Hardware Implementation and Security Analysis of Local-Masked NTT for CRYSTALS-Kyber

Rafael Carrera Rodriguez[1,2], Emanuele Valea[2], Florent Bruguier[1] and Pascal Benoit[1]

[1] LIRMM, University of Montpellier, CNRS, Montpellier, France,
rafael.carrera-rodriguez@lirmm.fr, florent.bruguier@lirmm.fr,
pascal.benoit@lirmm.fr
[2] Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France,
rafael.carrerarodriguez@cea.fr, emanuele.valea@cea.fr

Abstract. The rapid evolution of post-quantum cryptography, spurred by standardization efforts such as those led by NIST, has highlighted the prominence of lattice-based cryptography, notably exemplified by CRYSTALS-Kyber. However, concerns persist regarding the security of cryptographic implementations, particularly in the face of Side-Channel Attacks (SCA). The usage of operations like the Number Theoretic Transform (NTT) in CRYSTALS-Kyber introduces vulnerabilities to SCA, especially single-trace ones, such as soft-analytical side-channel attacks. To address this threat, Ravi et al. proposed local masking as a countermeasure by randomizing the NTT's twiddle factors, but its implementation and security implications require further investigation. This paper presents a hardware implementation of the NTT with local masking, evaluating its performance, area utilization, and security impacts. Additionally, it analyzes the vulnerabilities inherent in local masking and assesses its practical security effectiveness through non-specific t-tests, showing that there are configurations of local masking that are more prone to leakage than others.

Keywords: NTT · local masking · hardware implementation · SASCA · Kyber

1 Introduction

The landscape of post-quantum cryptography has been evolving quickly ever since efforts from government agencies, such as United States' NIST, proposed standardization processes for new cryptosystems [Nat]. With such efforts, researchers from all around the globe intensified the analysis on existing cryptosystems and new proposed ones. One of the most promising categories in NIST's contest for post-quantum cryptography was lattice-based cryptography, for its balance between security and performance. As a matter of fact, after round 3, the algorithm chosen as a standard for the category of Key-Encapsulation Mechanism is CRYSTALS-Kyber [ABD+21], which is based on Module-Learning with Errors [LS15], a problem in the lattice-based cryptography field.

Even though the scientific community is fairly confident on the security of many propositions in the NIST's contest and of the chosen standards, the security of the implementations of such propositions is another problem that is usually not tackled in the design of the algorithms. As a matter of fact, most of these algorithms assume a black-box model in which an attacker cannot access intermediate values in the algorithm. However, with the discovery of Side-Channel Attacks (SCA) [KJJ99], such assumption does not hold, specially in the case of embedded devices whose power consumption heavily depends on the execution of the cryptosystem in question.

SCA are not new, and multiple countermeasures have been proposed in the literature to reduce their effectiveness. However, with the introduction of new cryptosystems, the attack surface of these algorithms increased, with the usage of modules that were not analyzed or used before. Such is the case of the Number Theoretic Transform (NTT). This operation is used in CRYSTALS-Kyber to accelerate the polynomial multiplication. The polynomials are transformed to the NTT domain with a complexity of $O(n\log n)$ and then multiplied, with linear complexity, with the so-called *pointwise multiplication*. This operation is the most obvious candidate for a first-order differential power analysis, since in CRYSTALS-Kyber, the secret key is directly multiplied with part of the ciphertext [MWK+22, CBVB22]. This type of attack is thwarted easily with arithmetic masking, since the NTT is a linear operation [RRVV15, BGR+21].

However, single-trace attacks may pose different challenges since they are not particularly affected by arithmetic masking, apart from a decrease on the signal-to-noise ratio. In [PPM17], Primas *et. al.* first proposed one of such attacks against an NTT implementation. They adapted the Soft-Analytical Side-Channel Attack (SASCA) first proposed by Veyrat-Charvillon *et al.* [VCGS14]. This attack first performs a template matching phase [CRR03] and afterwards it applies the so-called Belief Propagation algorithm to combine the different probability distributions to reduce entropy in the guess of the secrets. Subsequently, further works increased the effectiveness and resilience of this type of attack against noise by changing the leakage target and making adaptations to the Belief Propagation algorithm [PP19], or by using chosen ciphertexts to induce zeros after the pointwise multiplication to reduce the entropy [HHP+21].

To prevent this type of attacks, Ravi *et al.* [RPBC20] proposed a suite of countermeasures, with the goal of randomizing operations or the order of the operations. They proposed *shuffling* and *local masking*, in different configurations each. Shuffling changes the order of operations, while local masking aims to randomize the multiplication of intermediates with constants in the NTT, called *twiddle factors*. Shuffling has been well analyzed in the literature. A security analysis in [HSST22], notes that this type of countermeasure does add to the security of an NTT implementation, if implemented correctly. Specially, they notice that the *coarse shuffling* version seems to be more resilient to adaptations of SASCA. Additionally, there are existing hardware implementations of a shuffled NTT [ZBT19, CMJ22]. However, to our knowledge, the literature does not report neither hardware implementations nor security analysis of the local masking countermeasure.

Our contributions in this paper are threefold. *First*, in order to analyze its implications on performance, area utilization and security, we propose a hardware implementation of an NTT equipped with the aforementioned countermeasure in a flexible manner. *Second*, we make an analysis of the possible vulnerabilities intrinsic to the local masking countermeasure. *Finally*, we analyze with a non-specific t-test, the practical security effects of this countermeasure in our proposed hardware implementation.

The organization of this paper is as follows: In Section 2, we provide all the background necessary to understand this paper. Then, in Section 3, we describe our proposed implementation of a local-masked NTT. Next, in Section 4, we present an analysis on the possible vulnerabilities of this countermeasure. In Section 5, we explore practically a security analysis of our proposed implementation. Finally, Section 6 draws conclusions.

2 Background

In this section, we provide the essential background required to grasp our contribution. Initially, we establish the notation employed throughout this article. Subsequently, we introduce NIST's standard for key encapsulation mechanism, CRYSTALS-Kyber. Following this, we explain the operation of one of Kyber's core modules, the Number Theoretic Transform. Afterwards, we outline a horizontal, and potentially single trace, attack on this

module, termed Soft-Analytical Side-Channel Attack (SASCA). Lastly, we describe the local masking countermeasure proposed by Ravi et al. [RPBC20] to mitigate this attack, which serves as the focal point of analysis in this paper.

2.1 Notation

The ring of integers modulo q is denoted as $\mathbb{Z}_q$. The polynomial ring defined over the previously mentioned ring, $\mathbb{Z}_q(x)/\phi(x)$, is represented as R_q, where $\phi(x)$ is the reduction polynomial, of degree n. When a polynomial f is defined as an element of such ring, the i-th coefficient of such polynomial is denoted as f_i. The representation of such polynomial in the number theoretic transform domain is denoted as $\hat{f}$. A polynomial vector is denoted with bold letters, *e.g.*, $\boldsymbol{a}$. When referring to a polynomial matrix, uppercase letters will be used, *e.g.*, $\boldsymbol{A}$.

2.2 CRYSTALS-Kyber

CRYSTALS-Kyber [ABD+21] is a key encapsulation mechanism (KEM), chosen by NIST to be one of the first PQC standards. It is composed of two basic parts. First, it comprises an indistinguishable under chosen-plaintext attack (IND-CPA) public key encryption scheme (PKE), with its security relying on the hardness of breaking the Module Learning with Errors problem (M-LWE)[LS15, Reg05]. Then, it executes a Fujisaki-Okamoto transform [FO99], in order to convert it to an indistinguishable under chosen-ciphertext attack (IND-CCA) KEM, by re-encrypting in decapsulation and checking if the output is equal to the sent ciphertext.

CRYSTALS-Kyber uses polynomials in the ring R_q, where $q = 3329$ and the reduction polynomial is a cyclotomic polynomial $\phi(x) = x^n + 1$, where $n = 256$. The flexible parameter is k, which is the rank of the module lattice in this cryptosystem and it dictates the size of vectors and matrices, as well as the value of other subparameters.

In the succeeding portion of this subsection, we will outline a simplified form of the underlying Public Key Encryption (PKE) within CRYSTALS-Kyber. In Algorithm 1, we show the procedure of the PKE key generation. First, seeds ρ and σ are chosen. Then, the public matrix $\hat{\boldsymbol{A}} \in R_q^{k \times k}$ is generated from the seed ρ, sampling it from a uniform distribution. This matrix is sampled in the Number-Theoretic Transform (NTT) domain, which is an operation used to accelerate polynomial multiplication. Next, the vectors $\boldsymbol{s}, \boldsymbol{e}$ are sampled from a binomial distribution using the seed σ. Afterwards, the secret vector $\boldsymbol{s}$ is transformed to the NTT domain. Following that, the M-LWE instance $\hat{\boldsymbol{t}}$ is obtained by multiplying $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{s}}$ and then adding the error $\boldsymbol{e}$ in the NTT domain. Finally, the keys are returned, with $pk = (\hat{\boldsymbol{t}}, \rho)$ as the public key and $sk = \hat{\boldsymbol{s}}$ as the secret key.

Algorithm 1 CRYSTALS-Kyber PKE KeyGen (Simplified)

1: **Output:** Public key pk, secret key sk
2: Draw seeds ρ, σ
3: $\hat{\boldsymbol{A}} = \text{Sample}_U(\rho)$
4: $(\boldsymbol{s}, \boldsymbol{e}) = \text{Sample}_B(\sigma)$
5: $\hat{\boldsymbol{s}} = \text{NTT}(\boldsymbol{s})$
6: $\hat{\boldsymbol{t}} = \hat{\boldsymbol{A}} \circ \hat{\boldsymbol{s}} + \text{NTT}(e)$
7: **return** $pk = (\hat{\boldsymbol{t}}, \rho)$, $sk = \hat{\boldsymbol{s}}$

We now show the procedure for the PKE encryption in Algorithm 2. First, the public matrix $\hat{\boldsymbol{A}}$ is regenerated from ρ. Then, the randomizer vector $\boldsymbol{r}$ and the errors $\boldsymbol{e}_1, e_2$ are sampled from a binomial distribution using the seed τ. Next, the vector $\boldsymbol{r}$ is transformed into the NTT domain. Then, an M-LWE instance $\hat{\boldsymbol{u}}$ is generated from $\hat{\boldsymbol{A}}$ transposed, $\hat{\boldsymbol{r}}$

and the error e_1. Finally, an encoding of the message in R_q is embedded into another M-LWE instance, composed of $\hat{\boldsymbol{t}}, \hat{\boldsymbol{r}}$ and e_2. This final operation is the vector v. Finally, the ciphertext is returned as $c = (\boldsymbol{u}, v)$.

Algorithm 2 CRYSTALS-Kyber PKE Encryption (Simplified)

1: **Input:** Public key $pk = (\hat{\boldsymbol{t}}, \rho)$, message m, seed τ
2: **Output:** Ciphertext $c = (\boldsymbol{u}, v)$
3: $\hat{\boldsymbol{A}} = \text{Sample}_U(\rho)$
4: $(\boldsymbol{r}, \boldsymbol{e}_1, e_2) = \text{Sample}_B(\tau)$
5: $\hat{\boldsymbol{r}} = \text{NTT}(\boldsymbol{r})$
6: $\boldsymbol{u} = \text{NTT}^{-1}(\hat{\boldsymbol{A}}^T \circ \hat{\boldsymbol{r}}) + \boldsymbol{e}_1$
7: $v = \text{NTT}^{-1}(\hat{\boldsymbol{t}} \circ \hat{\boldsymbol{r}}) + e_2 + \text{Encode}(m)$
8: **return** $c = (\boldsymbol{u}, v)$

Finally, in Algorithm 3, we describe the decryption procedure. The vector $\boldsymbol{u}$ is transformed into the NTT domain and multiplied with the secret key $\hat{\boldsymbol{s}}$. Then, v is subtracted from the result of this multiplication. The result of this subtraction is then decoded to retrieve the message m.

Algorithm 3 CRYSTALS-Kyber PKE Decryption (Simplified)

1: **Input:** Secret key $sk = \hat{\boldsymbol{s}}$, ciphertext $c = (\boldsymbol{u}, v)$
2: **Output:** Message m
3: $m = \text{Decode}(v - \text{NTT}^{-1}(\hat{\boldsymbol{s}}^T \circ \text{NTT}(\boldsymbol{u})))$
4: **return** m

2.3 Number Theoretic Transform

In CRYSTALS-Kyber, in order to reduce complexity for the polynomial multiplications, the authors choose to use the Number Theoretic Transform (NTT), which works like a Fast-Fourier Transform over the integer ring. It performs the mapping from R_q to the vectorial space defined by the factorization of the reduction polynomial $\phi(x)$. To obtain only linear factors with degree 0, $\phi(x)$ must be factorized into n polynomials of degree 1, which can be obtained if the ring of integers modulo q contains $2n\text{-}th$ primitive roots of unity. However, because of the modulus q chosen in CRYSTALS-Kyber, the ring $\mathbb{Z}_q$ contains only $n\text{-}th$ primitive roots of unity. This means that the defining polynomial of CRYSTALS-Kyber cannot be factorized in 256 polynomials of degree 1, but in 128 polynomials of degree 2. Therefore, the NTT mapping of a polynomial $f \in R_q$ is a vector of 128 polynomials of degree two. Then, according to the specification, for $\zeta = 17$, as the first $256\text{-}th$ primitive root of unity and $\text{br}_7(\cdot)$ as the 7-bit-reversion operation, NTT is defined as follows:

$$\hat{f}_{2i} = \sum_{j=0}^{127} f_{2j} \zeta^{(2\text{br}_7(i)+1)j} \qquad \hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2j+1} \zeta^{(2\text{br}_7(i)+1)j} \tag{1}$$

Multiplication of two elements $f, g \in R_q$ is a polynomial multiplication. This operation, transformed in the NTT domain, becomes a point-wise multiplication (PWM), $\hat{f} \circ \hat{g} = \hat{h}$. For Kyber's way of executing NTT, PWM is defined as follows:

$$\hat{h}_{2i} = \hat{f}_{2i}\hat{g}_{2i} + \hat{f}_{2i+1}\hat{g}_{2i+1} \cdot \zeta^{2br_7(i)+1} \qquad \hat{h}_{2i+1} = \hat{f}_{2i}\hat{g}_{2i+1} + \hat{f}_{2i+1}\hat{g}_{2i} \tag{2}$$

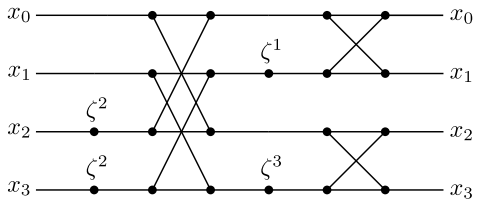


Figure 1: Example for an execution of the NTT with $n = 4$.

After the PWM, an inverse NTT (INTT) is performed to retrieve the result in normal domain.

For memory saving purposes, the NTT is usually implemented as an in-place transform. In the CRYSTALS-Kyber documentation, when executing NTT, the inputs are in normal order (NO) and the output is in bit-reversed order (BR). When executing INTT, the input is in BR and the output is in NO. This suggests the usage of Cooley-Tukey (CT) butterflies (3) for the NTT and Gentleman-Sande (GS) butterflies (4) for the INTT. In this equations, ω represents a power of ζ, the first primitive n-th root of unity. This powers are called *twiddle factors* in the FFT/NTT literature. A graphical representation of a forward NTT with 4 coefficients is shown in Figure 1.

$$c = a + b \cdot \omega \qquad d = a - b \cdot \omega \tag{3}$$

$$c = a + b \qquad d = (a - b) \cdot \omega \tag{4}$$

2.4 Soft-Analytical Side-Channel Attacks on NTT

Soft-Analytical Side-Channel Attacks (SASCA) are a type of attacks that combine information from traces and from the algorithm itself. They were proposed for the first time by Veyrat-Charvillon *et. al.*.[VCGS14]. Such attacks can potentially be single-trace attacks, and therefore, they can beat masked implementations, since they are able to recover multiple shares of a secret at a time. They work by representing the attack as a noisy decoding problem. The first step is to create templates and to perform profiling on some intermediate values [CRR03]. This yields conditional probabilities on the leakage trace ℓ, $\Pr(T = t|\ell)$, where T is an intermediate value in the algorithm. Then, the Belief Propagation (BP) algorithm is used to combine the probabilities obtained for the different intermediates using a graph representation of the attacked algorithm called a factor graph.

Primas et. al. executed the first SASCA on NTT [PPM17]. In their work, they profiled the modular multiplication between intermediate coefficients and twiddle factors. Their work targeted the inverse NTT in the decryption procedure of the LPR scheme [LPR10], which is a precursor of CRYSTALS-Kyber. For CRYSTALS-Kyber, the analog target operation would be the inverse NTT in line 3 of Algorithm 3. After this work, Pessl and Primas [PP19] made some improvements over the previous work, augmenting its practicality. Their target is the NTT of the randomizer vector $\hat{\boldsymbol{r}}$ in the encryption of CRYSTALS-Kyber in line 5 of Algorithm 2, that is drawn from a binomial distribution with a smaller support compared to a random uniform variable in $\mathbb{Z}_q$. Posterior to this, Hamburg et. al. [HHP+21] performed what they called a *k-trace* attack on CRYSTALS-Kyber. They target the INTT in the decryption at line 3 of Algorithm 3, by sending specially crafted ciphertexts. These chosen ciphertexts induce zeros after the PWM with the goal of reducing the entropy of the graph. The more sparse these ciphertexts are, the

more resilient to noise the attack is, albeit requiring more traces. For example, with 64 non-zero coefficients, the attack requires k traces, $k \in \{2, 3, 4\}$ being the security level of the attacked version of CRYSTALS-Kyber.

Ravi et. al. [RPBC20] proposed two types of countermeasures for these SASCA attacks on NTT, namely *shuffling* and *local masking*. The shuffling countermeasures were analyzed by Hermelink et. al. [HSST22] where they introduce techniques such as mixing priors, a shuffling factor node or extraction of permutations with subgraphs to attack this countermeasure. They show that the shuffling countermeasure is a very valid tool and in certain instances (specially in the *coarse full shuffling*), it does increase the security of an NTT implementation against SASCA, reducing its practicality.

2.5 Local masking countermeasure

In [RPBC20], a masking countermeasure for protecting the NTT against SASCA attacks was proposed. This *local masking* scheme works essentially by randomizing the twiddle factor in a butterfly. For example, take a CT butterfly as the one shown in (3). Let ζ^x be the twiddle factor of this butterfly. If the results c and d of this butterfly are multiplied by a random twiddle factor ζ^y, then it yields the following:

$$\begin{aligned} c' &= c \cdot \zeta^y \\ &= (a + b \cdot \zeta^x) \cdot \zeta^y \\ &= a \cdot \zeta^y + b \cdot \zeta^{x+y} \end{aligned} \qquad \begin{aligned} d' &= d \cdot \zeta^y \\ &= a \cdot \zeta^y - b \cdot \zeta^{x+y} \end{aligned} \tag{5}$$

As it can be observed in (5), this masking requires one extra multiplication compared to an unmasked CT butterfly, as in (3).

The usage of twiddle factors for local masking is justified from a convenience and performance point of view. For instance, the sampling of random twiddle factors ζ^y is done in the interval $[0, n)$, instead of the whole $\mathbb{Z}_q$. Additionally, removing the mask is equal to multiplying with the twiddle factor that has the power that is the additive inverse of y mod n, since all the twiddle factors are n-th roots of unity, $\zeta^n = 1$. Moreover, remasking an already masked intermediate coefficient, resolves on only adding the powers of ζ of the masks. For example, if the inputs of a butterfly are $a' = a \cdot \zeta^i$ and $b' = b \cdot \zeta^i$, masking with ζ^y is equivalent to:

$$\begin{aligned} c' &= (a' + b' \cdot \zeta^x) \cdot \zeta^y \\ &= a' \cdot \zeta^y + b' \cdot \zeta^{x+y} \\ &= a \cdot \zeta^{i+y} + b \cdot \zeta^{i+x+y} \end{aligned} \qquad d' = a \cdot \zeta^{i+y} - b \cdot \zeta^{i+x+y} \tag{6}$$

This type of butterfly has the same mask for both the inputs and the outputs and it is denoted as MSISO (mask same input, same output). It has a cost of two modular multiplications. Likewise, other masking strategies can be employed. For instance, if the inputs have different masks, as $a' = a \cdot \zeta^i$ and $b' = b \cdot \zeta^j$, the inputs can be unmasked and masked with the new ζ^y in a single operation like:

$$\begin{aligned} c' &= c \cdot \zeta^y \\ &= (a + b \cdot \zeta^x) \cdot \zeta^y \\ &= (a' \cdot \zeta^{n-i} + b' \cdot \zeta^{n-j+x}) \cdot \zeta^y \\ &= a' \cdot \zeta^{n-i+y} + b' \cdot \zeta^{n-j+x+y} \end{aligned} \qquad d' = a' \cdot \zeta^{n-i+y} - b' \cdot \zeta^{n-j+x+y} \tag{7}$$

This type of butterfly is denoted as MDISO (mask different input, same output) and it has the same cost of two modular multiplications. If the inputs are masked with different masks and the outputs must be masked in the same way, then it becomes more complex. Take $a' = a \cdot \zeta^i$ and $b' = b \cdot \zeta^j$, as the inputs that are masked with different masks. If c must be masked with ζ^k and d with ζ^l then the operations are the following:

$$\begin{aligned} c' &= c \cdot \zeta^y \\ &= (a + b \cdot \zeta^x) \cdot \zeta^k \\ &= (a' \cdot \zeta^{n-i} + b' \cdot \zeta^{n-j+k}) \cdot \zeta^y \\ &= a' \cdot \zeta^{n-i+k} + b' \cdot \zeta^{n-j+x+k} \end{aligned} \qquad d' = a' \cdot \zeta^{n-i+l} - b' \cdot \zeta^{n-j+x+l} \tag{8}$$

This type of butterfly is denoted as MDIDO (mask different input, different output) and it has a cost of four modular multiplications, and it is the costlier butterfly configuration. A similar type of butterfly can be derived for inputs with the same mask and outputs with different mask, denoted as MSIDO, with the same complexity as the MDIDO butterfly.

A similar set of equations can be derived for the Gentleman-Sande butterfly, in (4), with the difference that the cost of the butterflies MSIDO and MDISO are inverted, *i.e.*, two and four modular multiplications respectively. The cost for each of the masked butterflies is resumed in Table 1.

Table 1: Cost of masked butterflies in terms of modular multiplications. An unmasked butterfly requires only one multiplication.

Type of masking	**Modular multiplications** Cooley-Tukey / Gentleman-Sande
MSISO	2 / 2
MDISO	2 / 4
MDIDO	4 / 4
MSIDO	4 / 2

For a local-masked NTT execution, different configurations are proposed. If one mask is used per NTT stage, the cheapest butterflies, i.e. MSISOs, can be used. In this way, the last mask could be the additive inverse of the sum of the powers of the previous masks in mod n, such that the output is unmasked. This configuration is noted as *coarse masking*. Ravi *et al.* hint that this configuration can be the easiest to attack since an attacker could *brute-force* over the masking space or possibly exploit the links between the masks at the different butterflies.

To avoid this intuition, it is proposed to use n masks per stage, one for each coefficient, denoting it as *fine masking*. This configuration uses only the costliest MDIDO butterflies.

As a compromise between both of this versions, *generic masking* is proposed. In this version, the number u of masks is variable and possibly random. It is also proposed that u be limited to powers of 2, such that the control logic is less convoluted. Here, the kind of masked butterflies to be employed is dependent on the number of masks and in which stage they are used. An example is shown in Figure 2, where the type of mask per round is depicted for an NTT with $n = 8$ and $u = 2$ for an expanding NTT in (a) and a shrinking NTT in (b), which in a usual implementation of CRYSTALS-Kyber, they correspond to forward and inverse NTT respectively. In case (a), the first butterfly is a MSIDO butterfly for providing different masks for the outputs. The rest of the butterflies are MSISO. In case (b), all butterflies are MSISO, except the last one, since the inputs have different masks, necessitating a MDISO butterfly. *Generic masking* is not constant time, since it depends on the butterfly complexity which depends in the number u of masks used, but its runtime is not secret-dependent.

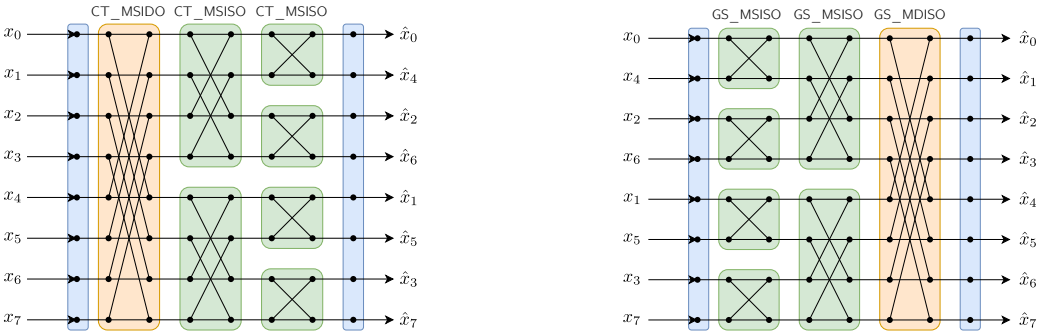


Figure 2: Example for an execution of the NTT with generic masking configuration with $n = 8$ and $u = 2$. Denoted in yellow are the layers that use 4-multiplication butterflies, whereas in green, the butterflies require 2 multiplications.

3 Implementation of NTT countermeasure

Since the NTT is a critical operation for the performance of an execution of CRYSTALS-Kyber, hardware accelerators have been proposed throughout the literature to handle this operation [YMÖS21, BNAMK21, IUH22, LTHW23]. Against SASCA attacks, there exist two hardware propositions that are equipped with the shuffling countermeasure [ZBT19, CMJ22]. However, there is no accelerator equipped with the local masking countermeasure, that we are aware of. In this section we propose an architecture capable to execute the countermeasure from [RPBC20] in a flexible fashion for the user. Our objective is to explore trade-offs between area, performance, and security.

We implement the *generic masking* proposition from [RPBC20], where the user can choose the number of masks for an execution of the NTT or the INTT, as a trade-off with performance. Making use of four modular multipliers in our architecture enables parallelization of modular multiplications, particularly advantageous when using butterflies requiring fewer than four modular multiplications. Additionally, the user can perform PWM, making it a full polynomial multiplication unit for CRYSTALS-Kyber.

The description of the architecture follows a top-down approach, explaining first the overview of the architecture in general, features and parameters of the module, then going down to each of the composing elements. Moreover, we provide timing and synthesis results for both FPGA and ASIC targets.

3.1 General overview of the architecture

We show the diagram of the architecture in Figure 3. As an usual NTT implementation, the main blocks of the architecture are: memory elements storing coefficients and twiddle factors; the processing element that executes the butterfly operations; the control unit to orchestrate everything. Our architecture also contains a module called *masking unit*, that is in charge of randomizing the twiddle factors and applying the desired countermeasure.

The processing element contains four modular multipliers, such that the most complex protected butterfly operations can be executed in one single clock cycle, such as CT-MSIDO, CT-MDIDO, GS-MDISO and GS-MDIDO, as shown in Table 1. It is also flexible, such that the multipliers can be used to parallelize butterfly operations by a factor of 2, in the case of the butterflies that are less complex, or by a factor of 4, in the case of an unprotected execution. The processing element is thoroughly discussed in Section 3.4.

The memories are divided in three sections: primary memory, secondary memory and twiddle factor (TWF) memory. The primary memory contains the coefficients that are used for an NTT/INTT operation. It is divided in 8 banks of 32x20 bits, each address

storing one coefficient and its current mask. The division in 8 banks is done in order to guarantee collision-free memory access, employing the scheme proposed by [MRW+22]. The secondary memory contains the second polynomial that is used in PWM operations. The TWF memory is a read-only memory that contains the 256 possible twiddle factor powers. The layout of the memories and their access schemes are explained in Section 3.5.

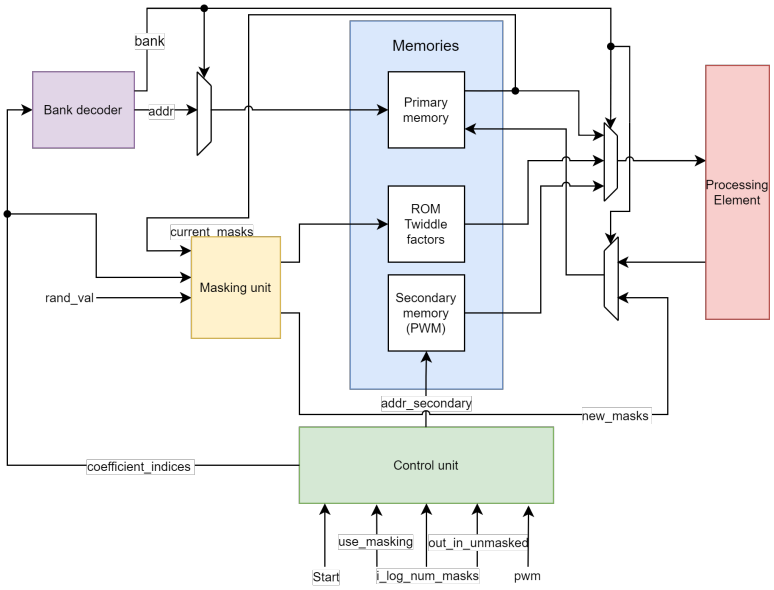


Figure 3: Simplified schematic of the proposed architecture for a local-masked NTT. The control signals from the control unit are omitted.

3.2 NTT/INTT features

The user can perform the NTT of a polynomial in an unprotected or protected fashion with different levels of security. As explained in Section 2.5, the performance of an implementation of [RPBC20], depends on the number of masks used per NTT stage, since it modifies the type of butterflies used. Additionally, the user may leave the output (resp. input) of an NTT (resp. INTT) execution masked as an additional protection for PWM, as explained in Section 3.3. Summarizing, the parameters given by the user for an NTT/INTT execution are:

- Flag `use_masking` indicates the usage of TWF masking countermeasure. Leaving this flag not asserted, yields the best performance for NTT/INTT, but without the countermeasure for the SASCA attack.

- Number u of masks per stage for TWF masking. For simplification of the control unit, this parameter is limited to powers of 2. Therefore the user provides $i = log_2(u)$. Additionally, i is constrained to be in the range $[0, 7]$, as explained in Section 3.3.

- Flag `out_in_unmasked` indicates that the output (resp. input) of an NTT (resp. INTT) is left or taken unmasked. This is for applying an additional protection for PWM, as explained in Section 3.3.

- Flag `pwm` to indicate the execution of a pointwise multiplication.

3.3 Additional protection for PWM

Local masking can be used to implement a blinding countermeasure against DPA/CPA for the PWM that was originally proposed by [Saa17]. In doing so, single-trace attack protection could be extended to DPA/CPA protection for the PWM.

Let the output of even and odd coefficients of an NTT in Kyber, masked by powers j and k, be

$$\hat{f}'_{2i} = \hat{f}_{2i} \cdot \zeta^j \qquad \hat{f}'_{2i+1} = \hat{f}_{2i+1} \cdot \zeta^k \tag{9}$$

Then, executing PWM between a masked polynomial $\hat{f}'$ and a polynomial $\hat{g}$, equals to

$$\begin{aligned} \hat{h}'_{2i} &= \hat{f}'_{2i} \cdot \hat{g}_{2i} + \hat{f}'_{2i+1} \cdot \hat{g}_{2i+1} \cdot \zeta^x \\ &= \hat{f}_{2i} \cdot \hat{g}_{2i} \cdot \zeta^j + \hat{f}_{2i+1} \cdot \hat{g}_{2i+1} \cdot \zeta^{k+x} \end{aligned} \qquad \begin{aligned} \hat{h}'_{2i+1} &= \hat{f}'_{2i} \cdot \hat{g}_{2i+1} + \hat{f}'_{2i+1} \cdot \hat{g}_{2i} \\ &= \hat{f}_{2i} \cdot \hat{g}_{2i+1} \cdot \zeta^j + \hat{f}_{2i+1} \cdot \hat{g}_{2i} \cdot \zeta^k \end{aligned} \tag{10}$$

If $k = j$, then $\hat{h}'_{2i} = \hat{h}_{2i} \cdot \zeta^j, \hat{h}'_{2i+1} = \hat{h}_{2i+1} \cdot \zeta^j$. This condition ensures that the real polynomial can be easily retrieved from the masked polynomial. Ensuring this equality becomes feasible by restricting the number of masks per stage of the NTT to 128. With this limitation, the pairs of coefficients that intervene in the PWM will be masked with the same values. Consequently, the parameter i is confined to the range $[0, 7]$.

After PWM, INTT is executed taking into account the masks of the coefficients incoming from the PWM.

This blinding countermeasure in the PWM protects against first order DPA/CPA attacks [KJJ99, BCO04] that target the modular multiplication [MWK+22, CBVB22]. However, it is important to note that writing to and reading from the module do not benefit from any SCA countermeasure. Therefore, it is expected that first-order leakage will occur during these operations.

3.4 Processing element

The processing element (PE) is designed to be flexible and be able to handle all of the different butterfly operations, for masked and unmasked configurations. Additionally, it can be configured for the execution of the PWM operation. The PE is comprised of two double butterfly units (DBUs) that can be cascaded to execute the operations for PWM.

A double butterfly unit (DBU) is the basic unit of the processing element. It can carry out two butterfly operations in the case of unprotected NTT, one butterfly operation in the case of the simplest protected butterflies or half of a butterfly operation when executing the most complex butterflies. It can also simply output the result of a modular multiplication, for the purposes of executing PWM. Its architecture is shown in Figure 4. Such unit is pipelined and has a latency of 9 clock cycles for executing a butterfly operation and 7 clock cycles for modular multiplication.

A butterfly operation is composed of two basic operations: multiply and combine, where the intermediate values are added. The order of these operations depends on the executed butterfly. For example, unprotected Cooley-Tukey, shown in (3) performs multiplication and then combination, while unprotected Gentleman-Sande, in (4) executes combination and then multiplication. The DBU has then a configurable combine stage, followed by a modular multiplication stage and a final configurable combine stage.

Each combine stage is equipped with a modular adder and a modular subtractor. These modules are merged with modular division by 2, in order to avoid the postscaling step by $n^{-1} \bmod q$ that is required by the INTT operation. In order to merge addition and subtraction with division by 2, we employ the known strategy proposed by [ZYC+20], which executes the $1/2$ operation with only shifts and one addition: $\frac{x}{2} (\bmod\ q) = (x \gg$

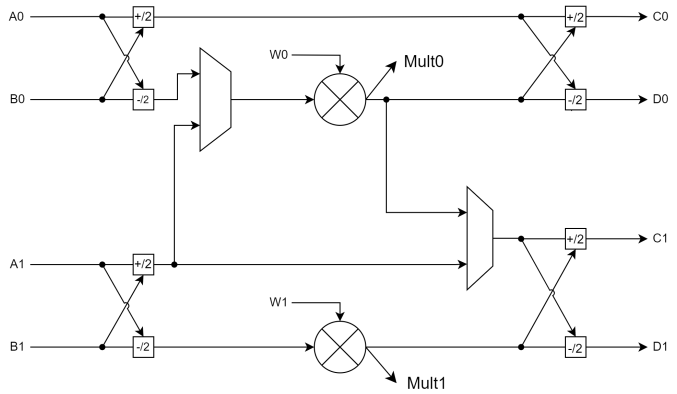


Figure 4: Architecture of a Double Butterfly Unit. Control signals of the multiplexers are omitted.

$1) + x[0] \cdot (\frac{q+1}{2})$. This operation is easily merged with a modular adder or subtractor, as shown by [LTHW23]. The latency of a merged modular adder/subtractor is 1 clock cycle.

We implemented the modular multiplier as usual 12×12 bit multiplication followed by a modified Barrett reduction, proposed by [XL21] in their NTT module for Kyber. Our implementation of this modular multiplication has a latency of 1 clock cycle for the multiplication and 4 clock cycles for the reduction.

For executing the PWM in hardware architectures, it is common to apply a Karatsuba-like reduction to this operation [XL21, AMI+23], in order to reduce the number of multiplications from 5 to 4, as shown in (11).

$$\begin{aligned} \hat{h}_{2i} &= \hat{f}_{2i} \cdot \hat{g}_{2i} + \hat{f}_{2i+1} \cdot \hat{g}_{2i+1} \cdot \omega^{2br(i)+1} \\ \hat{h}_{2i+1} &= (\hat{f}_{2i} + \hat{f}_{2i+1}) \cdot (\hat{g}_{2i} + \hat{g}_{2i+1}) - (\hat{f}_{2i} \cdot \hat{g}_{2i} + \hat{f}_{2i+1} \cdot \hat{g}_{2i+1}) \end{aligned} \tag{11}$$

However, we notice that there are 2 consecutive multiplications. Therefore, there is the need of cascading the DBUs for performing a pipelined PWM operation, as done by [AMI+23]. The first DBU executes the multiplications $m_0 = \hat{f}_{2i} \cdot \hat{g}_{2i}$ and $m_1 = \hat{f}_{2i+1} \cdot \hat{g}_{2i+1}$. Then, between the two DBUs, modular adders execute the operations $s_0 = \hat{f}_{2i} + \hat{f}_{2i+1}$, $s_1 = \hat{g}_{2i} + \hat{g}_{2i+1}$ and $s_2 = -m_0 - m_1$. Finally, the last DBU calculates $\hat{h}_{2i} = m_0 + m_1 \cdot \omega^{2br(i)+1}$ and $\hat{h}_{2i+1} = s_2 + s_0 \cdot s_1$.

3.5 Memories and memory access schemes

In the following section, we will provide more details on the memory organization.

Because of the varying parallelism levels of a butterfly execution, we decided to implement the memory access scheme from [MRW+22]. For a given array of processing elements with width w_{PE}, they propose the usage of $2 \times w_{PE}$ banks of memory along with a coefficient index generation algorithm. Their algorithm generates the coefficient indices in every stage s in two phases with two different arithmetic progressions. In the first one, the arithmetic progression between coefficients has a common difference of 2^{7-s}, whereas the second phase has a common difference of $N/2w_{PE}$. The authors formally proof that such a scheme is collision-free (reading two or more coefficients from a single memory bank) and free of data-dependency between stages (read-after-write), if $c_{PE} \leq \frac{n}{2w_{PE}}$, where c_{PE} is the number of clock cycles necessary for calculating a butterfly from the coefficient index generation until the storage in memory. We refer the reader to [MRW+22] for the full formal proof.

Then, the bank and address of each coefficient is calculated with the equations (12), where idx is the index of the coefficient, $b = 2 \times w_{PE}$ is the number of banks and d_i is each of the digits of idx in base b representation.

$$\text{bank} = \left(\sum_{i=0}^{L-1} d_i \right) \bmod b, \ \text{addr} = \left\lfloor \frac{idx}{b} \right\rfloor \tag{12}$$

In our accelerator, w_{PE} varies from 1 to 4, depending on the version of the butterflies executed. If NTT is unprotected, then $w_{PE} = 4$. Therefore we implement 8 banks of RAM with depth of 32 addresses each, avoiding memory collision issues in all of the possible processing element configurations. The mapping of the coefficient indexes to every bank of memory is shown in Figure 5. Additionally, taking in consideration all the different cases of parallelization and NTT configurations the worst case for pipeline latency is when $w_{PE} = 4$. Therefore, for this architecture the pipeline latency must be $c_{PE} \leq 16$, to guarantee the nonexistence of read-after-write collision issues. We designed the architecture with $c_{PE} = 13$.

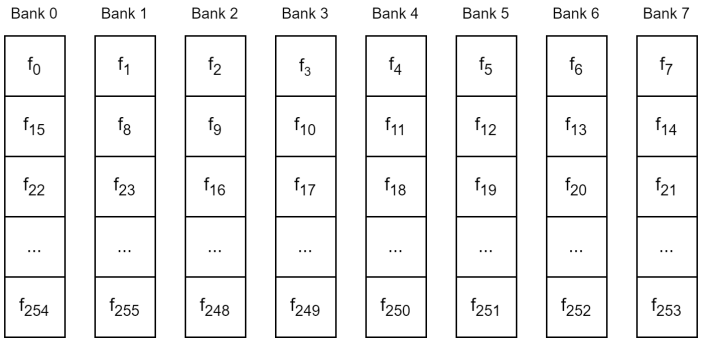


Figure 5: Mapping of each coefficient index to all banks of memory. The scheme from [MRW+22] guarantees avoidance of memory collisions.

Finally, the width of the primary memory is expanded from 12 to 20 bits. This expansion facilitates the storage, within the same memory address, of the current power of each coefficient's mask, if local masking is used. In this way, we mutualize read and write logic for the data of the masking unit and the processing unit.

A second memory is necessary for storing the second polynomial used in the PWM operation. This memory is a lot simpler since PWM was not designed with a flexible parallelism like NTT/INTT. The PWM operation requires two coefficients per clock cycle. Therefore, the secondary memory consists of one single bank of memory, with a width of 24 bits and a depth of 128 addresses, for storing two coefficients at each address. A single port memory is used, simplifying the logic needed for accessing the data.

For easiness of computation, twiddle factors in embedded accelerators are typically precalculated and stored in a read-only memory. The powers of the twiddle factors are often related and grouped within a single address. However, because of the usage of local masking, the powers of the twiddle factors may not be related at all and therefore replication of data is needed. In fact, in this accelerator with four multipliers, it is entirely possible that four unrelated twiddle factors are needed. Consequently, four read-only memories are employed, each with the 256 possible twiddle factors.

3.6 Masking unit

The masking unit is the module in charge of randomizing the twiddle factors and effectively applying the local-masking countermeasure. It is equipped with a dual port memory with

128 addresses of 8 bits that store the randomness for the masked twiddle factors and four arithmetic units for calculating the new powers, as shown in Figure 6. It requires one clock cycle for loading per address of randomness The arithmetic units perform the masking/demasking of the powers of the twiddle factors in mod n, as shown in 2.5 and according to previous masks and new masks loaded in its internal memory. If no masking is performed, then the powers of the twiddle factors are untouched.

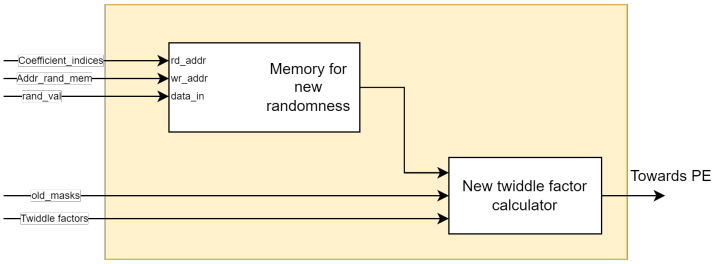


Figure 6: Diagram of the masking unit.

3.7 Implementation results

In this subsection, we show the implementation results of the proposed architecture. First, we demonstrate the timing results in terms of clock cycles, which are independent of the platform used. Then, we showcase the resource utilization for implementations on both a Xilinx Artix-7 FPGA and an ASIC, using the 22nm FDSOI library from Global Foundries.

3.7.1 Timing

The timing results of the execution of NTT and INTT are shown in Table 2. The latency depends directly on the number of masks that is desired for the execution and on the parameter `out_in_unmasked` for the PWM protection. In general, the latency is composed of processing time, mask-loading time and pipeline latency, which is 13 clock cycles for both NTT and INTT. The time needed to write and read the accelerator memories from the external world is not considered in this analysis. The additional overhead of the PWM protection mode is due to more requirements of randomness for leaving the output masked or using more costly butterflies to ensure all values are correctly masked. For example, executing the PWM-protected NTT necessitates an additional layer of randomness. Consequently, more time is required to load this additional randomness. Also, the PWM-protected INTT with $u = 128$ starts with a MDIDO butterfly, instead of MSISO, which incurs in additional time penalty.

For PWM, the latency is 148 clock cycles, composed of 128 cycles for processing and 20 cycles for the pipeline latency. This latency is higher in this operation because of the cascading of the butterfly units, explained in Section 3.4.

The acceleration factor in terms of clock cycles compared to the ARM Cortex M4 microcontroller implementation of the NTT for Kyber in [RPBC20] is reported in Table 3. This comparison is provided for the NTT/INTT without PWM protection and for the comparable number of masks u reported. As seen, the acceleration factor is between 95x-213x.

3.7.2 Resource utilization

This architecture was implemented for two different hardware targets: FPGA and ASIC.

For the FPGA target, the chosen chip was the Xilinx Artix-7 XC7A100t-3. The synthesis and implementation was done with Vivado 2021.1 with default synthesis and

implementation options. The NTT module uses 3651 LUT, 1430 FF, 6 BRAM, 4 DSP for a maximum frequency of 167.37 MHz. The critical path is between the coefficient index generated by the control unit through the bank decoding logic, ending at the input of the memories.

In Table 4, we compare the results of our implementation with FPGA unprotected NTT designs for Kyber with 4 DSPs in order to have comparable numbers. It must be noted that the comparison can be unfair since there is not an implementation with local-masking reported in the state of the art. The metric for comparison is the Area-Time Product (ATP) which we define here as $ATP = (LUTs + FFs) * 1/f_{max} * CCs$, where f_{max} is the maximum frequency in MHz of the implementation and CCs is the number of clock cycles required for performing an unprotected NTT operation. It is evident that the implementation involves additional overhead compared to those selected in the state of the art. There are several elements in the architecture that avoid optimizations or add area utilization. For starters, the configurable level of protection in terms of number of masks u is achieved with a parallelization flexibility in terms of number of effective PE in a butterfly calculation. Therefore, we chose a memory scheme that allow for this variable parallelism [MRW+22]. This incurs in overheads in the memory blocks used, coefficient index generation and configuration signals coming from the control unit. Additionally, the masking unit represents a module that is not present in any of the cited implementations, that contains a memory block and arithmetic units for the calculation of the updated twiddle factors. Finally, the processing element itself is built to support masked Cooley-Tukey and Gentleman-Sande butterflies, which adds additional complexity compared to a PE not equipped with this capabilities.

The design was also synthesized for an ASIC target. The technology node used was the 22nm FDSOI library from Global Foundries.

Apart from the synthesis of the general logic into standard cells, three different types of memories were compiled. For the 8 banks of memories for the primary memory, a register-file-based memory was used, with 20 bits of width and 5 bits of depth. For the secondary memory, a single-port SRAM memory was generated with a width of 24 bits and 7 bits of depth. Finally, for the memory of the masking unit, a dual-port SRAM memory was compiled, with 8 bits of depth and 7 bits of width. The total area of the memories is 20045.89 μm^2 and this is independent of the target frequency of the general simulation.

For a maximum frequency of 1.199 GHz, the total area of the device is 31793 μm^2, yielding 11147.11 μm^2 for the rest of the logic without memories. This corresponds to 159 kGE, using the NAND2 area of the standard cell library as a reference. The critical path corresponds to the internal generated memories. The standard cells used are of type RVT.

Table 2: Timing results for NTT and INTT. In parenthesis, the percentage of the overhead in clock cycles compared to the unmasked case ($u = 0$).

Number of masks u	Clock cycles without PWM protection (NTT/INTT)		Clock cycles with PWM protection (NTT/INTT)	
0	237/237	−/−	−/−	−/−
1	467/467	(97.04/97.04)	468/467	(97.47/97.04)
2	537/537	(126.58/126.58)	539/537	(127.43/126.58)
4	613/613	(158.65/158.65)	617/613	(160.34/158.65)
8	701/701	(195.78/195.78)	709/701	(199.16/195.78)
16	813/813	(243.04/243.04)	829/813	(249.79/243.04)
32	973/973	(310.55/310.55)	1005/973	(324.05/310.55)
64	1229/1229	(418.57/418.57)	1293/1229	(445.57/418.57)
128	1613/1613	(580.59/580.59)	1805/1677	(661.60/607.59)

Table 3: Acceleration of local-masked NTT/INTT for CRYSTALS-Kyber compared to the software implementation from [RPBC20]

Number of masks u	Clock cycles of software implementation from [RPBC20] for NTT/INTT	Acceleration factor in clock cycles of *this work* for NTT/INTT
0	31×10^3 / 50.6×10^3	130.8/213.50
1	44.6×10^3 / 63.9×10^3	95.50/136.83
2	66.5×10^3 / 83.7×10^3	123.84/155.87
4	72.1×10^3 / 87.2×10^3	117.62/142.25

Table 4: FPGA resource comparison with state of the art. PE layout refers to the width and depth of the butterfly array.

Implementation	PE layout	LUT	FF	BRAM	DSP	f_{max}	CCs	ATP
This work	4x1	3651	1430	6	4	167	237	7197
[LTHW23]	4x1	1170	1164	2	4	303	235	1810
[YMÖS21]	4x1	2543	792	9	4	182	232	4251
[BNAMK21]	2x2	801	717	2	4	222	324	2215
[IUH22]	2x2	904	811	2.5	4	216	268	2127

4 Weaknesses in NTT countermeasure

The countermeasure introduced in [RPBC20] is at its essence, a multiplicative masking. As such, we have identified three main vulnerabilites that may undermine its effectiveness for SASCA and side-channel attacks in general. Firstly, there is a variability in the runtime depending on parameters of local masking. Secondly, multiplicative masking is ineffective when dealing with 0-values. Finally, we identify that this masking scheme lacks surjectiveness and therefore it may leak information.

4.1 Variable runtime depending of parameters

The numbers of masks determines the type of butterflies that are used, therefore the algorithm runs in variable time. This in itself does not reveal any secret information to the attacker through timing. However, the information on the configuration of the algorithm is essential for the attacker to build the factor graph for the attack.

4.2 The 0-value problem

The 0-value problem is a well known downside of multiplicative masking schemes. Zeros cannot be masked with multiplications. The first multiplicative masking scheme on AES, proposed by Akkar and Giraud [AG01], was shown by Golić and Tymen [GT03] to be vulnerable to first-order zero-value DPA because of this very same reason.

Such characteristic can be leveraged by a SASCA attacker on the [RPBC20] countermeasure in two ways:

- First way is as follows: suppose a usual NTT butterfly $c = a + b \cdot \omega^i,\ d = a - b \cdot \omega^i$. If the attacker can successfully identify zeros in any layer of the NTT, they can know that the inputs of the previous butterfly are related as $b \cdot \omega^i = q - a$ or $a = b \cdot \omega^i$ depending if the zero was detected on c or d. This information is not hidden by the multiplicative masking on any parameter for the [RPBC20] countermeasure.

  This is specially true for the case of an NTT execution that is not previously additive-masked. If this is the case, averaging several traces can reveal the presence of 0's,

simplifying the graph for the attack and reducing the entropy of it by revealing the relationship between the previous coefficients.

In the case when the execution of the NTT is additive-masked, the averaging cannot be done.

- Second way is by applying the strategies proposed by Hamburg *et al.* [HHP+21] for attacking the inverse NTT and the long term key. In their work, they craft chosen ciphertexts to force the result of PWM to zero in certain coefficients. This is to reduce the size of the graph, reducing the entropy of some variables to zero. Such strategy can also be applied here to attack an implementation of the inverse NTT equipped with the [RPBC20] countermeasure, even in the additive-masked case.

As 0-values are not masked, then this effect should be possible to be seen in a t-test of an implementation with the [RPBC20] countermeasure. If the leakage of such implementation can be modeled with the noisy hamming weight model, the detection of such effect could be resilient to a high level of noise.

4.3 Non-surjective masking

Consider the set of Twiddle factors $W = \{\zeta^i \bmod q | i \in [0, n-1]\}$. The masking of each coefficient in the [RPBC20] countermeasure follows the map $m : \mathbb{Z}_q \times W \to \mathbb{Z}_q$. For a fixed coefficient $x \in \mathbb{Z}_q$, the mapping becomes non-surjective. This means that for a fixed x, there is only n possible values to which it can be mapped. This non-uniform mask reveals information about the fixed coefficient x.

For a fixed $x \in \mathbb{Z}_q$, the set of possible values resulting from the map is $\{x \cdot \zeta^i \bmod q | i \in [0, n-1]\}$. Let $y \in \mathbb{Z}_q$ be a member of this set such that $y \equiv x \cdot \zeta^i \mod q$. Then, $x \equiv y \cdot \zeta^{-i} \mod q$. And because ζ is an n-th root of unity, $x \equiv y \cdot \zeta^{n-i} \mod q$. Therefore, x can also be generated from the map from y, with $j = n - i$ and is inside the set $\{y \cdot \zeta^j \bmod q | j \in [0, n-1]\}$. As every set possible contains n values, then the number of unique sets is $(q-1)/n$, which in Kyber is 13. We will denote each of these sets with W_k, where k is the smallest value that multiplies the possible twiddle factors and generates the sets such that $\{k \cdot \zeta^i | i \in [0, n-1]\}$, and $k \in [1, 2, 3, 4, 5, 8, 9, 10, 11, 15, 20, 25, 31]$.

If an attacker can identify the ocurrence of one set in the mapping, then it reduces the number of possible coefficient values from 3329 to 256.

An attacker could gather several traces until it narrows the possible cases. For example, if they can distinguish individual values, then a set is identified and the number of candidates is reduced to 256. However, if the attacker is only equipped with a Hamming weight distinguisher, then it becomes more complicated. In Table 5, the ocurrence of Hamming weights for each of the sets W_k is shown. As seen, some of the Hamming weights are impossible to obtain for certain sets. For example, if a Hamming weight of value 11 is detected, then the only possible sets are W_2 and W_4. Likewise, a detection of a Hamming weight of 1 discards the set W_{20}. Another strategy could be of obtaining several traces and observing the distribution of Hamming weights. Then, a maximum likelihood strategy could be employed to select the correct set.

It must be noted that means of the Hamming weights of each of the sets W_k, shown in Table 5 are not the same as the mean of the Hamming weight of a random variable, which is 5.69149895. If a local-masked NTT implementation leaks the Hamming weight, the effect of the non-surjectiveness should be possible to be seen in a t-test, although it should be more difficult to be seen than the previous vulnerability since the averages of Hamming weights are very close and it should be very susceptible to noise.

Additionally, we argue that the MSISO butterfly is inherently more prone to leaks than the MDIDO butterfly. This is because the set of intermediate values and every set of outputs will be masked with the same mask, giving only n possible mappings of all these

Table 5: Hamming weight frequencies for the values in the sets W_k, along with the weighted means of Hamming weights

Set W_k \ HW	1	2	3	4	5	6	7	8	9	10	11	Mean of HW
W_1	1	7	21	38	47	62	48	20	11	1	0	5.6328125
W_2	1	7	19	35	49	58	54	21	8	3	1	5.71875
W_3	1	3	17	34	64	61	44	23	6	3	0	5.6875
W_4	1	6	15	41	56	55	47	23	10	1	1	5.6875
W_5	1	3	21	35	60	47	57	23	8	1	0	5.69140625
W_8	1	6	12	30	60	54	51	28	14	0	0	5.859375
W_9	1	8	9	36	61	67	44	22	8	0	0	5.66796875
W_{10}	1	3	23	39	53	57	48	22	9	1	0	5.6328125
W_{11}	1	6	10	36	56	67	40	29	10	1	0	5.78515625
W_{15}	1	7	12	39	57	68	42	22	7	1	0	5.63671875
W_{20}	0	4	23	41	50	59	43	26	6	4	0	5.65625
W_{25}	1	4	17	41	61	65	35	25	3	4	0	5.58984375
W_{31}	1	2	20	33	54	64	43	29	8	2	0	5.765625

sets. In an implementation with parallelization, this could increase the signal to noise ratio for a non-specific t-test. On the other hand, the MDIDO butterfly provides many more possible mappings. For the set of four multiplications that need to be done, as shown in (8), it provides n^4 possible mappings and n^2 for the outputs of the butterfly. Therefore the noise expected for this operation is bigger, and leakage could be harder to find, specially for an implementation with parallelization as the one presented in Section 3.

5 Leakage assessment on FPGA

In this section, we describe the leakage asseessment campaign for an FPGA programmed with the implementation of [RPBC20] countermeasure discussed in Section 3. The method for this assessment is a non-specific t-test, as described first in [GJJR11] and more formally in [SM15]. A t-test is used to assess if whether the difference of a moment of two sets is significant or not. In the case of the first statistical moment, it tests the hypothesis that the means of two groups can be considered as different. The t-statistic is defined in (13), where μ_0 (resp. μ_1) is the mean, s_0^2 (resp. s_1^2) is the sample variance and n_0 (resp. n_1) is the cardinality of set 0 (resp. 1) .

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}} \tag{13}$$

For the side-channel scenario, the two samples come from an execution with a fixed input and another one with a random input, *i.e.* a *non-specific t-test*. In the literature, it is an empiric standard to consider that an implementation is leaking information if $|t| > 4.5$, since, for $n > 5000$ the confidence of rejecting the null hypothesis that the two sets have the same moment is $99,999\%$ [GJJR11, SM15].

5.1 Setup and methodology

The evaluation bench consists of a Tektronix MSO64 oscilloscope, a Femto HSA-X-2-40 low-noise amplifier and a Langer RF-U 5-2 EM probe. The evaluation board used is a Digilent Basys-3 with a Xilinx Artix-7 XC7A35TCPG236 FPGA chip programmed with

the implementation described in Section 3. The sample rate of the oscilloscope was fixed to 625 MS/s, and the FPGA uses the built-in clock of 100 MHz.

To perform the leakage assessment for our implementation, we configure the accelerator with $u = 1$ and with $u = 128$, according to the number of masks per stage as explained in Section 2.5. We chose this cases, because they are the extreme cases for our implementation, *i.e.* the NTT execution is masked with the least and most masks per round possible, respectively. Additionally, $u = 1$, uses only MSISO butterflies, while $u = 128$ uses only MDIDO butterflies for the intermediate layers (that are not input or output layers). This allows us to draw some evidence about the hypothesis that the non-surjectiveness vulnerability should be easier to detect in MSISO butterflies compared to MDIDO ones. For each scenario we first perform a control experiment, by feeding 0 in the randomness port to the masking unit, which means that the operation is always masked with a twiddle factor of $\zeta^0 = 1$.

5.2 Results

Case 1: u**=1**. The results of the control experiment are shown in Figure 7, using 20k traces for the fixed and random input executions. The seven stages of the NTT execution are highlighted in different colors. They overlap because of the pipeline slack in the architecture. As seen in the figure, the leakage is present throughout the execution without the multiplicative masking.

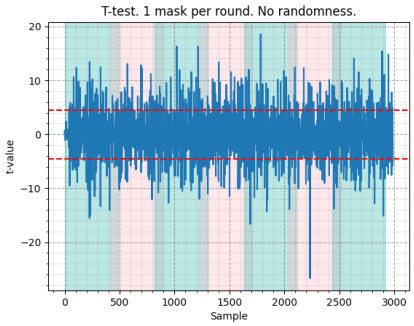
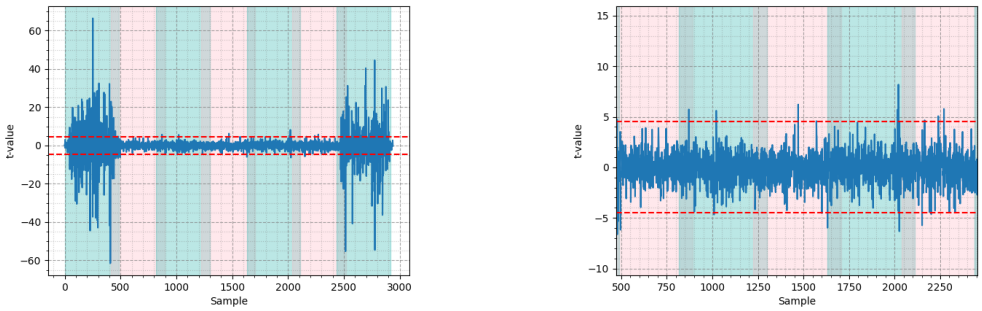


Figure 7: Control experiment for case 1, $u = 1$. 20k traces.

For the next experiment, we chose a fixed input polynomial which would yield 0 as intermediate value at a certain point, to test the hypothesis of Section 4.2 that an implementation would leak the 0-value. The results are shown in Figure 8, with a number of traces per sample set of 500k. It can be seen that the t-value of the samples of all the stages is reduced, except for the first and the last stage. This is expected, since the input and the output of the NTT operation is unmasked. In Figure 8b we zoom on the stages 1 to 5. The fixed polynomial that was inputted to the device yielded zeros in stages 3 and 5, so it was expected that t-values surpassing 4.5 would appear in these stages. However, there are surpassing t-values that also appear in other stages. We make the conjecture that the contribution to the leakage could also be coming from the non-surjectiveness explained in Section 4.3.

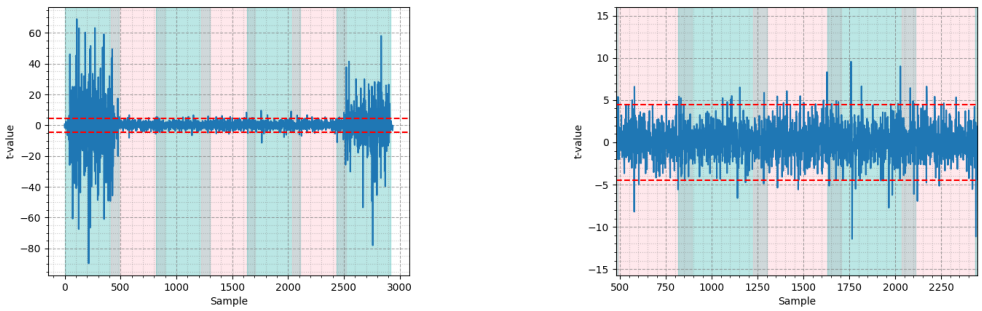
To test the hypothesis that leakage can come from the non-surjectiveness vulnerability, we choose a fixed polynomial that does not yield any zero in an intermediate value. In this experiment, 1 million traces per set were obtained. The results are shown in Figure 9. As seen in the zoomed Figure 9b, there is leakage detected in every stage. We make the conjecture that this leakage is coming from the non-surjectiveness vulnerability.

Case 2: $u = 128$. The t-test of the control experiment is shown in Figure 10, using



(a) Full t-test. (b) Zoom on stages 1-5.

Figure 8: T-test for case 1, $u = 1$ using a polynomial that yields intermediate values with zero. 500k traces.



(a) Full t-test. (b) Zoom on stages 1-5.

Figure 9: T-test for case 1, $u = 1$ using a polynomial that does not yield intermediate values with zero. 1 million traces.

30k traces for each of the sets. Again, the leakage is shown to be present throughout the whole execution. In this case, the stages do not overlap since before every stage (except for the last one that demasks the output), there is a loading phase of the masks of 128 clock cycles, one for each mask.

Then, the test for non-surjectiveness is done by feeding a polynomial that will not yield a zero as intermediate value. In this experiment, we used around 1 million traces per set and the results are shown in Figure 11. As seen in the figure, no leakage is actually detected in the intermediate rounds. This is because of the expected increased noise of using MDIDO butterflies, as explained in Section 4.3.

Finally, to test for the zero-value problem, we feed a polynomial that will yield a zero in round 4. With 336k traces, the result is shown in Figure 12. As seen in Figure 12b, leakage is detected in round 5, since the zero coming from round 4 is an input for round 5. As the non-surjectiveness seems to be more dificult to detect with $u = 128$ than with $u = 1$ because of the usage of MDIDO butterflies, we make the conjecture that this leakage comes from the zero-value issue.

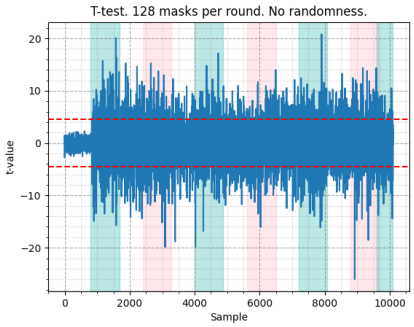
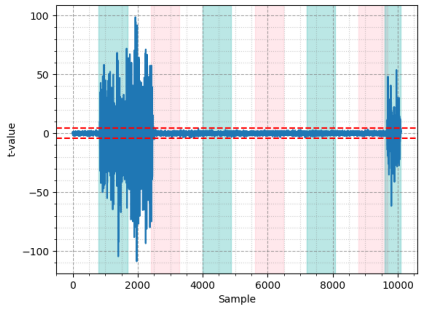
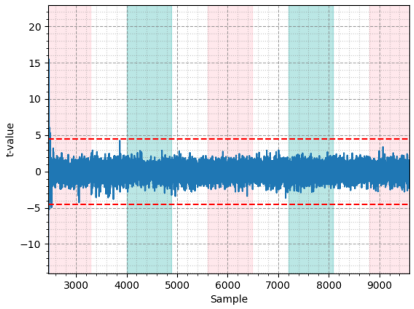


Figure 10: Control experiment for case 2, $u = 128$. 30k traces.



(a) Full t-test.



(b) Zoom on stages 1-5.

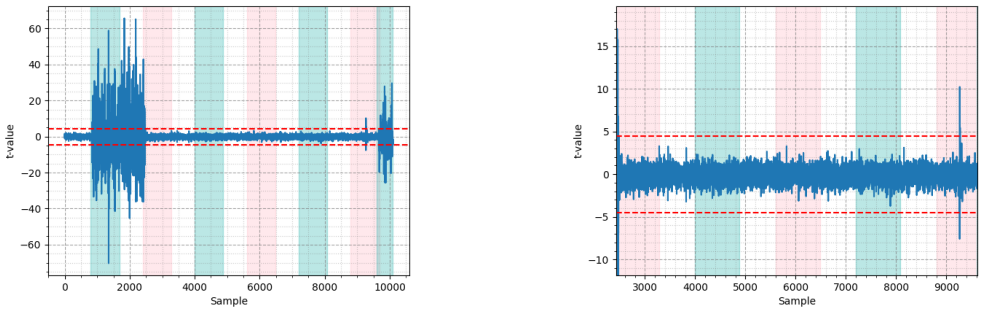
Figure 11: T-test for case 2, $u = 128$ using a polynomial that does not yield intermediate values with zero. 1 million traces. No leakage detected in intermediate rounds.

6 Conclusion

In this paper we presented an analysis of surface, performance and security of the masking countermeasure for the NTT against soft-analytical side-channel attacks presented in [RPBC20]. We first introduced a hardware implementation of a local-masked NTT, capable of executing different levels of protection according to the number of masks per stage u, with the goal of making an exploration on area, performance and a posterior security analysis. We showed the possible overhead incurred in implementing such a countermeasure and the challenges behind ensuring maximum utilization of the modular multipliers. Then, we described some of the possible vulnerabilities that are inherent to the masking countermeasure. Finally, we performed a leakage assessment on an FPGA board programmed with our proposed implementation to support our hypothesis of inherent leakage of the local masking scheme.

Even if we argue that an implementation with this type of masking inherently leaks information, it must be said that it does augment the level of noise of an NTT implementation and the entropy on individual intermediate values. As seen in the t-tests, the t-values are significantly reduced, while still failing, specially for the MSISO case. If used, this countermeasure should be used mainly with MDIDO butterflies and in conjunction with usual additive masking to avoid attacks that improve the attacker knowledge, such as the averaging for the zero value explained in Section 4.2 or a maximum-likelihood strategy, as showed in Section 4.3.

As future work, we leave the evaluation of a practical SASCA attack on a local-masked



(a) Full t-test. (b) Zoom on stages 1-5.

Figure 12: T-test for case 2, $u = 128$ using a polynomial that yields intermediate values with zero in round 4. 336k traces. Leakage detected in round 5.

NTT taking advantage of the possible vulnerabilities identified in this paper. Additionally, the effect of extending this local masking countermeasure to protect the PWM, as shown in Section 3.3, can be studied in terms of security and its integration to the whole protocol of CRYSTALS-Kyber. Moreover, a comparative study can be undertaken between local masking and shuffling to analyze the security, area and performance aspects.

References

[ABD+21] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation, 2021.

[AG01] Mehdi-Laurent Akkar and Christophe Giraud. An Implementation of DES and AES, Secure against Some Attacks. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, Lecture Notes in Computer Science, pages 309–318, Berlin, Heidelberg, 2001. Springer.

[AMI+23] Aikata Aikata, Ahmet Can Mert, Malik Imran, Samuel Pagliarini, and Sujoy Sinha Roy. KaLi: A Crystal for Post-Quantum Security Using Kyber and Dilithium. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(2):747–758, February 2023. Conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems (CHES)*, pages 16–29. Springer Berlin Heidelberg, 2004.

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine Van Vredendaal. Masking Kyber: First- and Higher-Order Implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 173–214, August 2021. Publisher: Universitatsbibliothek der Ruhr-Universitat Bochum.

[BNAMK21] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. High-Speed NTT-based Polynomial Multiplication Accelerator for Post-

Quantum Cryptography. In *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*. IEEE, June 2021.

[CBVB22] Rafael Carrera Rodriguez, Florent Bruguier, Emanuele Valea, and Pascal Benoit. Correlation Electromagnetic Analysis on an FPGA Implementation of CRYSTALS-Kyber. Cryptology ePrint Archive, Paper 2022/1361, 2022. https://eprint.iacr.org/2022/1361.

[CMJ22] Zhaohui Chen, Yuan Ma, and Jiwu Jing. Low-Cost Shuffling Countermeasures against Side-Channel Attacks for NTT-based Post-Quantum Cryptography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2022.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, 2003.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 537–554. Springer Berlin Heidelberg, 1999.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatg. A Testing Methodology for Side–Channel Resistance Validation, 2011. Published: NIST Non-Invasive Attack Testing Workshop.

[GT03] Jovan D. Golić and Christophe Tymen. Multiplicative Masking and Power Analysis of AES. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, Lecture Notes in Computer Science, pages 198–212, Berlin, Heidelberg, 2003. Springer.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine Van Vredendaal. Chosen Ciphertext k-Trace Attacks on Masked CCA2 Secure Kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 88–113, August 2021.

[HSST22] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. Adapting Belief Propagation to Counter Shuffling of NTTs, October 2022. Version Number: 20221013:132543 Published: Cryptology ePrint Archive, Paper 2022/555.

[IUH22] Yuma Itabashi, Rei Ueno, and Naofumi Homma. Efficient Modular Polynomial Multiplier for NTT Accelerator of Crystals-Kyber. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 528–533, August 2022. ISSN: 2771-2508.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Advances in Cryptology — CRYPTO' 99*, pages 388–397. Springer Berlin Heidelberg, 1999.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, Berlin, Heidelberg, 2010. Springer.

[LS15] Adeline Langlois and Damien Stehlé. Worst-Case to Average-Case Reductions for Module Lattices. *Des. Codes Cryptography*, 75(3):565–599, June 2015.

[LTHW23] Minghao Li, Jing Tian, Xiao Hu, and Zhongfeng Wang. Reconfigurable and High-Efficiency Polynomial Multiplication Accelerator for CRYSTALS-Kyber. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(8):2540–2551, August 2023.

[MRW+22] Jianan Mu, Yi Ren, Wen Wang, Yizhong Hu, Shuai Chen, Chip-Hong Chang, Junfeng Fan, Jing Ye, Yuan Cao, Huawei Li, and Xiaowei Li. Scalable and Conflict-free NTT Hardware Accelerator Design: Methodology, Proof and Implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2022.

[MWK+22] Catinca Mujdei, Lennert Wouters, Angshuman Karmakar, Arthur Beckers, Jose Maria Bermudo Mera, and Ingrid Verbauwhede. Side-Channel Analysis of Lattice-Based Post-Quantum Cryptography: Exploiting Polynomial Multiplication. *ACM Transactions on Embedded Computing Systems*, November 2022.

[Nat] National Institute of Standards and Technology. Post-Quantum Cryptography Standardization Process.

[PP19] Peter Pessl and Robert Primas. More Practical Single-Trace Attacks on the Number Theoretic Transform. In *Progress in Cryptology – LATINCRYPT 2019*, pages 130–149. Springer International Publishing, 2019.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. In *Lecture Notes in Computer Science*, pages 513–533. Springer International Publishing, 2017.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing - STOC '05*. ACM Press, 2005.

[RPBC20] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On Configurable SCA Countermeasures Against Single Trace Attacks for the NTT. In *Security, Privacy, and Applied Cryptography Engineering*, pages 123–146. Springer International Publishing, 2020.

[RRVV15] Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A Masked Ring-LWE Implementation. In *Lecture Notes in Computer Science*, pages 683–702. Springer Berlin Heidelberg, 2015.

[Saa17] Markku-Juhani O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering*, 8(1):71–84, January 2017. Publisher: Springer Science and Business Media LLC.

[SM15] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - a clear roadmap for side-channel evaluations, 2015. Publication info: A minor revision of an IACR publication in CHES 2015.

[VCGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft Analytical Side-Channel Attacks. In *Lecture Notes in Computer Science*, pages 282–296. Springer Berlin Heidelberg, 2014.

[XL21] Yufei Xing and Shuguo Li. A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 328–356, February 2021.

[YMÖS21] Ferhat Yaman, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1020–1025, February 2021. ISSN: 1558-1101.

[ZBT19] Timo Zijlstra, Karim Bigou, and Arnaud Tisserand. FPGA Implementation and Comparison of Protections Against SCAs for RLWE. In *Lecture Notes in Computer Science*, pages 535–555. Springer International Publishing, 2019.

[ZYC+20] Neng Zhang, Bohan Yang, Chen Chen, Shouyi Yin, Shaojun Wei, and Leibo Liu. Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 49–72, March 2020. Publisher: Universitatsbibliothek der Ruhr-Universitat Bochum.