Fast First-Order Masked NTTRU

Daniel Heinz[1,2](✉) and Gabi Dreo Rodosek[1]

[1] Research Institute CODE, Universität der Bundeswehr München, 85577 Neubiberg, Germany
{Daniel.Heinz,Gabi.Dreo}@unibw.de

[2] Infineon Technologies AG, Am Campeon 1-15, 85579 Neubiberg, Germany

Abstract. Even though Kyber is the lattice-based KEM selected for standardization by NIST, NTRU and its variants are still of great relevance to several practical applications. This is why we want to shed light on the side-channel resilience of NTTRU, which is a very fast variant of NTRU designed to use the Number-Theoretic Transform. It outperforms NTRU-HRSS significantly in an unprotected context, which raises the question of whether this performance advantage holds when side-channel attacks have to be considered.

To answer that, we present the first masked implementation of NTTRU optimized for first-order. To achieve a fast performance, we present a table-based approach for the masked sampler and the modulus conversion, similar to the A2B conversion proposed by Debraize in 2012. The modulus conversion is also applicable to other NTRU variants. Due to its usage in NTTRU, we present a fully first-order masked SHA512 implementation based on A2B and B2A conversions. We come to the conclusion that performance is heavily impacted by the SHA2 family in masked implementations and strongly encourage the employment of SHA3 in these cases. This result is also of relevance for the 90s/AES variants of the NIST standardization candidates Kyber and Dilithium.

We achieve a performance of the NTTRU-SHA3 of around 3.1 million cycles on the ARM Cortex M4. Finally, we show that our proposed methods provide side-channel security in practice by employing the well established TVLA methodology.

Keywords: Lattice-based cryptography · NTRU · NTTRU · DPA · Countermeasure · Masking · ARM Cortex M4

1 Introduction

In recent years, post-quantum cryptography has seen increased research attention as classic public-key cryptographic solutions could be broken by advanced quantum computers using Shor's algorithm [1]. During the NIST standardization process [2], several quantum-resistant schemes have been proposed to make secured key exchanges possible even when large-scale quantum computers become available. The schemes are based on different mathematical problems. Among the lattice-based candidates, Kyber [3], Saber [4], and NTRU [5,6] were

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
E. B. Kavun and M. Pehl (Eds.): COSADE 2023, LNCS 13979, pp. 127–148, 2023.
https://doi.org/10.1007/978-3-031-29497-6_7

part of the final round of the NIST standardization process. For instance, NTRU is a cryptosystem that makes use of structured lattices to exchange keys in a 'quantum secure' way. An advantage of schemes based on structured lattices are their comparatively small key sizes. Additionally, encryption and decryption can often be performed faster than in traditional RSA or EC-based schemes [7].

Despite ciphertext, public key, and secret key being almost of the same size as in Kyber, in general, NTRU-based schemes perform better in terms of speed during encryption and decryption. Additionally, NTRU-based ciphertexts only consist of one element which is of advantage when a zero-knowledge proof of the honest generation of the ciphertext is needed [7]. Even though NTRU and its variants were not standardized, NTRU is still a very important cryptosystem. An important example is the OpenSSH [8] program that includes an implementation of NTRUprime since April 2022. Additionally, Google has recently announced to use NTRU-HRSS for their internal encryption-in-transit protocol ALTS [9]. This shows that alternatives to the NIST competition are of great relevance for practical applications.

In lattice-based algorithms, the speed of polynomial multiplication is one of the bottlenecks. Depending on the modulus of the underlying algebraic ring, various schemes tackle this issue differently. The key encapsulation mechanism (KEM) Saber [4], for instance, makes use of a combination of Toom-Cook, schoolbook, and Karatsuba multiplication whereas Kyber's parameter set [3] allows fast multiplication using the Number-Theoretic Transform (NTT).

The NTT approach for polynomial multiplication is especially fast in dimensions that are a power of two. Kyber solves this issue by using a matrix/vector structure with multiple polynomials of dimension 2^8. To obtain a security level of 128 bits, the dimension of the ring is, according to current security analysis, required to be around 700 to 800 [10]. NTRU-based schemes do not use a matrix/vector structure and, thus, secret key, public key and ciphertext only consist of one polynomial. As there exists no power of two in the 128 bit security range between 700 and 800, the most efficient NTT technique is not applicable for NTRU with this security parameter. This might be the reason why an NTRU-based scheme that makes use of NTTs was not part of the NIST standardization process. The authors of [7] propose a specific parameter set to use the NTT approach in the NTRU scheme to gain additional performance gains and call their scheme NTTRU. The authors consider it at least as secure as the corresponding NTRU-HRSS variant that was part of the third round of the NIST competition. Therefore, it is worth taking a closer look at the so-called NTTRU.

Due to its good performance, NTTRU is a potential candidate to be used on embedded devices. Naturally, embedded devices are exposed to a large number of physical attacks such as fault attacks or side-channel attacks as first demonstrated by Kocher et al. [11]. Thus, it is crucial to secure cryptographic schemes against these threats. Correlation between power consumption or electromagnetic radiation and secret intermediate values can be counteracted by the so-called masking countermeasure where each sensitive variable is split into

several randomized shares. Each share is then processed separately from a secret intermediate value. Some of the PQC lattice-based candidates have already seen increased research attention in this regard. For Kyber [12–14] and Saber [15,16] first- and higher-order masked implementations exist. Recently, a masked higher-order implementation of NTRU-HRSS has been proposed [17]. However, no first-order optimized version of NTRU has been published. We aim at closing the gap with our work.

Contributions. In this work, we present the first first-order masked implementation of NTTRU. We employ the first-order masking technique in the complete scheme. Hereby, we propose a new table-based method for a first-order secured modulus conversion. We emphasize that this technique is potentially applicable to all other NTRU variants. Subsequently, we present a first-order masked implementation of the SHA2-512 algorithm based on fast table-based conversions, because it is an important building block of NTTRU, and a new table-based sampling technique. We provide detailed performance numbers on the different components and conclude that the SHA2 family is significantly more expensive to protect with masking compared to the SHA3 family. This result is also of great interest when taking a look at the 90s/AES versions of the NIST selected algorithms Kyber and Dilithium. We verify the results using the state-of-the-art TVLA methodology for our newly proposed components. Finally, we propose a slightly adapted version of NTTRU that achieves a cycle count for decapsulation of around 3.1 million cycles on the ARM Cortex-M4 even without assembler optimized code for the ARM Cortex-M4. This is about a factor of ten faster than the first-order cycle count for NTRU-HRSS on the ARM Cortex M3 [17].

2 Preliminaries

In this section, we present the preliminaries of masking the NTTRU scheme.

2.1 Notation

For any prime q and a polynomial f, we denote R_q as the polynomial ring $\mathbb{Z}_q[X]/(f)$ where $\mathbb{Z}_q$ denotes the quotient ring $\mathbb{Z}/q\mathbb{Z}$. Polynomials in R_q are denoted as lowercase letters. The NTT transform of a polynomial a is represented as $\hat{a}$ and the base multiplication in the NTT domain (not necessarily coefficientwise) is denoted as $\circ$. The i-th coefficient of a polynomial p is denoted as $p[i]$. Given a distribution χ, we use $x \leftarrow \chi$ to mean x is sampled according to the distribution χ. For a polynomial, this is adjusted such that $p \leftarrow \chi^n$ where $n-1$ is the degree of the polynomial. We denote the modular reduction of x to the domain $[-(q-1)/2, (q-1)/2]$ as $x \bmod {}^{\pm}q$.

We denote the j-th share of a shared variable $x^{(\cdot)}$ as $x^{(j)}$, whereas the unshared variable itself is denoted as x. Concatenation is represented as $||$.

2.2 The Number-Theoretic Transform

A common solution to make fast arithmetic in lattice-based solutions possible is the usage of the Number-Theoretic Transform (NTT). It is based on the Chinese Remainder Theorem For a prime q and a polynomial f that factors into the product $f = gh$ with g and h relatively prime, the isomorphism

$$\mathbb{Z}_q[X]/(f) \cong \mathbb{Z}_q[X]/(g) \times \mathbb{Z}_q[X]/(h) \tag{1}$$

is valid. Apparently, it is possible to compute a linear operation in the two factor rings and invert the result back to the original ring. If the map and inverse map to the smaller factor rings can be computed efficiently, it is possible that this approach is more efficient than the simple computation in the main ring $\mathbb{Z}_q[X]/(f)$.

2.3 NTTRU

In the final round of the NIST standardization process [2] two NTRU-based schemes were present. Both, NTRU [6] and NTRUprime [18] make use of polynomial arithmetic. The discerning feature of NTRUprime is that it deliberately avoids cyclotomic rings. In [7], Lyubashevsky and Seiler propose a specific parameter set to optimize NTRU for NTT-based multiplication. In contrast to both finalists, a decryption error can occur when using this parameter set. However, in [7], it is proven that the resulting IND-CCA2 KEM is still appropriately secure. The authors additionally state that their scheme is at least as secure as NTRU-HRSS as they use the same error distribution while increasing the ring dimension and decreasing the modulus. It is not possible to give a formal security reduction because of the different rings. According to their findings, this results in a major speed-up of the scheme. We give an overview of the underlying OW-CPA secure encryption scheme in Algorithms 1–3.

Algorithm 1: NTTRU.KeyGen

Output: Key Pair (sk, pk)
1. $f' \leftarrow \beta_2^{768}$
2. $f \leftarrow 3f' + 1$
3. $\hat{f} \leftarrow NTT(f)$
4. $g \leftarrow \beta_2^{768}$
5. $\hat{3g} \leftarrow NTT(3g)$
6. if f is not invertible: restart
7. $\hat{h} \leftarrow \hat{3g} \circ \hat{f}^{-1}$
8. **return** $(sk = \hat{f}, pk = \hat{h})$

Algorithm 2: NTTRU.Encrypt

Input: message m, randomness r, public key $\hat{h}$
Output: ciphertext $\hat{c}$
1. $\hat{r} \leftarrow NTT(r)$
2. $\hat{m} \leftarrow NTT(m)$
3. $\hat{v} \leftarrow \hat{r} \circ \hat{h}$
4. **return** $\hat{c} := \hat{v} + \hat{m}$

The FO-Transform. The direct usage of these algorithms results in a scheme that is not resilient against chosen-ciphertext attacks. To counter these attacks the NTTRU scheme introduces a re-encryption step. The decrypted message is

Algorithm 3: NTTRU.Decrypt

Input: ciphertext $\hat{c}$, secret key $\hat{f}$
Output: message m
1 $\hat{m} \leftarrow \hat{c} \circ \hat{f}$
2 **return** $m := INTT(\hat{m}) \bmod {}^{\pm}3$

re-encrypted and the resulting ciphertext is compared with the input ciphertext. The approach was first proposed at Crypto '99 by Fujisaki and Okamoto [19]. The transformed algorithm is shown in Algorithm 4. In contrast to the OW-CPA version, the randomness for (re-)encrypting is not sampled completely at random but derived deterministically from the message to encrypt. This way, any wrongly decrypted message results in different randomness and consequently completely randomizes the re-encrypted ciphertext. The comparison at the end will fail and the wrongly decrypted message will not be the output. The algorithm will return 0. In this context, we write $\mathcal{H}_{D_{\mathcal{R}}}$ to denote a cryptographic hash function that generates elements according to the distribution $D_{\mathcal{R}}$ with an input seed m. The hash $\mathcal{H}_{\mathcal{R}}$ produces elements uniformly at random in $\mathcal{R}$. In the context of NTTRU, $\mathcal{H}_{D_{\mathcal{R}}}$ is initialized as

$$\mathcal{H}_{D_{\mathcal{R}}} = (AES256ctr(SHA512(m), nonce)) \tag{2}$$

where $AES256ctr$ is the AES256 in counter mode with a key derived from the hash SHA512 [20] of m and a nonce. We describe the symmetric algorithms and the sampling algorithm in the next sections.

Algorithm 4: CCA.NTTRU.Decrypt

Input: ciphertext c, secret key f
Output: shared key k
1 $m \leftarrow NTTRU.Decrypt(c, sk)$
2 $seed \leftarrow \mathcal{H}_{D_{\mathcal{R}}}(m)$
3 $r \leftarrow Sampler(seed)$
4 **if** $c \neq NTTRU.Encrypt(m, r, pk)$ **then**
5 | **return** $k \leftarrow 0$
6 **return** $k \leftarrow \mathcal{H}_{\mathcal{K}}(m)$

2.4 Symmetric Primitives

SHA512. The FO-Transform and, hence, the hash function are an essential part of all CCA secured lattice-based schemes. As the input to the hash is the decrypted message, even a small error in the decryption (e.g. a chosen ciphertext input or an effective fault attack) will result in a completely randomized hash

value and, thus, in a shared key $k = 0$. In the NTTRU case, SHA512 [20] is used. In the presence of quantum computers, the preimage security of hashes is halved. The SHA512 algorithm [20] is part of the SHA2 family and operates on 512-bit blocks. The used functions are defined as

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \tag{3}$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{4}$$

$$\Sigma_0(x) = S^{28}(x) \oplus S^{34}(x) \oplus S^{39}(x) \tag{5}$$

$$\Sigma_1(x) = S^{14}(x) \oplus S^{18}(x) \oplus S^{41}(x) \tag{6}$$

$$\sigma_0(x) = S^{1}(x) \oplus S^{8}(x) \oplus R^{7}(x) \tag{7}$$

$$\sigma_1(x) = S^{19}(x) \oplus S^{61}(x) \oplus R^{6}(x) \tag{8}$$

In this definition, $S^n(x)$ denotes a shift to the right of x by n bits and R^n denotes a rotation to the right of x by n bits. In contrast to SHA256, for SHA512 the state variables are of size 64-bit. After one block of the message has been processed, the values resulting from the compression function are added to the state variables and reduced modulo 2^{64}. After processing the last block, the hash is obtained by simple concatenation of the eight state variables. The resulting output has a length of 64 bytes.

Keccak. Another symmetric primitive that is frequently used in lattice-based schemes is called Keccak. In 2015, Keccak won the SHA3 competition and became the successor of the SHA2 family. Similar to SHA2, the SHA3 family consists of several functions with different output lengths. The SHA3 standard is derived from special parametrization of the Keccak function. The state size is fixed to 1600 bits and the number of rounds is fixed to 24. Within the function f, the state vector of 1600 bits is processed in several rounds. Within each round of f, several subfunctions are called:

- θ takes two columns in the three-dimensional arranged state and the target bit as input and xor's the parity of the two columns onto the target bit,
- ρ and π rearrange the positions of the bits within the state,
- χ is the non-linear operation that is using the negation function, the boolean and function, and an xor operation, and
- ι which xor's the state vector with a round constant in each round.

Note, that none of these subfunctions requires an arithmetic operation.

2.5 Sampling Algorithms

In some lattice-based schemes, e.g. Kyber and NTRU, the output of the pseudorandom function (PRF) requires additional processing to follow a binomial distribution but the PRF outputs uniformly distributed bits. The uniformly random bitstream can, however, be used as an input to the centered binomial sampler. To obtain such a distribution in the domain $[-\eta, \eta]$, Kyber uses 2η

independent one-bit variables and starts by adding the first η variables and the next η variables. Then one of the two sums is subtracted from the other one. Thus, the coefficient $c \in [-\eta, \eta]$ is calculated as

$$c = \sum_{i=0}^{\eta-1} b_i - \sum_{i=0}^{\eta-1} b_{i+\eta}. \tag{9}$$

NTTRU requires an additional modular reduction to obtain random coefficients in $[-1, 1]$. The NTTRU reference implementation calculates each coefficient by

$$c = (b_1 + b_2) - (b_3 + b_4) \bmod 3. \tag{10}$$

The sampling operation in NTTRU is realized by a lookup table. Both of the sums can take three values, resulting in nine possible outcomes for the coefficient and the table entries. In NTTRU, the authors additionally simplify the approach by directly using the table-based approach on the four input bits. In practice, the table can be realized by a 32-bit variable that stores all the 2^4 possibilities in $\{0, 1, 2\}$ and is shifted by twice the value of the four input bits. Since the distribution is symmetric around zero, it is even possible to only use a 16-bit variable as a lookup table and directly shift by the number obtained from the four concatenated input bits, resulting in

$$c = (L >> (b_1 \| b_2 \| b_3 \| b_4)) \wedge 0x3 - 1 \tag{11}$$

with $L = 0xA815$.

2.6 Side-Channel Attacks and Protection

In recent years methods like Simple Power Analysis (SPA) [21] and Differential Power Analysis (DPA) [11] have seen increased focus for post-quantum schemes. Several attacks on (protected) lattice-based schemes have been proposed using power or timing side-channels [22–26]. The attacks include side-channel assisted CCA attacks where the information from the re-encryption step of the FO-Transform is used for secret recovery [27,28]. Therefore, it is crucial to protect not only the decryption but also the re-encryption step with appropriate countermeasures.

In practice, the most well-known countermeasure is called masking [29]. Secret variables are split into two or more randomized shares. One can choose between arithmetic masking, where the secret s is split into two shares such that $s = s_1 + s_2 \pmod q$ and Boolean masking resulting in a sharing s_1, s_2 such that $s = s_1 \oplus s_2$. In lattice-based cryptography, both possibilities are frequently used in conjunction. Different parts of the decapsulation work more efficiently on either arithmetic or Boolean masking. Therefore, methods to securely convert from one to the other exist [30,31]. Masked implementations of Saber [15,16], Kyber [12–14] and, recently, NTRU-HRSS [17] were proposed. However, no detailed analysis for first-order protection of NTTRU has been performed.

3 Side-Channel Protection of NTTRU

In this section, we will go through the primitives used in NTTRU and provide a first-order masking scheme for each function. This is visualized in Fig. 1. It shows how the two input shares of the secret s_1 and s_2 as well as the unmasked input ciphertext c and public key pk are processed in the algorithm. The masked functions are presented in chronological order from the input secret s_1 and s_2.

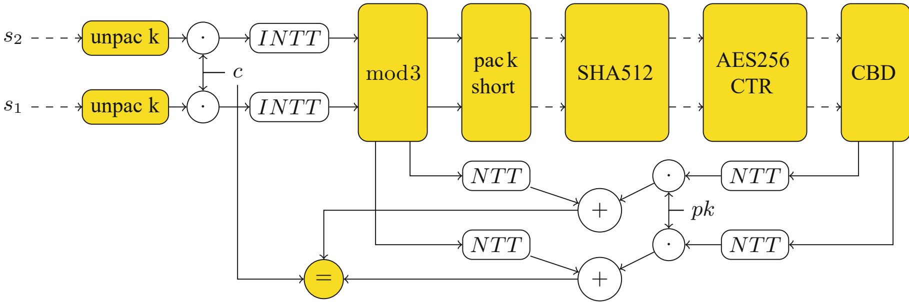


Fig. 1. Masked Decapsulation of NTTRU. Boolean shared data paths in dashed lines. Arithmetically shared data paths in solid lines. Non-linear functions in yellow. (Color figure online)

Masked Unpacking. The first function to encounter that works on secret data is the unpacking function. In our work, we directly store the generated secret key in arithmetic sharing on the device as in most use cases key generation is performed on the same platform. Hence, we do not need a so-called $B2A_q$ conversion. Such a conversion is quite expensive in terms of cycle counts. The approach is possible because the unpacking function does not compress the secret key. Thus, an arithmetic sharing requires the same amount of memory as a "packed" secret key.

3.1 Table-Based Masking of Modulus Conversion

A major challenge in masking NTTRU as well as NTRU is the masking of the modulus conversion. Concretely, it is required to mask the operation

$$(x \bmod {}^{\pm}q) \bmod {}^{\pm}3.$$

The challenge is, that different representatives of $x \bmod q$ lead to different results when reduced modulo 3. In the NTTRU reference implementation [7], the input to the mod 3 function, is an output from the inverse NTT. This means that the coefficients are distributed in $[-(q-1),(q-1)]$ because of the used Barrett reductions.

In the unmasked constant-time implementation, the correct representative of $x \bmod {}^{\pm}q$ is found by first retrieving the most significant bit of x. In case x is negative and, therefore, the most significant bit is 1, x is increased by q. This conditional addition is the most challenging part in the masked implementation. The result is a value in $[0, q-1]$ which is then subtracted by $\frac{q-1}{2}$. The procedure is repeated with the exception of the subtraction of the last constant. With a final subtraction of $\frac{q+1}{2}$ the original value modulo q is restored and the domain of the coefficient is then in $[-\frac{q-1}{2}, \frac{q-1}{2}]$.

We present an approach that incorporates the reduction to the correct representative $\bmod^{\pm}q$ and the reduction modulo 3 in a table-based approach. In our first-order masked approach, we first reduce each share to the domain $[-\frac{q-1}{2}, \frac{q-1}{2}]$ as previously presented, then we compute the A2B conversion of the shared coefficient $a^{(\cdot)}$ as proposed by Debraize [32] and later improved by Van Beirendonck et al. [33] and then extract the most significant bits of both shares. We obtain a boolean sharing $b^{(\cdot)}$ of the most significant bit. We then generate a random input mask bit r_1 and a random output mask r_2 in $[0, q-1]$. Then our lookup table is initialized for $r_1 = 0$:

- The first entry corresponds to the most significant bit being zero. The coefficient a is positive and we require a sharing of zero to be added to a. Consequently, the entry is the inverted output mask r_2.
- The second entry corresponds to the most significant bit being equal to one. The coefficient a is negative and does require the addition of q. Thus, the entry is initialized as $q - r_2$.

Apparently, if $r_1 = 1$ the table entries are initialized the other way around. We present the function in Algorithm 5.

Algorithm 5: Initialization of LUT

Input: Random bit r_1, random output mask $r_2 \in [0, q-1]$
Output: Table $T[2]$
1 $T[0 \oplus r_1] \leftarrow -r_2$
2 $T[1 \oplus r_1] \leftarrow q - r_2$
3 **return** T

After the initialization of the table, both shares are combined carefully with the random bit r_1 by an xor operation. The helper variable with two shares is initialized with $h^{(\cdot)} = (r_2, T[r_1 \oplus b^{(0)} \oplus b^{(1)}])$. Finally, sharewise addition of $a^{(\cdot)} + h^{(\cdot)}$ yields the arithmetically shared value in $[0, q-1]$. We repeat this procedure once after the subtraction of $\frac{q-1}{2}$. Finally, both shares are reduced modulo 3. We show the procedure in Algorithm 6.

Algorithm 6: Masked Conversion to Modulo 3

Input: Shared coefficient $a^{(\cdot)}$ with unmasked coefficients in $[-(q-1), q-1]$
Output: Shared coefficient $a^{(\cdot)} \mod 3$ with unmasked coefficients in $[-1, 1]$

1 `//Conditionally add` q
2 $h^{(\cdot)} \leftarrow A2B(a^{(\cdot)})$
3 $b^{(\cdot)} \leftarrow MSB(h^{(\cdot)})$
4 Sample random bit r_1, random $r_2 \in [0, q-1]$
5 $val \leftarrow r_1 \oplus b^{(0)} \oplus b^{(1)}$
6 $h^{(0)} \leftarrow r2$
7 $h^{(1)} \leftarrow T[val]$
8 $a^{(\cdot)} \leftarrow a^{(\cdot)} + h^{(\cdot)}$
9 `//Always subtract`
10 $a^{(0)} \leftarrow a^{(0)} - (q-1)/2$
11 `//Conditionally add` q
12 $h^{(\cdot)} \leftarrow A2B(a^{(\cdot)})$
13 $b^{(\cdot)} \leftarrow MSB(h^{(\cdot)})$
14 Sample random bit r_1, random $r_2 \in [0, q-1]$
15 $val \leftarrow r_1 \oplus b^{(0)} \oplus b^{(1)}$
16 $h^{(0)} \leftarrow r2$
17 $h^{(1)} \leftarrow T[val]$
18 $a^{(\cdot)} \leftarrow a^{(\cdot)} + h^{(\cdot)}$
19 `//Always subtract`
20 $a^{(0)} \leftarrow a^{(0)} - (q+1)/2$
21 `//Now reduce modulo 3 sharewise`
22 $a^{(\cdot)} \leftarrow a^{(\cdot)} \mod 3$
23 **return** $a^{(\cdot)}$

3.2 Masked Packing

To save memory, each coefficient of the message polynomial, which only requires two bits, is not stored in a full 16-bit variable. Instead, each coefficient is concatenated in an array of 96 bytes which is later used as an input for the symmetric primitives. This is the reason why the correct representative of $x \bmod 3$ is important. In contrast to the arithmetic modulo 3, for an input to the SHA512 $11_2 = -1 \neq 2 = 10_2$. According to the specification of NTTRU, coefficients of the polynomial are in the domain $[-1, 1]$ whereas the concatenated message is obtained by shifting the interval by one to $[0, 2]$. Consequently, we propose to combine both steps efficiently in one table for the first-order masked approach. Instead of only calculating the entries of the table as a Boolean sharing of the arithmetically shared value a, we provide the Boolean sharing for $a + 1$. In contrast to any higher-order compatible A2B conversion, we do not need a costly Boolean adder on the shares. For each coefficient, we refresh the masking with new random values. Concatenation of the Boolean shared values works sharewise.

3.3 Protected SHA512 and AES256-CTR

In this section, we provide details on how to protect the symmetric primitives from DPA attacks.

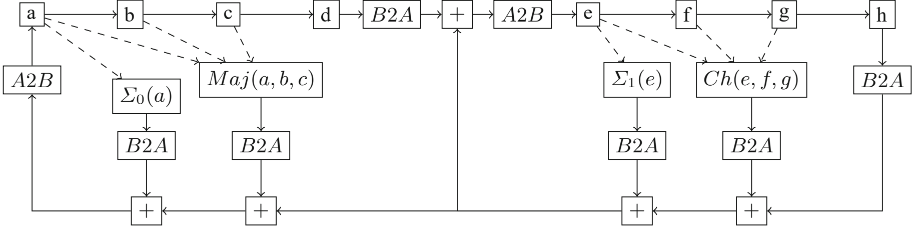


Fig. 2. Masked SHA512 Compression function with conversions in place.

SHA512. In NTTRU, the decrypted message is input to the SHA2-512 hashing function. Due to performance reasons, SHA2 is chosen over SHA3. The drawback of this choice becomes apparent when the masking technique is applied to the hashing algorithm. The SHA2 standard combines arithmetic operations modulo 2^{64} with bitwise Boolean operations. Thus, for masking SHA512, we have two options:

– Usage of A2B conversions: Boolean functions operate on Boolean shares, and arithmetic functions on arithmetic shares. The conversion is performed, if necessary, in between the functions.
– Usage of Boolean Adders: no arithmetic shares are used, and arithmetic additions modulo 2^{64} are performed on boolean shares using specific algorithms.

We evaluated both strategies for the first-order implementation and present the chosen strategy in this section. For the first case, we adapt the compression function to include A2B conversions, as proposed by Debraize [32] and later improved by Van Beirendonck [33], and B2A conversions are realized as presented by Goubin [30]. The performance of this approach (only 7 cycles per B2A conversion) is especially beneficial to the first-order implementation. The resulting flow is shown in Fig. 2. In the latter case, we refer to the control flow of the compression function from Fig. 2 without the conversions. Instead of additions modulo 2^{64}, we use an algorithm based on Goubin's Theorem and in detail analyzed by Coron et al. [34]. Its runtime dependency on the number of bits is rather disadvantageous for SHA512 as it operates on 64 bit variables. For the first-order case, the table-based approach combined with Goubins B2A conversion turns out to be preferable in terms of runtime.

In both cases - using boolean adder or conversions - the only part that remains to be masked is the non-linear AND. This operation cannot be realized sharewise and, thus, is realized as presented in [34].

AES256-CTR. In this work, we additionally adapted an open-source masked implementation of AES, as it is an essential part of the seed generation for the coefficient sampling. For AES128 in counter mode, several masked solutions exist [35–37]. All of these implementations do not mask the key expansion function as the expanded shared key is often assumed to be stored on the chip. In our implementation, this is not possible. The SHA512 hash value of the decrypted message is serving as the key and still has to be expanded. Since the AES was not the primary focus of this work, we adapted an open-source portable `C` implementation that already masks the key expansion for AES128 and uses the bitslicing technique [35]. To make their concept compatible to our approach, we first stored the last 32 bytes of the output of the SHA512 function in a bitsliced manner. We adjusted the key schedule function of the AES128 to match the AES256 specification and added four more rounds to the update function. The key is updated at the end of each round to obtain the next subkey from the previous subkey. As a message, the increasing nonce for each block combined with a zero-padded IV is used. Finally, the output is restored from the bitsliced variables and used as a pseudorandom input to the polynomial sampler of the NTTRU re-encryption. The results are not particularly optimized concerning cycle counts but still give an upper bound of the cycles needed for symmetric seed expansion. We emphasize that there is still a lot of performance to be gained when applying the several (architecture-specific) optimization techniques as presented, e.g., by Schwabe et al. [36].

3.4 Table-Based Masking of Coefficient Sampling

As described in Sect. 2.5, the sampling in NTTRU is slightly different to Kyber due to the additional modular reduction step. The output of the sampler is in the domain $[-1, 1]$ but has to be masked arithmetically mod$q$. In our masked approach, we first compute the table by computing a masked result for all possible 16 unmasked input values. This is shown in Algorithm 7. The second share of the table is a random value $r_{out} \in R_q$ that is equal for all outcomes. To minimize the size of the table, we additionally assume one share of the input to be random but identical r_{in} for all inputs.

Algorithm 7: Initialization of LUT for first order CBD sampling in the domain $[-1, 1]$

Input: Random input mask $r_{in} \in [0, 15]$, Random output mask $r_{out} \in [0, q-1]$
Output: Table $T[16]$

1 $val \leftarrow 0$
2 **while** $val < 16$ **do**
3 $\quad T[val] \leftarrow (0xA815 >> (val \oplus r_{in}) \wedge 0x3) + q - 1 - r_{out} \mod q$
4 $\quad val \leftarrow val + 1$
5 **end**
6 **return** $T[16]$

During the online phase (Algorithm 8), we remask each coefficient to take r_{in} as one Boolean share. The other share is an input to the lookup table. The table gives a randomized output in R_q that, together with the random but fixed value r_{out}, is equivalent to the arithmetic masking of the sampled value obtained from a centered binomial distribution modulo 3. Note that the sampling technique provides an implicit B2A$_q$ conversion. Finally, we remask the output for each coefficient. This approach does obviously not defend against horizontal attacks. Several other countermeasures, especially table-based approaches, face this issue. Yet, they can be used with additional countermeasures, e.g. shuffling or RNR [38,39], in place. This is out of scope of this paper and is an interesting direction for future work.

Algorithm 8: First order sampling in the domain $[-1, 1]$ based on LUT

```
   Input: Shared buffer buf^(·)[N/2]
   Output: Shared polynomial a^(·)[N] with N coefficients
 1 generate randomness r ∈ [0, q − 1], s ∈ [0, 15]
 2 initialize sampling table with r_in = s, r_out = r
 3 i ← 0
 4 while i < N/2 do
 5 |   h ← buf^(1)[i] ⊕ (s << 4 ∨ s)
 6 |   h ← h ⊕ buf^(0)[i]
 7 |   generate randomness rnd ∈ [0, q − 1]
 8 |   a^(0)[2i] ← rnd
 9 |   a^(1)[2i] ← (T[h ∧ 0xF] − rnd + r) mod q
10 |   generate randomness rnd ∈ [0, q − 1]
11 |   a^(0)[2i + 1] ← rnd
12 |   a^(1)[2i + 1] ← (T[h >> 4] − rnd + r) mod q
13 |   i ← i + 1
14 end
15 return a^(·)[N]
```

3.5 Masked Comparison

Comparing the original ciphertext to the re-encrypted ciphertext at the end of the FO-Transform (cf. Sect. 2.3) has to be appropriately protected as well because any leakage point in this function can compromise the security of the complete scheme [24,27]. The first approach to do so was proposed by Oder et al. [40]. They separately compare the public input ciphertext parts c_1, c_2 with their re-encrypted counterparts $\tilde{c}_1, \tilde{c}_2$. The methodology requires one randomized share $\tilde{c}_1^{(0)}$ to be subtracted from the public ciphertext c_1 yielding a randomized value. In case that $c_1 = \tilde{c}_1^{(0)} + \tilde{c}_1^{(1)}$ it is also true that $\mathcal{H}(c_1 - \tilde{c}_1^{(0)}) = \mathcal{H}(\tilde{c}_1^{(1)})$. If the re-encrypted ciphertext is different, the hash values yield different results. Thus, the result of $\mathcal{H}(c_1 - \tilde{c}_1^1) \oplus \mathcal{H}(\tilde{c}_1^2)$ does not leak any secret information. It yields

zero if the ciphertext parts are equal and a random number if they are not equal. The major drawback of this method is that it can not be used for higher orders. Additionally, this method is susceptible to the same attack vector as the higher-order compatible work by Bache et al. [41] as demonstrated by Bhasin et al. [27] in 2021. The partial unmasking of ciphertexts allows an attacker to distinguish between crafted ciphertexts that are re-encrypted identically or completely different depending on the error that was added to a valid ciphertext. In [15], the hash-based approach is taken and the two ciphertext parts are combined into one hash. Still, internally a Keccak-based hash is split up into multiple parts. The attack by D'Anvers et al. [42] makes use of this property. They propose another fast higher-order compatible comparison algorithm that incorporates the idea of [41] without partially unmasking the ciphertext. The algorithm outperforms the solution by [13], which compares uncompressed coefficients for second and higher orders. In line with the findings of [43] and the previously presented first-order optimized A2B and B2A conversions, we choose the so-called "simple" approach from [43, Algorithm 7] for our masked comparison.

3.6 Keccak (SHA3) as a Speed-Up

In this section, we propose a faster alternative to the presented NTTRU scheme when masking is in place. As described in Sect. 2.4 the SHA3 standard can replace the SHA2 functions without loss of security and offers the advantage of the underlying function Keccak does not need any arithmetic operations to compute the hash value. This is especially beneficial to any masked implementation because any masking conversion, especially at higher orders, requires a large computational overhead. In detail, the runtime is of magnitude $O(n^2k)$ [34] for a k bit variable in n shares. As SHA512 operates on 64 bit variables, this is a very costly operation that should be avoided if possible. In Keccak, all variables are shared in Boolean domain and the non-linear χ step is very efficient to mask as it includes only one `AND` operation. Although SHA2 seems to be the faster method of hashing with no side-channel countermeasures in place, as the authors of NTTRU state, it is recommended to use the SHA3 option when side-channel security has to be considered.

4 Evaluation

4.1 Performance Evaluation

In this work, we mostly use adapted code from the reference implementation [7] written in `C`. We also make use of a masked AES128 [35] in `C`. It has to be emphasized that most of the base code has a lot of potential in terms of performance. Furthermore, we build some functions on the fixed A2B conversion by Van Beirendonck et al. [33] which is optimized for the Cortex-M4 in terms of side-channel leakage. Additionally, we use the first-order implementation of Keccak for the SHA3 and SHAKE functions presented in [44]. A Cortex-M4

optimized implementation might lead to a faster first-order masked scheme than Kyber on this platform.

We measured the performance of our masked primitives on an ARM Cortex M4 mounted on an STM32F407G-DISC1 board offering up to 192 kByte of RAM. This environment was chosen as it is also the base microcontroller for the PQM4 project [45] for post-quantum algorithms. This is also why a lot of highly optimized code such as the masked assembler SHA3 already exists for this platform. Additionally, many masked implementations,e.g. of Kyber or Saber, exist for the ARM Cortex M4 leading to direct comparability of NTTRU with the NIST finalists. For our benchmarks, we set the clock frequency to 24 MHz. To improve the comparability between platforms we excluded cycle counts required for the randomness generation. For our evaluation, we did not use the onboard TRNG of the STM32F407-DISC1 board and opted for a pseudorandom number generator in software to generate the required masks. This enables easier debugging across several chips. As the development environment, we used the Keil Toolchain MDK Plus 5.29/µ Vision 5.29 with the ARM Compiler Version 5. The code size of our masked NTTRU decapsulation implementation is around 18 kB and the RAM requirement is around 77 kB.

Table 1. CCA2-secure decapsulation cycle counts for different masked lattice-based schemes.

Scheme	CPU	Cycles $\times 10^3$	Cycles $\times 10^3$
		Masked	Unmasked
Saber [15]	Cortex M4	2833	774
Kyber768 [12]	Cortex M4	2978	783
NTRU [17]	Cortex M3	32 472	10 508
NTTRU (This work)	Cortex M4	9448	796
NTTRU-SHA3 (This work)	Cortex M4	3119	

We give a comparison of performance numbers in Table 1. Using a state-of-the-art masked implementation of the SHA3-512 [44] and additionally replacing the non-optimized AES256 with the SHAKE256 option, we achieve a performance number for the first-order implementation of NTTRU that is in the magnitude of the NIST standardization candidate Kyber. We additionally give more in-depth performance numbers in Table 2. Once again, we emphasize that the polynomial arithmetic functions are not optimized for the ARM Cortex M4.

4.2 Side-Channel Evaluation

In this section, we show that our proposed techniques indeed fulfill the requirement of practical first-order security. We used the ChipWhisperer Lite Board with an STM32F303 providing an ARM Cortex M4 core running at 7.37 MHz.

Table 2. Cycle Counts for the masked components of NTTRU

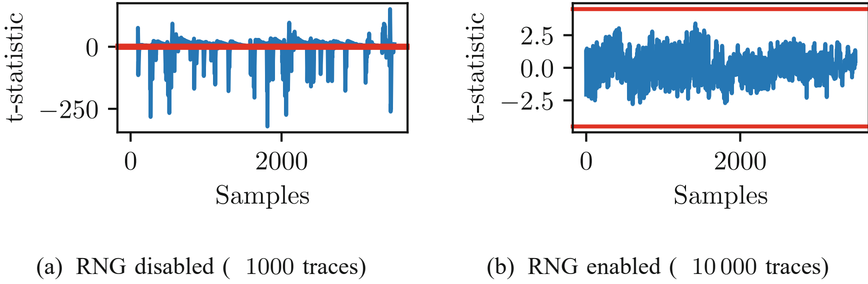
Function	Cycle count		Factor
	Unmasked	1st order	
`poly_unpack_uniform`	(19 396)	0	n.a.
`ntru_decrypt`	241 164	749 966	×3.1
polynomial arithmetic		436 214	
`poly_crepmod3`		313 713	
`poly_pack_short`	4170	96 261	×23
`SHA512`	27 305	4 359 092	×159
`crypto_stream` (AES)	24 028	2 808 228	×116
`ntru_encrypt`	436 570	962 539	×2.2
`poly_short`		106 277	
polynomial arithmetic		856 212	
`comparison`	4998	423 309	×84.7
`crypto_kem_dec`	796 712	9 448 510	×11.9

The sampling rate is four times the clock speed, resulting in 29 MS/s. An advantage of the CWLite board is the synchronized sample and device clock. It is relatively easy to capture small differences in power traces because the traces are perfectly aligned [46]. This lowers the amount of required power traces to detect possible leakage. A disadvantage lies in the small buffer size of around 24,400 samples. We circumvented this issue by capturing only small building blocks of the algorithm independently. For the ChipWhisperer evaluation, we compiled our code using `arm-none-eabi-gcc` version `10.3.1`. We show that our approaches do not have any obvious leakage points when implemented in practice. We applied the so-called non-specific t-test methodology by Schneider and Moradi [47] to do so. The inputs to the functions are either from a specific fixed ciphertext or a completely randomized ciphertext. We denote the set of traces obtained from function calls with fixed input as $\mathcal{S}_1$ and the set of traces obtained from random inputs as $\mathcal{S}_0$. Sample sizes n_0,n_1, standard deviations s_0, s_1 and sample means μ_0, μ_1 are denoted accordingly. At every point in time, we calculate the t-test statistic

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}} \tag{12}$$

The methodology by [47] requires a higher t value than 4.5 to correctly reject the hypothesis that both sets are not distinguishable with the confidence of around 99.999%. Thus, in a first-order secure implementation, all absolute values should be smaller than 4.5.

The first target is the table-based modulus conversion (Sect. 3.1). We adjusted our implementation slightly by generating the required random num-

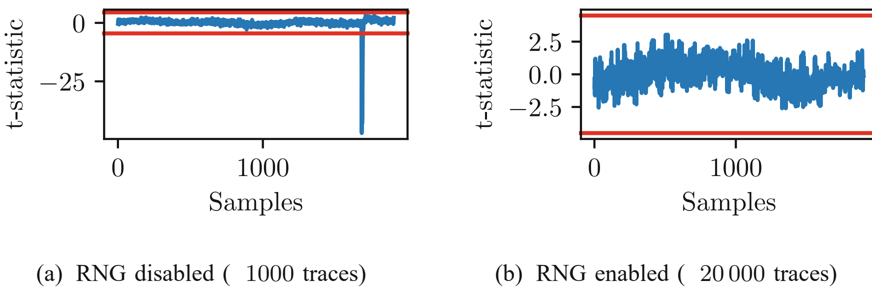


(a) RNG disabled (1000 traces) (b) RNG enabled (10 000 traces)

Fig. 3. t-statistic of the masked modulus conversion. Red lines indicate the threshold of 4.5. (Color figure online)

bers in advance. The generation is due to the rejection sampling mod q not constant time and would make our t-test useless. It is also not necessary to capture the complete conversion of the polynomial. It is sufficient to capture the conversion of only one coefficient as the conversion of all other coefficients is independent and redundant. Our first measurement was taken with the random number generator disabled. Thus, all masks are zero and the values are processed unmasked. In a correct setup of the side-channel setup, one should be able to see a lot of leaking points in this implementation. Therefore, Fig. 3a verifies our correct setup. Even with only 1000 traces several very high t-values can be seen.

We then activated our pseudorandom number generator. The obtained t-test values are visualized in Fig. 3b. We can see that even with 20000 traces and a sampling rate of four times per clock cycle no leakage peaks can be identified. Note that the hardened implementation requires a few minor tweaks and carefully crafted assembly routines to counter microarchitectural leakage.



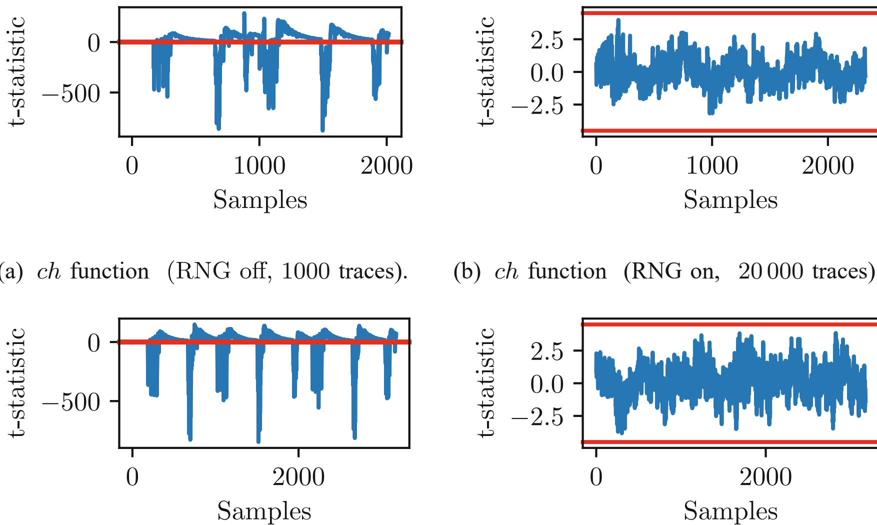
(a) RNG disabled (1000 traces) (b) RNG enabled (20 000 traces)

Fig. 4. t-statistic of the masked coefficient sampler. Red lines indicate the threshold of 4.5. (Color figure online)

For the sampling technique (Sect. 3.4), we performed a similar evaluation. We obtained the t-statistics visualized in Fig. 4a. The single leakage peak in the unmasked implementation stems from the assignment of the table value to the

second share of the coefficient. This corresponds to line 8 in Algorithm 8. The huge part without leakage corresponds to the generation of the table which is independent of the secret information. We can not identify leakage peaks with RNG enabled and the amount of 20000 traces and, thus, conclude that our implementation does not contain any obvious first-order leakage points.

In this work, we additionally presented a first-order masked SHA512 (Sect. 3.3). For the sake of simplicity, we evaluate only the non-linear choice (*ch*) and majority (*maj*) functions in this chapter. The functions that can be calculated on each share separately are easy to mask in practice with appropriate microarchitectural countermeasures in place, e.g. clearing registers or the ALU [33]. We show the results in Fig. 5.



(a) *ch* function (RNG off, 1000 traces). (b) *ch* function (RNG on, 20 000 traces).

Fig. 5. t-statistic of SHA512 functions. Red lines indicate the threshold of 4.5. (Color figure online)

5 Conclusion

The results once again show that a large performance gap between unprotected and protected implementations may more or less strongly impede the applicability of a scheme. As the first-order masking countermeasure can be seen as a minimum requirement nowadays, one should, if possible, aim for the usage of functions with minimal cost when masked. In detail, we strongly encourage the usage of SHA3 functions. As we have shown, their behavior with respect to additive and boolean masking allows NTTRU to be competitive among the first-order masked lattice-based schemes without reducing its security level. A lot of potential is additionally hidden in an optimized version of the NTT for

the Cortex M4 which is already available for Kyber. Such further optimizations combined with our proposed NTTRU-SHA3, might outperform masked implementations of the NIST finalists significantly on ARM Cortex-M4.

Acknowledgments. The authors would like to thank Thomas Pöppelmann and Peter Pessl for their valuable feedback and discussions. This work was supported by the German Federal Ministry of Education and Research (BMBF) under the project Aquorypt (16KIS1017). Presented project results were partly supported by the project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 830927.

References

1. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)
2. National Institute of Standards and Technology. Announcing request for nominations for public-key post-quantum cryptographic algorithms (2016). https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms
3. Avanzi, R., et al.: Crystals-kyber (version 3.02) - submission to round 3 of the nist post-quantum project (2021). https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf
4. Basso, A., et al.: SABER: Mod-LWR based KEM (round 3 submission) (2019). https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/SABER-Round3.zip
5. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a ring-based public key cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054868
6. Chen, C., et al.: Ntru - algorithm specifications and supporting documentation (2019). https://ntru.org/f/ntru-20190330.pdf
7. Lyubashevsky, V., Seiler, G.: NTTRU: truly fast NTRU using NTT. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(3), 180–201 (2019)
8. OpenSSH. Openssh release 9.0. https://www.openssh.com/txt/release-9.0. Accessed 14 Nov 2022
9. ISE Crypto PQC working group. Securing tomorrow today: Why google now protects its internal communications from quantum threats. https://cloud.google.com/blog/products/identity-security/why-google-now-uses-post-quantum-cryptography-for-internal-comms?hl=en. Accessed 21 November 22
10. Albrecht, M.R., Curtis, B.R., Deo, A., Davidson, A., Player, R., Postlethwaite, E.W., Virdia, F., Wunderer, T.: 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
11. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
12. Heinz, D., et al.: First-order masked kyber on ARM cortex-m4. IACR Cryptol. ePrint Arch., p. 58 (2022)
13. Bos, J.W., Gourjon, M., Renes, J., Schneider, T., van Vredendaal, C.: Masking kyber: first- and higher-order implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 173–214 (2021)

14. Fritzmann, T., et al.: Masked accelerators and instruction set extensions for post-quantum cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(1), 414–460 (2022)
15. Van Beirendonck, M., D'Anvers, J.-P., Karmakar, A., Balasch, J., Verbauwhede, I.: A side-channel-resistant implementation of SABER. ACM J. Emerg. Technol. Comput. Syst. **17**(2), 10:1–10:26 (2021)
16. Kundu, S., D'Anvers, J.-P., Van Beirendonck, M., Karmakar, A., Verbauwhede, I.: Higher-order masked saber. IACR Cryptol. ePrint Arch., p. 389 (2022)
17. Coron, J.-S., Gérard, F., Trannoy, M., Zeitoun, R.: High-order masking of NTRU. IACR Cryptol. ePrint Arch., p. 1188 (2022)
18. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU prime. IACR Cryptol. ePrint Arch., p. 461 (2016)
19. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_34
20. National Institute of Standards and Technology. Secure hash standard (2015). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
21. 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
22. Hermelink, J., Pessl, P., Pöppelmann, T.: Fault-enabled chosen-ciphertext attacks on kyber. In: Adhikari, A., Küsters, R., Preneel, B. (eds.) INDOCRYPT 2021. LNCS, vol. 13143, pp. 311–334. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92518-5_15
23. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 513–533. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_25
24. Guo, Q., Johansson, T., Nilsson, A.: A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 359–386. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_13
25. Ravi, P., Roy, S.S., Chattopadhyay, A., Bhasin, S.: Generic side-channel attacks on cca-secure lattice-based PKE and kems. IACR Trans. Cryptogr. Hardw. Embed. Syst., **2020**(3), 307–335 (2020)
26. Ravi, P., Bhasin, S., Roy, S.S., Chattopadhyay, A.: Drop by drop you break the rock - exploiting generic vulnerabilities in lattice-based pke/kems using em-based physical attacks. IACR Cryptol. ePrint Arch., p. 549 (2020)
27. Bhasin, S., D'Anvers, J.-P., Heinz, D., Pöppelmann, T., Van Beirendonck, M.: Attacking and defending masked polynomial comparison for lattice-based cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 334–359 (2021)
28. Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., van Vredendaal, C.: Chosen ciphertext k-trace attacks on masked CCA2 secure kyber. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 88–113 (2021)
29. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26

30. 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
31. Coron, J.-S., Tchulkine, A.: A new algorithm for switching from arithmetic to boolean masking. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 89–97. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45238-6_8
32. Debraize, B.: Efficient and provably secure methods for switching from arithmetic to boolean masking. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 107–121. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_7
33. Van Beirendonck, M., D'Anvers, J.-P., Verbauwhede, I.: Analysis and comparison of table-based arithmetic to boolean masking. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 275–297 (2021)
34. 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
35. Riou, S.: Masked bitsliced aes128. https://github.com/sebastien-riou/masked-bit-sliced-aes-128. Accessed 27 Sept 2022
36. Schwabe, P., Stoffelen, K.: All the AES you need on cortex-M3 and M4. In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 180–194. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_10
37. ANSSI LSC. Technical analysis of the masked aes implementation. https://github.com/ANSSI-FR/SecAESSTM32/blob/master/doc/technical-report/technical_analysis.pdf. Accessed 21 Nov 2022
38. Zijlstra, T., Bigou, K., Tisserand, A.: FPGA implementation and comparison of protections against SCAs for RLWE. In: Hao, F., Ruj, S., Sen Gupta, S. (eds.) INDOCRYPT 2019. LNCS, vol. 11898, pp. 535–555. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35423-7_27
39. Heinz, D., Pöppelmann, T.: Combined fault and DPA protection for lattice-based cryptography. IACR Cryptol. ePrint Arch., p. 101 (2021)
40. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical cca2-secure and masked ring-lwe implementation. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 142–174 (2018)
41. Bache, F., Paglialonga, C., Oder, T., Schneider, T., Güneysu, T.: High-speed masking for polynomial comparison in lattice-based kems. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 483–507 (2020)
42. D'Anvers, J.-P., Heinz, D., Pessl, P., Van Beirendonck, M., Verbauwhede, I.: Higher-order masked ciphertext comparison for lattice-based cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(2), 115–139 (2022)
43. D'Anvers, J.-P., Van Beirendonck, M., Verbauwhede, I.: Revisiting higher-order masked comparison for lattice-based cryptography: Algorithms and bit-sliced implementations. IACR Cryptol. ePrint Arch., p. 110 (2022)
44. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Building power analysis resistant implementations of Keccak (2010)
45. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: pqm4: testing and benchmarking NIST PQC on ARM cortex-m4. IACR Cryptol. ePrint Arch., p. 844 (2019)

46. O'Flynn, C., Chen, Z.D.: ChipWhisperer: an open-source platform for hardware embedded security research. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 243–260. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10175-0_17
47. Schneider, T., Moradi, A.: Leakage assessment methodology - extended version. J. Cryptogr. Eng. **6**(2), 85–99 (2016)