{z}_{unmasked}, \mathbf{c})$
 1: Let $t$ be a byte array of size $n$
 2: $\mathbf{a} \leftarrow \mathsf{GenA}(sd)$
 3: $(\mathbf{y}_i)_{0\leq i\leq N} \leftarrow \mathsf{DataGen}(B)$
 4: for $i = 0, \ldots, N$ do
 5:     $\mathbf{v}_i \leftarrow \mathbf{a} \cdot \mathbf{y}_i$
 6: end for
 7: $\mathbf{u} \leftarrow \mathsf{FullRound}((\mathbf{v}_i)_{0\leq i\leq N})$
 8: $\mathbf{c} \leftarrow \mathsf{Encode}(\mathsf{H}(\mathbf{u}, m))$
 9: for $i = 0, \ldots, N$ do
10:     $\mathbf{z}_i \leftarrow \mathbf{y}_i + \mathbf{s}_i \cdot \mathbf{c}$
11: end for
12: if $rs := \mathsf{FullRS}((\mathbf{z}_i)_{0\leq i\leq N}) = 0$ then
13:     goto 3
14: end if
15: for $i = 0, \ldots, N$ do
16:     $\mathbf{w}_i \leftarrow \mathbf{v}_i - \mathbf{e}_i \cdot \mathbf{c}$
17: end for
18: if $r := \mathsf{FullWR}((\mathbf{w}_i)_{0\leq i\leq N}) = 0$ then
19:     goto 3
20: end if
21: $\mathbf{z}_{unmasked} \leftarrow \mathsf{FullAdd}((\mathbf{z}_i)_{0\leq i\leq N})$
22: return $(\mathbf{z}_{unmasked}, \mathbf{c})$
```

3.4 Masked Scheme

In all signature schemes, two algorithms can leak the secret key through side channels: the key generation algorithm and the signing algorithm.

Masked Sign: The masked signature can be found in Algorithm 7. It uses the gadgets described in Sect. 3.3: the gadgets FullRS, FullWR and FullRound denote the extension of MaskedRS, MaskedWR and MaskedRound to all coefficients $j \in [0, n-1]$ of their input polynomial. Beside the removal of the PRF for $\mathbf{y}$, its structure follows closely the unmasked version of the signature.

Masked Key Generation: As the number of signature queries per private key can be high (up to 2^{64} as required by the NIST competition), whereas the key generation algorithm is typically only executed once per private key, the vulnerability of the key generation to side channel attacks is therefore less critical. We nevertheless masked the key generation algorithm using a CDT sampling. The detailed gadgets and proofs can be found in the full version of our paper [15]. The final algorithm is pretty inefficient because many comparisons are needed.

4 Proof of Masking

We first list in Table 2 all the known gadgets and new gadgets introduced together with their security properties. The techniques for proving the security properties are similar to the proof of Theorem 6. They can be found in the full version of our paper [15].

Table 2. Security properties of the known and new gadgets.

Existing gadgets			New gadgets (proofs in [15])	
Name	Property	Reference	Name	Property
SecAnd	N-NI	[3,12]	GenSecArithBoolModq	N-NI
SecAdd	N-NI	[3,12]	AbsVal	N-NI
SecArithBoolModq	N-SNI	[11,16]	MaskedRound	N-NIo
SecBoolArith	N-NI	[11,16]	FullRound	N-NIo
FullXor	N-NIo	[3]	MaskedWR	N-NIo
FullAdd	N-NIo	[3]	FullWR	N-NIo
DataGen	N-NIo	[3]	MaskedRS	N-NIo
MultAdd	N-NI	[3], denoted H^1	FullRS	N-NIo
Refresh	N-SNI	[19]		

4.1 Main Masking Theorem

In the following, we introduce a theorem that proves the N-NIo property of our masked signature algorithm. For simplicity and without losing generality, the theorem only considers one iteration for the signature: the signing algorithm outputs $\perp$ if one of the tests in Steps 13 or 19 in Algorithm 7 has failed. We also assume the security properties of Table 2. We denote by $\left(r^{(j)}\right)_{0 \leq j < n}$, $\left(rs^{(j)}\right)_{0 \leq j < n}$ and $\left(u^{(j)}\right)_{0 \leq j < n}$ the outputs of FullRS, FullWR and FullRound (the values for each coefficient $j \in [0, n-1]$).

Theorem 6. *Each iteration of the masked signature in Algorithm 7 is N-*`NIo` *secure with public outputs*[3]

$$\left\{ \left(r^{(j)}\right)_{0\leq j<n}, \left(rs^{(j)}\right)_{0\leq j<n}, \left(u^{(j)}\right)_{0\leq j<n} \right\}$$

(and the signature if returned).

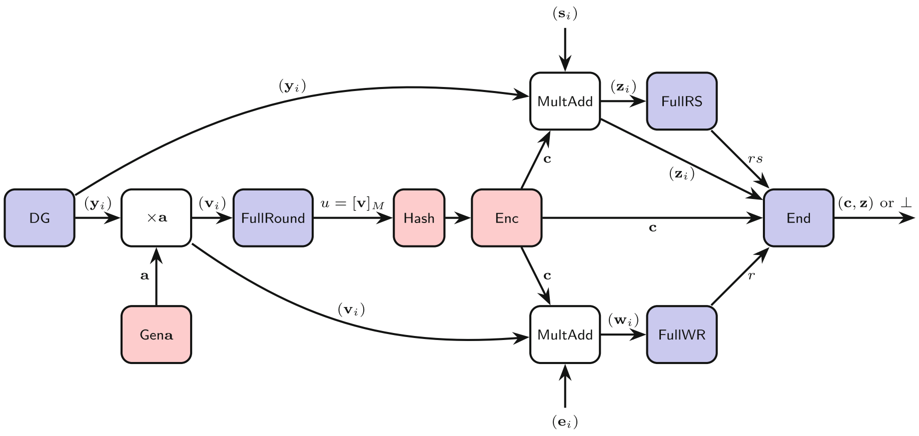

Fig. 1. Masked signature structure (The white (resp. blue, red) gadgets are proved N-`NI` (resp. N-`NIo`, unmasked)). The non sensitive element sd is ommited for clarity. (Color figure online)

Proof. The overall gadget decomposition of the signature is in Fig. 1.

Gadgets. The gadget $\times\mathbf{a}$ multiplies each share of the polynomial $\mathbf{y}$ by the public value $\mathbf{a}$. By linearity, it is N-`NI`. The gadget FullRound denotes the extension of the MaskedRound to all coefficients of $\mathbf{v}$ and is N-`NIo`. The gadget MultAdd takes $(\mathbf{y}_i)_{0\leq i\leq N}$, $(\mathbf{s}_i)_{0\leq i\leq N}$ and $\mathbf{c}$ (resp. $(\mathbf{v}_i)_{0\leq i\leq N}$, $(\mathbf{e}_i)_{0\leq i\leq N}$ and $\mathbf{c}$) and computes $(\mathbf{z}_i)_{0\leq i\leq N} = (\mathbf{y}_i)_{0\leq i\leq N} - \mathbf{c}\cdot(\mathbf{s}_i)_{0\leq i\leq N}$ (resp. $(\mathbf{w}_i)_{0\leq i\leq N} = (\mathbf{v}_i)_{0\leq i\leq N} - \mathbf{c}(\mathbf{e}_i)_{0\leq i\leq N}$). The gadget End simply outputs $(\mathsf{FullAdd}((\mathbf{z}_i)_{0\leq i\leq N}), \mathbf{c})$ if rs and r are true; and $\perp$ otherwise. By the N-`NIo` security of FullAdd, this gadget is also N-`NIo` secure.

Thus, all the subgadgets involved are either N-`NI` secure, N-`SNI` secure, N-`NIo` secure or they do not manipulate sensitive data (see Table 2 for the recap. We prove that the final composition of all gadgets is N-`NIo`. We assume that an attacker has access to $\delta \leq N$ observations. Our goal is to prove that all these δ observations can be perfectly simulated with at most δ shares of $(\mathbf{s}_i)_{0\leq i\leq N}$ and $(\mathbf{e}_i)_{0\leq i\leq N}$ and the knowledge of the outputs.
In the following, we consider the following distribution of the attacker's δ observations:

[3] Here too, the number of iterations of the gadget DG is ommited as a public output.

- δ_1 observed during the computations of DG that produces shares of $(\mathbf{y}_i)_{0 \le i \le N}$,
- δ_2 observed during the computations of the gadget $\times\mathbf{a}$ that produces the shares of $(\mathbf{v}_i)_{0 \le i \le N}$,
- δ_3 observed during the computations of FullRound,
- δ_4 observed during the computations of the upper MultAdd gadget that produces $(\mathbf{z}_i)_{0 \le i \le N}$,
- δ_5 observed during the computations of the lower MultAdd gadget that produces $(\mathbf{w}_i)_{0 \le i \le N}$,
- δ_6 observed during the FullRS,
- δ_7 observed during the FullWR,
- δ_8 observed during the End.

Some observations may be done on the unmasked gadgets (GenA, Hash and Enc) but their amount will not matter during the proof. Finally, we have $\sum_{i=1}^{8} \delta_i \le \delta$.

We build the proof from right to left. The gadgets End, FullRS, FullRound and FullWR are N-NIo secure with the output $(\mathbf{z}, \mathbf{c})$ or $\perp$ (resp. $\left(rs^{(j)}\right)_{0 \le j < n}$, $\left(u^{(j)}\right)_{0 \le j < n}$, $\left(r^{(j)}\right)_{0 \le j < n}$). As a consequence, all the observations from their call can be perfectly simulated with at most δ_8 (resp. δ_6, δ_7) shares of $\mathbf{z}$ (resp. $\mathbf{z}$, $\mathbf{w}$). For the upper MultAdd gadget, there are at most $\delta_8 + \delta_6$ observations on the outputs and δ_4 local observations. The total is still lower than δ and thus they can be simulated with at most $\delta_4 + \delta_6 + \delta_8 \le \delta$ shares of $\mathbf{y}$ and $\mathbf{s}$.

Concerning the lower MultAdd gadget, there are at most δ_7 observations on $\mathbf{w}$ and δ_5 made locally. Thus they can be simulated with at most $\delta_5 + \delta_7 \le \delta$ shares of $\mathbf{v}$ and $\mathbf{e}$.

The gadget FullRound is N-NIo so all the observations from its call can be simulated with at most δ_3 shares of $\mathbf{v}$. Thus, there are $\delta_3 + \delta_5 + \delta_7$ observations on the output of gadget $\times\mathbf{a}$. And then, they can be simulated with at most $\delta_3 + \delta_5 + \delta_7 + \delta_2$ shares of $\mathbf{y}$. Summing up all the observations of $\mathbf{y}$ gives $(\delta_3 + \delta_5 + \delta_7 + \delta_2) + (\delta_4 + \delta_6 + \delta_8) \le \delta$. This allows to conclude the proof by applying the N-NIo security of DG. All the observations on the algorithm can be perfectly simulated with at most $\delta_4 + \delta_6 + \delta_8 \le \delta$ shares of $\mathbf{s}$, $\delta_5 + \delta_7 \le \delta$ shares of $\mathbf{e}$ and the knowledge of the public outputs. □

4.2 EUF-CMA Security in the *N*-probing Model

We recall the EUF-CMA security in the N-probing model. For the complete game description, we refer to [3].

Definition 7. *A signature scheme is EUF-CMA-secure in the N-probing model if any PPT adversary has a negligible probability to forge a signature after a polynomial number of queries to a leaky signature oracle. By leaky signature oracle, we mean that the signature oracle will (1) update the shares of the secret key with a refresh algorithm (2) output a signature together with the leakage of the signature computation.*

Definition 8. *We denote by (r, rs, u)-qTESLA a variant of qTESLA where all the values*

$$\left\{ \left(r^{(j)}\right)_{0\leq j<n}, \left(rs^{(j)}\right)_{0\leq j<n}, \left(u^{(j)}\right)_{0\leq j<n} \right\}$$

are outputted for each iteration during the signing algorithm.

Theorem 6 allows to reduce the EUF-CMA security in the N-probing model of our masked `qTESLA` signature at order N to the EUF-CMA security of (r, rs, u)-`qTESLA`. The security of (r, rs, u)-`qTESLA` is actually not fully supported by the security proof of `qTESLA` because the adversary is not supposed to see these values for the failed attempts of signing. However, based on the work of [3], we can prove that, under some computational assumptions, outputting $\left(u^{(j)}\right)_{0\leq j<n}$for each iteration does not affect the security. We redirect the reader to [3] for further discussions on this issue. The values $\left\{ \left(r^{(j)}\right)_{0\leq j<n}, \left(rs^{(j)}\right)_{0\leq j<n} \right\}$ correspond to the conditions of rejection, and more precisely, the positions of the coefficients of the polynomials that do not pass the rejections. Such a knowledge do not impact the security of the scheme because the rejection probability does not depend on the position of the coefficients (Table 3).

5 Practical Aspects

Our masking scheme has been implemented inside the reference code of `qTESLA` available on the repository of their project [26]. We performed benchmarks for the two parameters sets `qTESLA-I` and `qTESLA-III` on a desktop computer with and without the random number generator activated (in gadgets). The reason why we decided to switch off the RNG[4] is to show how masking schemes of this magnitude are sensitive to the speed at which the device is capable of retrieving randomness. We also tested the smaller parameter set at order 1 on a Cortex-M4 microcontroller to see how it performs on a device more realistically vulnerable to side-channel attacks. We speculate that the scaling difference between the microcontroller and the computer is due to the fact that architectural differences matter less for the masking code than for the base signature code.

Our tests with the randomness enabled were performed using xoshiro128** [6], a really fast PRNG that has been recently used to speed-up public parameters generation in a lattice-based cryptosystem [7]. One looking for real life application of our technique and believing that masking needs strong randomness would maybe want to use a cryptographically secure PRNG instead. Another option could be to expand a seed with the already available `cSHAKE` function but as we will see in the sequel, it might be pretty expensive as the number of random bytes required grows very fast with the number of shares.

The results for all individual gadgets, both parameters sets as well as number of calls to the random number generator lead to interesting considerations. We

[4] To switch the RNG off, we just set the `rand_uint32()` function to return 0.

refer to the full version of our paper [15] for more details. Our general conclusion of all these tests is that beside our much needed design change, the performances are largely dictated by the randomness generation speed and that the bottleneck gadget is the arithmetic to boolean conversion.

Table 3. Median speed of masked signature in clock cycles over 10000 executions for `qTESLA-I` on Intel Core i7-6700HQ running at 2.60 GHz

Masking order	Unmasked	Order 1	Order 2	Order 3	Order 4	Order 5
`qTESLA-I` (RNG off)	645 673	2 394 085	7 000 117	9 219 826	16 577 823	24 375 359
`qTESLA-I` (RNG on)	671 169	2 504 204	13 878 830	24 582 943	39 967 191	59 551 027
`qTESLA-I` (RNG on) Scaling	1	×4	×21	×37	×60	×89

Table 4. Median speed of masked signature in clock cycles over 1000 executions for `qTESLA-I` on cortex-M4 microcontroller

Masking order	Unmasked	Order 1
`qTESLA-I` CortexM4	11 304 025	23 519 583

As noted in [23], the power of two modulus allows to get a reasonable penalty factor for low masking orders. Without such a modification, the scheme would have been way slower. Besides, our implementation seems to outperform the masked implementation of Dilithium as given in [23]. The timing of our order 1 masking for `qTESLA-I` is around 1.3 ms, and our order 2 is around 7.1 ms. This result comes with no surprise because the unmasked version of `qTESLA` already outperformed Dilithium. However, we do not know if our optimizations on the gadgets could lead to a better performance for a masked Dilithium (Table 4).

Acknowledgements. We thank Sonia Belaïd for interesting insights about the masking proofs. We acknowledge the support of the French Programme d'Investissement d'Avenir under national project RISQ P14158. This work is also partially supported by the European Union's H2020 Programme under PROMETHEUS project (grant 780701). This research has been partially funded by ANRT under the programs CIFRE N 2016/1583.

References

1. Alkim, E., et al.: The lattice-based digital signature scheme qTESLA. Cryptology ePrint archive, report 2019/085 (2019). https://eprint.iacr.org/2019/085
2. 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
3. Barthe, G., et al.: Masking the GLP lattice-based signature scheme at any order. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 354–384. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_12

4. Barthe, G., et al.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., et al. (ed.) ACM CCS 2016, pp. 116–129. ACM Press, October 2016
5. Bindel, N., et al.: qTESLA. Technical report, National Institute of Standards and Technology (2017). https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions
6. Blackman, D., Vigna, S.: Scrambled linear pseudorandom number generators. In: CoRR abs/1805.01407 (2018). arXiv:1805.01407
7. Bos, J.W., et al.: Fly, you fool! Faster Frodo for the ARM Cortex-M4. Cryptology ePrint archive, report 2018/1116 (2018). https://eprint.iacr.org/2018/1116
8. Bruinderink, L.G., Pessl, P.: Differential fault attacks on deterministic lattice signatures. IACR Trans. Cryptograph. Hardw. Embedded Syst. **2018**(3), 21–43 (2018). https://tches.iacr.org/index.php/TCHES/article/view/7267
9. Coron, J.-S.: High-order conversion from boolean to arithmetic masking. Cryptology ePrint archive, report 2017/252 (2017). http://eprint.iacr.org/2017/252
10. Coron, J.-S.: Higher order masking of look-up tables. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 441–458. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_25
11. Coron, J.-S., Großschädl, J., Vadnala, P.K.: Secure conversion between boolean and arithmetic masking of any order. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 188–205. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_11
12. Coron, J.-S., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from arithmetic to boolean masking with logarithmic complexity. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 130–149. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48116-5_7
13. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_24
14. Ducas, L., et al.: CRYSTALS-Dilithium: a lattice-based digital signature scheme. IACR Trans. Ctyptograph. Hardw. Embedded Syst. **2018**(1), 238–268 (2018). https://tches.iacr.org/index.php/TCHES/article/view/839
15. Gérard, F., Rossi, M.: An efficient and provable masked implementation of qTESLA. Cryptology ePrint archive, report 2019/606 (2019). https://eprint.iacr.org/2019/606
16. Goubin, L.: A sound method for switching between boolean and arithmetic masking. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 3–15. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_2
17. Goudarzi, D., et al.: Unifying leakage models on a Rényi Day. Cryptology ePrint archive, report 2019/138 (2019). https://eprint.iacr.org/2019/138
18. 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
19. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
20. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9

21. Langlois, A., Stehlé, D.: Hardness of decision (R)LWE for any modulus. Cryptology ePrint archive, report 2012/091 (2012). http://eprint.iacr.org/2012/091
22. 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
23. Migliore, V., et al.: Masking Dilithium: efficient implementation and side-channel evaluation. Cryptology ePrint archive, report 2019/394 (2019). https://eprint.iacr.org/2019/394
24. M'Raïhi, D., Naccache, D., Pointcheval, D., Vaudenay, S.: Computational alternatives to random number generators. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 72–80. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48892-8_6
25. Poddebniak, D., et al.: Attacking deterministic signature schemes using fault attacks. Cryptology ePrint archive, report 2017/1014 (2017). http://eprint.iacr.org/2017/1014
26. qTESLA team. https://qtesla.org/