Combined Fault and DPA Protection for Lattice-Based Cryptography

Daniel Heinz[1] and Thomas Pöppelmann[2]

[1] Research Institute CODE, Universität der Bundeswehr München, Germany
Daniel.Heinz@unibw.de
[2] Infineon Technologies AG, Germany
Thomas.Poeppelmann@infineon.com

Abstract. The progress on constructing quantum computers and the ongoing standardization of post-quantum cryptography (PQC) have led to the development and refinement of promising new digital signature schemes and key encapsulation mechanisms (KEM). Especially lattice-based schemes have gained some popularity in the research community, presumably due to acceptable key, ciphertext, and signature sizes as well as good performance results and cryptographic strength. However, in some practical applications like smart cards, it is also crucial to secure cryptographic implementations against side-channel and fault attacks. In this work, we analyze the so-called redundant number representation (RNR) that can be used to counter side-channel attacks. We show how to avoid security issues with the RNR due to unexpected de-randomization and we apply it to the Kyber KEM and show that the RNR has a very low overhead. We then verify the RNR methodology by practical experiments, using the non-specific t-test methodology and the ChipWhisperer platform. Furthermore, we present a novel countermeasure against fault attacks based on the Chinese remainder theorem (CRT). On an ARM Cortex-M4, our implementation of the RNR and fault countermeasure offers better performance than masking and redundant calculation. Our methods thus have the potential to expand the toolbox of a defender implementing lattice-based cryptography with protection against two common physical attacks.

Keywords: Lattice-Based Cryptography · Module-LWE · Kyber · Side-Channel Attacks · ARM Cortex-M

1 Introduction

The importance of post-quantum cryptography (PQC) has significantly grown over the past couple of years with Shor's polynomial-time algorithm for prime factorization [Sho94], advances in the construction of quantum computers [Mos18], and the ongoing NIST PQC standardization process [Nat16]. Some promising families of cryptosystems currently in round 3 of the NIST process [NIS20] are based on the hardness of certain lattice problems. An advantage of lattice-based cryptography is that it allows constructing public-key encryption (PKE) and digital signatures at the same time with certain similarities in their structure. In addition, implementations of lattice-based schemes in the NIST process have proven to be quite efficient on embedded devices [KRSS20], with reasonable public key and ciphertext or signature sizes.

Even though reference implementations already claim to have constant or secret independent timing behavior (e.g., see [ABD+20b, JKL+, AAB+20]), this is not sufficient in a setting where an attacker may gain full control over a device, e.g., in smart cards for payment, digital identification documents, or digital signatures. In such scenarios, there is

a need for appropriate side-channel and fault attack protection to prevent long-term secrets from getting compromised [MOP07]. Thus, the development of efficient countermeasures against physical attacks, which take into account the constraints of embedded devices, is a prerequisite before PQC can be deployed in the aforementioned use-cases.

A good overview of already existing implementations of PQC is given in [NDR+19] and a good summary of attacks and countermeasures can be found in [TE15]. Recently, works like [RRdC+16, RdCR+16, OSPG18, BDK+20, BPO+20] have focused on the side-channel protection of components or of the whole decryption operation of lattice-based schemes that are based on the ring learning with errors (RLWE), ring learning with rounding (RLWR) or the modular versions of these assumptions (MLWE/MLWR). The implementation strategy on microcontrollers and reconfigurable hardware is usually to employ arithmetic masking to protect arithmetic operations in the polynomial ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, a masked decoder, sampling of noise polynomials using a masked sampler [SPOG19], and a masked comparison [BPO+20].

However, an interesting question is whether alternatives to straightforward arithmetic masking can be developed to appropriately protect computations in R_q and how such alternatives perform in comparison to masking in terms of security and efficiency. Note that most countermeasures to protect elliptic curve cryptography (ECC) [FGM+10, FV12], like scalar randomization, random projective coordinates, and base point blinding, are quite efficient and exploit some underlying mathematical structure of the elliptic curve for efficiency. Besides, some implementations that use masking are still susceptible to single-trace attacks under certain conditions [PP19, PPM17] and would require further protection as [RPBC20]. Such new techniques could then either be combined with masking to increase the security level or used as an alternative due to better performance.

An approach that may complement but not fully replace masking is polynomial blinding [Saa18] in which two polynomials $\mathbf{f}, \mathbf{g} \in R_q$ are multiplied by a random integer $a \in \mathbb{Z}_q$ such that $(a\mathbf{f}) \cdot (a^{-1}\mathbf{g})) = \mathbf{f} \cdot \mathbf{g}$. Another approach is shuffling, which was evaluated in [Pes16] for the context of Gaussian sampling. In addition, Zijlstra, Bigou, and Tisserand [ZBT19] recently proposed the usage of the redundant number representation (RNR). The core idea is that one randomizes coefficients $c \in \mathbb{Z}_q$ in R_q by adding a random value $r \cdot q$ where $r \in [0, 2^k)$ for some integer k. Computations are then carried out $\bmod(2^k q)$ and the final result is obtained by reducing mod q. The FPGA implementation of the countermeasure described in [ZBT19] showed reasonable overhead and promising security properties in simulations. However, the redundant representation method may also be appealing on a microcontroller where coefficients of typical lattice-based KEMs are in the range of 12 bits (for Kyber [ABD+20b] in round 2/3), 13 bits (for Saber [DKRV19]) to 14 bits (for Newhope [AAB+20]) and thus do not fill a 16 or 32-bit register.

Besides side-channel attacks, it is also important to consider fault attacks, e.g., laser fault injection, as an attacker might choose the path of least resistance or may even combine semi-invasive and non-invasive attacks. In previous work, it was shown that attacks on the control flow of reference implementations of lattice-based KEMs and signature schemes or even simple faults in polynomial arithmetic can lead to powerful attacks [BBK16, EFGT18, BP18, RJH+19, VOGR18]. In contrast to a large number of attacks, an implementer currently has only a few effective tools and concepts available to counter such attacks. For example, in [HKM+19] the authors propose noise samplers with built-in fault protection. A generic method against simple fault attacks is double computation and a comparison of both results. However, when combining masking with double computations the performance overhead may become too high.

As a consequence, it currently seems to be an open question, how to efficiently combine fault and side-channel countermeasures for arithmetic operations in lattice-based schemes, while achieving low computational overhead. Moreover, a suitable fault countermeasure should indicate faults in the arithmetic (e.g., a bit flip in a coefficient of a polynomial in

R_q) but also faults in the control flow, e.g., where a fault onto a pointer leads to a mix-up or exchange of a secret-key polynomial for an attacker-controlled value.

Contribution. In this work, we provide methods for the protection of arithmetic operations[1] in lattice-based cryptography against side-channel and fault attacks. We evaluate the effectiveness of the redundant number representation (RNR) from [ZBT19] and show that a naive instantiation of the RNR is unsafe. One caveat is the canceling of randomization due to constant values and the other is the full canceling of randomization due to adversarially chosen inputs. We then implement the updated RNR approach on an ARM Cortex-M4 microcontroller and we show how it can be applied to fast software implementations of Kyber using a state-of-the-art 32-bit NTT[2]. Our RNR-protected NTT implementation achieves a low overhead with only 7 737 cycles compared to 6 829 cycles for an unprotected NTT. In addition, we perform practical side-channel evaluations using the t-test methodology to verify the approach. Furthermore, we present a novel method to detect fault-attacks in lattice-based cryptography using the Chinese Remainder Theorem (CRT) and evaluate it for common parameter sets of lattice-based cryptography (e.g., Kyber and Dilithium) and against different realistic fault models. For the first time we then show how the fault countermeasure can be combined with the RNR for the linear parts of Kyber decryption with lower overhead than masking and redundant calculation to counter both side-channel and fault attacks. This is non-trivial as a careful parameter selection for RNR and CRT is required to still be able to use the NTT for fast computation.

2 Preliminaries

In this section, we introduce notation for lattice-based cryptography, the Kyber scheme, and previous work on fault and side-channel protection of implementations of lattice-based cryptography.

2.1 Notation

For $x \in \mathbb{R}$, we write $\lceil x \rfloor$ to mean the closest integer to x (where $\lceil y + \frac{1}{2} \rfloor := y + 1$ for $y \in \mathbb{Z}$). For $x \in \mathbb{R}$, define $\lceil x \rfloor = \lfloor x + \frac{1}{2} \rfloor \in \mathbb{Z}$. Let $\mathbb{Z}_q$ denote the quotient ring $\mathbb{Z}/q\mathbb{Z}$ for an integer $q > 1$. Thus, $\mathbb{Z}_q$ is the ring of cosets $x + q\mathbb{Z}$ with addition and multiplication operations. For $a, b \in \mathbb{Z}$, we write $a \bmod^{(+)} b$ for the unique integer $\hat{a} \equiv a \bmod b$ such that $0 \leq \hat{a} < b$. Let $R = \mathbb{Z}[X]/(f)$, where f is usually $f = X^n + 1$ for n being a power of 2, and $R_q = R/(q) = \mathbb{Z}_q[X]/(f)$ for some positive integer q. Any element $\mathbf{a} \in R_q$ as well as vectors of these elements are denoted as bold lower case letter. We use the notation $\mathbf{a}[i]$ for $i = 0, \ldots, n-1$ to access the i-th coefficient of $\mathbf{a}$. Matrices of elements in R_q are denoted as bold upper case letters. For a given set S and a probability distribution D over S, we use $s \xleftarrow{r} D$ to mean $s \in S$ sampled according to D using coins r. In addition, we use $s \xleftarrow{\$} S$ to mean $s \in S$ sampled uniformly at random from S. Hereby, $U(q)$ denotes the uniform distribution on R_q, whereas χ denotes an error distribution to be defined for the specific algorithm. When covering our implementation we define the $x \bmod q$ operation for integers x, q to always produce an output in the range $[0, q-1]$. Unless stated otherwise, when we access an element $\mathbf{a}[i]$ of a polynomial $\mathbf{a} \in R_q$, we always assume that $\mathbf{a}[i]$ is reduced modulo q and in the range $[0, q-1]$.

[1] Of course, re-encryption, sampling, and decoding have to be appropriately protected in practice [OSPG18, BDK+20]. However, they are out of the scope of this work and require different techniques, like the ones in [SPOG19, BPO+20, RRVV15].

[2] After publication of our work we will make the source code of our adapted NTT implementation available to allow independent validation of our results.

2.2 The NTT

Designers of lattice-based schemes can use the number-theoretic transform (NTT) to reduce the computational cost of polynomial multiplications (see [Für09]). Exemplary, the NIST round 3 finalists Dilithium [DKL+18] and Kyber [ABD+20b] as well as the round 2 scheme NewHope [AAB+20] have NTT-friedly parameter sets or even incorporate the NTT into the definition of the scheme. The NTT enables relatively efficient and simple to implement polynomial multiplication for suitably parameterized rings R_q. The product of two polynomials $\mathbf{a}, \mathbf{b} \in R_q$ can be computed as $\mathbf{a} \cdot \mathbf{b} = \text{INTT}(\text{NTT}(\mathbf{a}) \circ \text{NTT}(\mathbf{b}))$, where $\circ$ denotes coefficient-wise multiplication. However, a straightforward application of the NTT to R_q and avoidance of any zero padding requires the existence of a $2n$-th root of unity γ in R_q, which is the square root of the n-th root of unity ω. This holds for R_q when n is a power of 2 and q is a prime such that $q \equiv 1 \mod 2n$.

For a polynomial $\mathbf{g} = \sum_{i=0}^{n-1} \mathbf{g}[i] X^i \in R_q$ we define $\text{NTT}(\mathbf{g}) = \hat{\mathbf{g}} = \sum_{i=0}^{n-1} \hat{\mathbf{g}}[i] X^i$ with $\hat{\mathbf{g}}[i] = \sum_{j=0}^{n-1} \gamma^j \mathbf{g}[j] \omega^{ij} \bmod q$ where ω is an n-th primitive root of unity and $\gamma = \sqrt{\omega} \bmod q$. The inverse function INTT is defined as $\text{INTT}(\hat{\mathbf{g}}) = \mathbf{g} = \sum_{i=0}^{n-1} g_i X^i$ with $\mathbf{g}[i] = \left(n^{-1} \gamma^{-i} \sum_{j=0}^{n-1} \hat{\mathbf{g}}[j] \omega^{-ij} \right) \bmod q$ [AAB+20]. In comparison to aforementioned NTT-enabled schemes, some submissions to the NIST standardization process explicitly avoid the NTT, e.g., due to setting q as a power of 2 [DKRV19] or by avoiding the existence of roots of unity by choosing $f = X^p - X - 1$ [BBC+20]. As a side note, the NTT may be used by an implementer to speedup schemes like Saber [CHK+20] that were not designed with NTT-friendly parameters.

2.3 Kyber

The KEM Kyber is currently in the third round of the NIST PQC standardization process [ABD+20b, BDK+18]. Kyber's security is based on the hardness of solving the learning-with-errors (LWE) problem in module lattice (see [LS15]). To achieve semantic security with respect to an adaptive chosen ciphertext attack (CCA), Kyber internally uses an IND-CPA secured public-key encryption (PKE) scheme and applies a variation of the Fujisaki-Okamoto (FO) transform [HHK17, FO99]. Kyber instantiates the ring R_q with the polynomial ring $\mathbb{Z}_q[X]/(X^n + 1)$ with $n = 256$ and scales its security by using the module structure.

2.3.1 Changes to the Kyber NTT in Round 2

In this work, we always refer to the third round version of Kyber. However, we would like to highlight a tweak introduced in round 2 of the NIST process [ABD+20a] that has an impact on the exact realization of the NTT.

Kyber updated the definition of the NTT to allow the choice of a smaller modulus $q = 3329$ ($n = 256$ stays the same). This leads to smaller ciphertexts and public keys and enables the usage of a smaller noise distribution, which in turn reduces the amount of required pseudorandom bits.

Definition 1 (Number-Theoretic Transform of Kyber)**.** *Let $\mathbb{Z}_q$ be a finite field and ζ be a primitive n-th root of unity in $\mathbb{Z}_q$. Then the NTT of a vector $\mathbf{x} \in R_q$ computes a vector $\mathbf{y} \in R_q$ via the map*

$$\mathbf{y}[2k] = \sum_{j=0}^{n/2-1} \mathbf{x}[2j] \zeta^{(2br_7(i)+1)j}, k \in \{0, ..., n/2-1\}$$

$$\mathbf{y}[2k+1] = \sum_{j=0}^{n/2-1} \mathbf{x}[2j+1] \zeta^{(2br_7(i)+1)j}, k \in \{0, ..., n/2-1\}$$

Hereby, $br_7(i)$ denotes the bitreversed number of a seven-bit integer. The inverse NTT (INTT) is given by inverting both parts of the NTT individually as in Section 2.2.

Let $\zeta = 17$ be the first primitive 256th root of unity in the case of Kyber. Then the equality

$$X^{256} + 1 = \prod_{i=0}^{127}(X^2 - \zeta^{2i+1}) = \prod_{i=0}^{127}(X^2 - \zeta^{2br_7(i)+1}) \tag{1}$$

from [ABD+20b] holds. The Chinese remainder theorem (CRT) provides an isomorphism such that the polynomial multiplication of degree 256 can be performed in 127 polynomial multiplications of degree two. If not explicitly stated otherwise, all usages of the NTT from now on refer to the updated definition of the NTT for Kyber introduced in round 2 and kept in round 3.

2.3.2 Simplified Kyber

For later reference we provide a simplified[3] version of the public-key encryption scheme Kyber.CPA = (Kyber.CPA.Gen, Kyber.CPA.Enc, Kyber.CPA.Dec) as in Algorithms 1, 2 and 3. Define integers $n = 256, q = 3329, \eta_2 = 2$ and let k, η_1, d_t, d_u, d_v be positive integers. We denote $\mathcal{M} = \{0,1\}^n$ as the plaintext space, where each message $m \in \mathcal{M}$ can be seen as a polynomial in R with coefficients in $\{0,1\}$. Define the functions

$$\begin{aligned} \text{Compress}_q(x,d) &:= \lceil (2^d/q) \cdot x \rfloor \bmod^{(+)} 2^d, \\ \text{Decompress}_q(x,d) &:= \lceil (q/2^d) \cdot x \rfloor. \end{aligned}$$

We set χ_η as the centered binomial distribution with support $\{-\eta, \ldots, \eta\}$, and let $\chi_{n,\eta}$ be the distribution of polynomials of degree n with entries independently sampled from χ_η. When we apply the NTT to a vector of polynomials, the NTT gets applied to each polynomial individually.

$(\rho, \sigma) \xleftarrow{\$} \{0,1\}^{256} \times \{0,1\}^{256}$;
$\mathbf{A} \xleftarrow{\rho} U(q)^{k \times k}$;
$(\mathbf{s}, \mathbf{e}) \xleftarrow{\sigma} \chi_{n,\eta_1}^k \times \chi_{n,\eta_1}^k$;
$\hat{\mathbf{s}} \leftarrow \text{NTT}(\mathbf{s})$;
$\hat{\mathbf{e}} \leftarrow \text{NTT}(\mathbf{e})$;
$\hat{\mathbf{t}} \leftarrow \mathbf{A} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$;
return $pk_{CPA} := (\hat{\mathbf{t}}, \rho)$, $sk_{CPA} := \hat{\mathbf{s}}$;

Algorithm 1: Kyber.CPA.Gen.

Input: $pk_{CPA} = (\hat{\mathbf{t}}, \rho)$
Input: $m \in \mathcal{M}$
Input: $r \xleftarrow{\$} \{0,1\}^{256}$
$\mathbf{A} \xleftarrow{\rho} U(q)^{k \times k}$;
$(\mathbf{r}, \mathbf{e}_1, \mathbf{e_2}) \xleftarrow{r} \chi_{n,\eta_1}^k \times \chi_{n,\eta_2}^k \times \chi_{n,\eta_2}$;
$\hat{\mathbf{r}} \leftarrow \text{NTT}(\mathbf{r})$;
$\mathbf{u} \leftarrow \text{INTT}(\mathbf{A} \circ \hat{\mathbf{r}}) + \mathbf{e}_1$;
$\mathbf{v} \leftarrow \text{INTT}(\hat{\mathbf{t}} \circ \hat{\mathbf{r}}) + \mathbf{e_2} + \lceil \frac{q}{2} \rfloor \cdot m)$;
$c_1 \leftarrow \text{Compress}_q(\mathbf{u}, d_u)$;
$c_2 \leftarrow \text{Compress}_q(\mathbf{v}, d_v)$;
return $c := (c_1, c_2)$;

Algorithm 2: Kyber.CPA.Enc.

Input: $sk_{CPA} = \hat{\mathbf{s}}$
Input: $c = (\mathbf{u}, \mathbf{v})$
$\mathbf{u} \leftarrow \text{Decompress}_q(\mathbf{u}, d_u)$;
$\mathbf{v} \leftarrow \text{Decompress}_q(\mathbf{v}, d_v)$;
return $m = \mathit{Compress}_q(\mathbf{v} - \mathit{INTT}(\hat{\mathbf{s}} \circ \mathit{NTT}(\mathbf{u})), 1)$;

Algorithm 3: Kyber.CPA.Dec.

[3] Sampling procedures and encoding into byte arrays has been simplified as these operations are of less importance to our work.

2.4 Side-Channels in Lattice-Based Cryptography

Techniques like simple power analysis (SPA) [Koc96] and differential power analysis (DPA) [KJJ99] use statistical methods to recover a secret key from a power trace or other leakage information [MOP07]. They can also be applied to implementations of lattice-based cryptography. Recent works that describe attacks on lattice-based cryptography or its building-blocks are [BFM+18, ATT+18, KPP20, RRCB20, RBRC20, RPBC20, ACLZ20, GJN20].

A common countermeasure against DPA attacks is the randomized splitting of secret information into two (or more) arithmetic or Boolean shares, which is usually called masking [CJRR99]. Before the RLWE/MLWE schemes became popular, previous work aiming at the protection of lattice-based cryptography mainly focused on NTRU, e.g., [ABGV08, WZW13, MKSDG10, ZWW13]. Reparaz, Sinha Roy, Vercauteren, and Verbauwhede then proposed the first masked implementation of an RLWE-based scheme in [RRVV15] and analyzed its security against first-order attacks. They focus on the CPA-secured decryption operation, which requires the computation of $m = \text{Decode}(\mathbf{c}_1\mathbf{r}_2 + \mathbf{c}_2)$ for ciphertexts $\mathbf{c}_1, \mathbf{c}_2 \in R_q$, secret key $\mathbf{r}_2 \in R_q$, and a decoding function Decode to obtain the message $m \in \{0,1\}^n$ (the scheme works similarly to Line 3 of Algorithm 3 for $k = 1$). A very sensitive operation that is prone to DPA or SPA attacks is the computation of $\mathbf{c}_1\mathbf{r}_2$ in which an attacker controlled polynomial $\mathbf{c}_1$ is multiplied by the fixed secret polynomial $\mathbf{r}_2$. The strategy in [RRVV15] is to split the secret key $\mathbf{r}_2$ into two shares and to compute $\mathbf{r}_2 \cdot \mathbf{c}_1$ independently on both shares. Then, $\mathbf{c}_2$ is added to one share and both shares are processed by a masked decoder. The optimization or analysis of such decoders for various parameters (besides other improvements) has been addressed in works like [OSPG18, RRdC+16, RdCR+16]. However, in [OSPG18] it is argued that the protection of the CPA-secured decryption routine of common RLWE/MLWE schemes is not sufficient. In a common DPA scenario, the attacker is assumed to have full control over a device and is thus supposed to be able to craft arbitrary ciphertext that can be sent to a device, which contains the secret key for decryption (see [BDH+19] for some so-called misuse attacks). As a consequence, it is necessary to implement also a CCA conversion and to prevent an attacker from carrying out a misuse attack with information obtained from side-channel leakages. The first-order secured implementation of the RLWE-based encryption scheme from [OSPG18], which is similar to NewHope, thus additionally masks the re-encryption mandated by the FO transformation. This includes polynomial arithmetic but also the sampling of error vectors. Following a similar approach, a first-order secured masked implementation of the medium-security version of Saber [BDK+20] was recently provided. Saber does not rely on the NTT and uses a rounding assumption, which allows the improved performance of building blocks in comparison to [OSPG18]. To achieve protection against higher-order attacks, recent works have applied higher-order masking to building blocks like a binomial sampler [SPOG19] or the masked comparison in the FO transform [BPO+20]. In addition, recent works show how to protect lattice-based signature schemes with masking [BBE+18, MGTF19, GR19]. In general, it appears that masking is very straightforward to apply to the linear parts of lattice-based algorithms. Thus, masking schemes protecting the arithmetic of the secret key operation in RLWE/MLWE-based lattice-based cryptography appear to be considerably simpler than masking used to protect highly non-linear functions in symmetric cryptography (see [RBN+15]).

2.5 Fault Attacks on Lattice-Based Cryptography

When injecting faults into a device that is computing a cryptographic algorithm, it may be possible to derive information on a secret key from the behavior or output of the device. Popular countermeasures are redundant computation, duplication, checksums, or sensor-based countermeasures such as (laser) light detection or supply voltage monitoring

(see the survey in [BCN+06, VKS11]). In practice, usually, a combination of different countermeasures is applied to achieve reliable fault protection. Examples of very effective fault attacks are the Boneh-DeMillo-Lipton Fault Attack on RSA-CRT [BDL01] or the recovery of a DSA key as shown in [BCN+06]. A fault can be injected by various means, e.g., using a laser, varying voltage, or manipulating an external clock to force glitches (see [BCN+06]). As it may not be very obvious at first that a manipulated output can be used to compromise secret information, it often appear to be worth to aim for detection of any manipulated computation.

Some previous fault attacks on lattice-based cryptography succeed by using faults to skip necessary validity checks in unprotected implementations or the sampling of random elements [BBK16, EFGT18]. In a similar vein, loop-abort faults [EFGT18] can be used to partially skip the generation of random elements to obtain weak instances of the underlying lattice problem. The application of such attacks on lattice-based KEMs is described in [VOGR18]. Other attacks target constant values, e.g., the domain separation in hash functions and PRNGs of implementations of KEMs [RJH+19]. More complex analysis of faulted data, e.g., using lattice-reduction, also allows fault attacks without the requirement to affect a single bit precisely [BP18]. Further refinements of previous attacks and experimental results for fault attacks on signature schemes in pqm4 [KRSS20] were given in [RJH+19].

3 Redundant Number Representation

In this section, we analyze the applicability of the redundant number representation to lattice-based schemes and show unexpected pitfalls and security issues for some parameter choices. In their work Zijlstra, Bigou, and Tisserand [ZBT19] propose to randomize a coefficient $c \in \mathbb{Z}_q$ by adding $r \cdot q$ to it, where $r \in [0, 2^k)$ is a random number for some integer k. The integer k denotes the number of bits of randomness. Then, all arithmetic operations are performed in $\mathbb{Z}_{2^k q}$. Therefore, in each execution, all adders, multipliers, and decoders are handling randomized inputs. The redundant number representation countermeasure is then analyzed by a simulated correlation power analysis (CPA) attack for different redundancy levels, i.e. values of $k = 0$ to $k = 8$. Moreover, it is shown that the overhead in an FPGA implementation is low.

In this work, we use q' to denote the multiple of the modulus q, which is $q' = 2^k$ in [ZBT19]. Thus, when randomization is applied, all operations are performed mod $q'q$ and constants or intermediate values are randomized by adding rq with random $r \in [0, q')$. Finally, to remove the randomization all values are reduced modulo q. To obtain a fast and side-channel secured Kyber decryption (Kyber.CPA.Dec), the RNR can be applied during the computation of $\text{INTT}(\hat{\mathbf{s}} \circ \text{NTT}(\mathbf{u}))$. In addition, it can be used to protect the linear parts of the Kyber re-encryption (Kyber.CPA.Enc). However, two issues arise in a typical Kyber microcontroller implementation where the RNR should be used to protect also the NTT. One is related to the incompatibility of a power-of-two q' with typical NTT implementations and the other with de-randomization caused by NTT constants or an attacker.

3.1 Enabling of Fast Montgomery Reduction

An important techniques for efficient lattice-based cryptography in finite rings with prime modulus is the usage of Montgomery reduction and representation for fast modular reduction [Mon85]. The technique was first applied to a fast software implementation of NewHope [ADPS16] where all precomputed constants used for the NTT are stored in Montgomery representation and is widely used in other implementations as well. However, this technique requires a Montgomery constant M of the form $M = 2^z$ for an integer z. In

addition, this constant has to be relatively prime to the modulus of the finite ring. Hence, the usage of $q' = 2^k$ for $R_{q'q}$ is excluded. A straightforward solution is to use an odd integer q' for the ring extension method to make Montgomery reduction work efficiently with the RNR.

3.2 The NTT and (Adversarial) De-Randomization

To avoid unforeseen pitfalls when using the RNR method, a theoretic analysis of the algebraic properties of the ring extension is required on top of the evaluation in [ZBT19]. Hereby, we focus on the case where q is a prime number. This parameter choice has been made by the NIST finalist schemes Kyber [ABD+20b], Dilithium [DKL+20], and Falcon [JKL+]. Note that the finalists Saber [DKRV19] and NTRU [CDH+20] select q as a power of two. However, for these schemes, the RNR is not applicable, as for any q equal to a power of two integer the value rq is equal to a bit shift to the left of r and does not impact lower-order bits when added to a constant or intermediate value.

Now we recall a basic algebraic theorem concerning finite rings.

Theorem 1. *Given a finite ring R, any element $a \in R$ is either a unit in the ring, e.g. there is an element $b \in R$ with $ab = 1_R$, or a zero divisor, e.g. there exists an element $b \in R$ with $b \neq 0_R$ and $ab = 0_R$.*

In the case of a prime modulus q, the ring R_q is a finite field, e.g. R_q does not contain any zero divisors. By using the RNR, the commutative ring $R_{q'q}$ satisfies every property of a field except for the existence of inverse elements. As stated in Theorem 1, the existence of zero divisors in the ring $R_{q'q}$ is given. Since the proposed method is based on arithmetic operations, this poses the question if the randomness can be removed by multiplying with a specific number. Indeed, during the NTT in the Kyber decryption, some precomputed powers of ζ in Montgomery representation (see Section 2.3.1) are multiplied with secret intermediate values. The powers of ζ are fixed and could eliminate randomness during every execution of the decryption. This may enable SPA or DPA attacks. For instance, for the choice of $q' = 257$, the multiplication with $\zeta^{71} \cdot M \bmod q = 1799 = 7 \cdot 257$ (including the Montgomery constant $M = 2^{16}$) removes the randomness part during every decryption. For a coefficient c randomized by addition of rq for $r \in [0, q')$ it holds that

$$\begin{aligned} t &= (c + r \cdot q) \cdot (\zeta^{71} \cdot M \bmod q) && \bmod q'q \\ &= c \cdot 1799 + r \cdot (1799)q && \bmod q'q \\ &= c \cdot 1799 + r \cdot (7 \cdot 257)q && \bmod q'q \\ &= c \cdot 1799 + (r \cdot 7)q'q && \bmod q'q \\ &= c \cdot 1799 && \bmod q'q. \end{aligned}$$

To avoid processing any non-randomized values, the parameter choice q' can be adjusted such that no constants contain a multiple of q'. Additionally, an adversary might also use this approach to remove the randomization during a side-channel attack, e.g., when attacking an implementation as presented in [ZBT19]. For Kyber, an attacker could create a ciphertext $\mathbf{u}$ such that coefficients of $\hat{\mathbf{u}} = \mathrm{NTT}(\mathbf{u})$ contain q' as a factor. The coefficient-wise multiplication of the secret $\hat{\mathbf{s}}$ by such a $\hat{\mathbf{u}}$ would then trigger the removal of the randomization of $\hat{\mathbf{s}}$ and might then cause Hamming weights leakage of the result of the coefficient-wise multiplication or allow attacks on the INTT operation. Thus, randomization of the input ciphertext coefficients is recommended. Additionally, the option to randomize the twiddle factors, i.e. the powers of ζ, for every execution exists. However, the second option is less favorable since it does not protect the coefficient-wise multiplication against malicious ciphertexts.

3.3 Validation in Hamming Weight Leakage Model

We now verify the effectiveness of the RNR in the commonly used Hamming weight leakage model [MD99, KJJ99] for different choices of q and q'. Hereby, we assume that an adversary can observe a leakage $\mathrm{L}(x) = \mathrm{HW}(x) + \mathcal{N}$, where $\mathcal{N}$ denotes an additive noise with mean $\mu = 0$ and $\mathrm{HW}(x)$ the Hamming weight of x. We simulate an attack for an adversary obtaining $\mathrm{L}(s' \cdot c \mod (qq'))$ with zero noise, where a fixed secret $s \in \mathbb{Z}_q$ is protected as $s' = s + r \cdot q \mod (qq')$ for a random $r \in \mathbb{Z}_{q'}$ and a public and changing coefficient c. This resembles a DPA attack on the decapsulation operation in Kyber where a secret key $\mathbf{s}$ is multiplied coefficient-wise with ciphertext coefficients $\mathrm{NTT}(\mathbf{u})$ controlled by the attacker. We simulate $N = 100$ traces for a fixed secret $s \in \mathbb{Z}_q$ and varying coefficients c and randomness r. We then make use of the maximum-likelihood method [Sch06] that calculates the probability of occurrence for every possible value $\hat{s}$ based on the simulated leakage. This is an expensive yet very detailed brute-force approach commonly used [CAB19, CS21]. We then choose the $\hat{s}$ that makes the observations most probable (commonly called the maximum likelihood estimate). If the chosen s is correct, i.e. $\hat{s} = s$, the attack was a success.

First, we test the impact of the size of q' in combination with q. In Figure 1, we plot the success rate using maximum-likelihood estimation (MLE) depending on the size of q' and for different values of q. For each tested q' we select 1000 different secrets s' and compute the success rate, i.e., cases where the MLE correctly predicted $\hat{s} = s$. Note that the different values of q have a slightly different binary representation, including

- $q = 8192 = 10000000000000_2$ (Saber)
- $q = 12289 = 11000000000001_2$ (NewHope)
- $q = 3329 = 110100000001_2$ (Kyber)

to capture the behavior of this method for different lattice-based schemes. As expected, the RNR does not work well with the power-of-two modulus of Saber. Moreover, the success rate for the Kyber modulus is smaller than for NewHope. Note that this experiment is in line with correlation power simulations in the Hamming weight leakage model already performed in [ZBT19]. However, they only tested for an uneven q' and a prime number q.

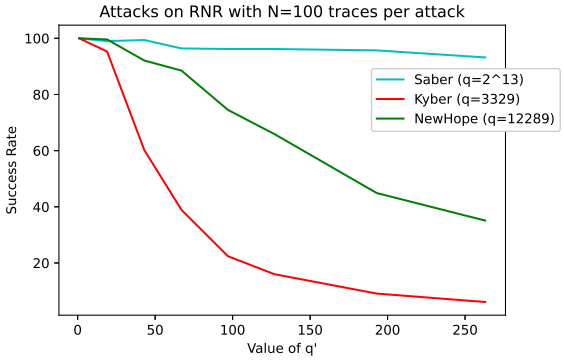
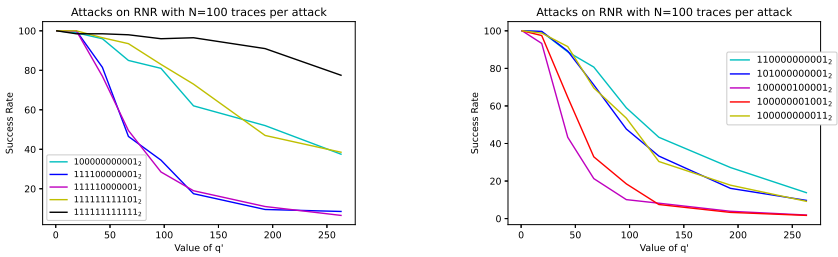


Figure 1: Comparison of success rates using maximum-likelihood estimation on the Hamming weight of the result $\mathrm{L}(s' \cdot c \mod (qq'))$ for different lattice-based schemes.

As shown in Figure 1, we observed a different success rate for the RNR for the NewHope and Kyber modulus. Thus, we ran experiments for different moduli shown in Figure 2. From the experimental results we draw the following conclusions:



(a) Success Rates for different Hamming weights.

(b) Success Rates for different binary structures with the same Hamming weight.

Figure 2: Success rates for different settings with respect to q.

- Randomization of lower order bits: In order to randomize any value $x \in [0, q)$ properly, the least significant bit (LSB) should be one. Similar to the case where q is a power of two, this prevents an early bit shift to the left. Hence, the least significant bits of the sensitive value are always affected by randomization.
- Hamming weight of q: Not only a too sparse structure of q but also too many bits equal to one can impede the effectiveness of this method as shown in Figure 2a. This is because $q = 111111111111_2$, for instance, can be written as $q = 2^{13} - 1$. This reduces the number of affected bits by randomization as the addition of $r \cdot 2^{13}$ does not have an impact on the 12 lower order bits.
- Structure of q in binary representation: As can be seen in Figure 2b, not only the Hamming weight impacts the effectiveness of this method. Figure 2b shows the results of the experiment for values of q with Hamming weight three. The values of q which do not randomize every bit have significantly higher success rates. For instance, the choice of $q = 2051 = 100000000011_2$ will always leave the second most significant bit unrandomized when a q' with at most 8 bits is selected.

As a consequence, values of q that are less optimal in the context of the RNR need a larger q' and thus larger randomness $r \in [0, q')$ to increase entropy.

As mentioned in Section 3.2, the NTT itself consists of multiplications with fixed values. The multiplication of a randomized coefficient with a known value should also be randomly distributed. The pitfall of derandomization is not covered in the analysis of [ZBT19]. It can be captured by a slightly modified version of the maximum-likelihood attack for random but known coefficients c. In this version, we simulate a number of N traces of the multiplication with a specific power of ζ. Let s be the correct subkey guess. We can observe the Hamming weight HW_{obs} of the result of the multiplication $(s + r \cdot q) \cdot \zeta^i$ using the Hamming weight leakage model. Then, for every possible subkey $\hat{s}$ the probability $\mathrm{P}(\mathrm{HW}((\hat{s} + r \cdot q) \cdot \zeta^i) = HW_{obs})$ is calculated. We repeat this process for each trace and sum up the probabilities for each $\hat{s}$. If the randomization has failed for the specific power of ζ and q', the correct subkey guess is among the most likely ones, e.g.

$$\max_{\hat{s}} \sum_{j=1}^{N} \mathrm{P}(\mathrm{HW}((\hat{s} + r_j \cdot q) \cdot \zeta^i) = HW_{obs}) = \sum_{j=1}^{N} \mathrm{P}(\mathrm{HW}((s + r_j \cdot q) \cdot \zeta^i) = HW_{obs}).$$

This is then a successful attack as the adversary can narrow down the subkey possibilities to all $\hat{s}$ with $\mathrm{HW}(\hat{s} \cdot \zeta^i) = HW_{obs}$. We show the success rates for 100 attacks on every multiplication with a power of ζ used in the NTT of Kyber in Figure 3 for $q = 3329$ different parameters $q' = 257$ compared to $q' = 263$ and the number of $N = 100$ traces.

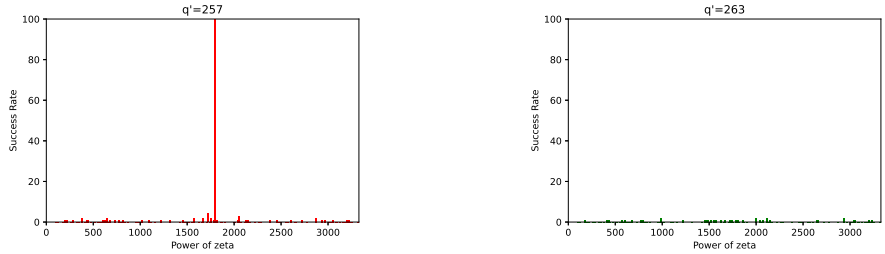


Figure 3: Success rates (correct subkey guess being among the most likely ones) of the maximum-likelihood attack on multiplication with a constant containing q' as factor (left) and a parameter set where this is not the case (right).

4 Fault Protection with Chinese Remainder Theorem

As discussed in Section 2.5, several fault attacks on lattice-based schemes have already been proposed and only a few works on effective countermeasures exist. Common countermeasures are redundant calculations, checksums, or sensor-based countermeasures such as light detection or supply voltage monitoring (see [BCN+06, KSV13, VKS11]). As some of these techniques require a high computational overhead or changes to a device (e.g., light sensors), they are often not cost-effective.

In this work, we propose a new technique for lattice-based cryptography. When applied properly, it can be used to detect fault attacks on coefficients in data structures representing polynomials in R_q in memory or during computations and also offers some protection of the control flow. Basically, in our countermeasure we use the Chinese remainder theorem (CRT) to combine elements of a lattice-based cryptosystem (e.g., secret keys, public elements) in a ring modulo q with constants in a ring modulo q' (as q' has a similar effect as in Section 3, we use the same notation). A function g consisting of operations like polynomial addition, multiplication (including NTTs and coefficient-wise multiplication) is then carried out over the combined ring modulo $\hat{q} = qq'$. After a sequence of computations is finished, the two rings are split again and the results obtained in the ring modulo q' are compared to predetermined checkvalues. The checkvalue is obtained by computing g on the constants in the ring modulo q'. The intuition is, that faults introduced during the modulo $\hat{q}$ computations will also disturb the final result obtained in the ring modulo q'. Our method is related to CRT codes [GRS99], the Residue Number System used for ECC [PFPB19] and Shamir's Countermeasure for RSA and its analog method for elliptic curves in [Joy19]. However, we do not aim at error correction and work on fundamentally different data structures and constraints imposed by ring parameters. We would also like to note that an attacker might use any fault detection method, like the one provided, to carry out safe-error attacks (see [FGM+10] for such attacks on ECC) or other attacks that exploit the presence of an error detection mechanism. A countermeasure that makes some safe-error attacks harder, is randomization of the processed data. A topic that we consider in Section 5 in combination with fault protection.

4.1 Formal Description

More formally, given the factor ring $R_q = \mathbb{Z}_q[X]/(f)$ for a suitable f, e.g., $f = X^n + 1$, we introduce two new factor rings $R_{q'} = \mathbb{Z}_{q'}[X]/(f)$ and $R_{\hat{q}} = \mathbb{Z}_{\hat{q}}[X]/(f)$ with q being relatively prime to q' and $q \cdot q' = \hat{q}$. In this case the CRT yields an isomorphism

$$\mathbb{Z}_{\hat{q}}[X]/(f) \cong \mathbb{Z}_{q'}[X]/(f) \times \mathbb{Z}_q[X]/(f) \tag{2}$$

in which coefficients $\mathbf{r}[j]$ of $\mathbf{r} \in R_{q'}$ are associated with coefficients $\mathbf{v}[j]$ of input data $\mathbf{v} \in R_q$ for $j = 1, ..., n$.

We can achieve fault protection when carrying out a function $g_R : R^l \to R^k$ with l ring elements from a ring R as inputs that produces k ring elements from R as output. As an example, the function $\mathbf{t}_1 = g_{R_q}(\mathbf{a}, \mathbf{e}, \mathbf{s}) = \mathbf{a} \cdot \mathbf{e} + \mathbf{s}$ computes a sample from the RLWE distribution [LPR13] with $l = 3$ inputs and one output ($k = 1$) in R_q.

Our countermeasure is initialized by fixing suitable values for n, q, q', f that define the rings $R_q, R_{q'}, R_{\hat{q}}$ and by fixing a function g that then implies values of l and k. Without our countermeasure implemented, the device would compute g_{R_q} on input values $\mathbf{v}_i$ for $i = 1, ..., l$ (e.g., decryption or encryption in Kyber). The first step (see Figure 4) is performed offline before the device is deployed. In this step constants $\mathbf{r}_1, ..., \mathbf{r}_l \in R_{q'}$ are sampled from a suitable distribution, like the uniform distribution or using a deterministic pseudo-random generator (PRG) based on a seed. The checkvalues $\mathbf{t}_1, ..., \mathbf{t}_k$ are computed as $g_{R_{q'}}(\mathbf{r}_1, ..., \mathbf{r}_l) = (\mathbf{t}_1, ..., \mathbf{t}_k)$ and the constants (or seeds) and checkvalues are then stored in the device's ROM or non-volatile memory (NVM).

In the online phase, the attacker has access to the device. The function g is protected by combining coefficients of the $\mathbf{v}_i$'s component-wise with the respective coefficients of the $\mathbf{r}_i$'s as

$$\mathbf{z}_i[j] = (\mathbf{r}_i[j] \cdot q \cdot (q^{-1} \bmod q') + \mathbf{v}_i[j] \cdot q' \cdot (q'^{-1} \bmod q)) \mod \hat{q} \tag{3}$$

for $i = 1, ..., l$ and $j = 1, ..., n$. Then, the function $g_{R_{\hat{q}}}$ is applied to the lifted coefficients which results in values $(\mathbf{p}_1, \ldots, \mathbf{p}_k) = g(\mathbf{z}_1, \ldots, \mathbf{z}_l)$. The integrity of the results of the calculation can now be checked by simply reducing all $\mathbf{p}_i \bmod q'$ and comparing the result with the respective $\mathbf{t}_i$. Accordingly, the intended result of the function g_{R_q} can then be obtained by reducing all values $\mathbf{p}_i$ modulo q as $\mathbf{w} = \mathbf{p}_i \bmod q$. Of course, it is also possible to set $g(\mathbf{v}, \hat{\mathbf{s}}, \mathbf{u}) = \mathbf{v} - \mathrm{INTT}(\hat{\mathbf{s}} \circ \mathrm{NTT}(\mathbf{u}))$ to protect the linear parts of the Kyber decryption. Note that our countermeasure does not interfere with the MLWE structure of Kyber and has no dependency on n or f in R_q. Moreover, the computational overhead is rather low as $q \cdot (q^{-1} \bmod q')$ and $q' \cdot (q'^{-1} \bmod q)$, which are used during the combination, are constant. Thus, the overhead for combining for one coefficient is two multiplications and one addition modulo $\hat{q}$. The computation of the result and checksum requires reductions modulo q and q', respectively.

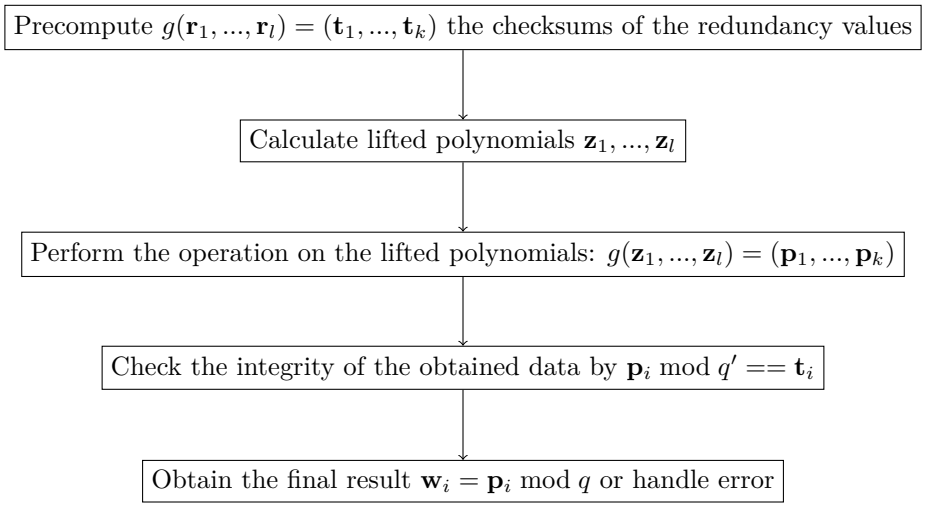


Figure 4: Functionality of the fault countermeasure with CRT

4.2 Evaluation and Analysis

In this work, we aim for an appropriate fault protection on a standard microcontroller using a realistic fault model [KSV13, MDH+13] that is also related to attack classes presented in [BBK16]. We consider an attacker who is able to precisely control the injection of one fault out of the three categories into a coefficient in $\mathbb{Z}_{\hat{q}}$ in a data processing or storage part of the device during the execution of the function g:

- Bit-flipping fault: An attacker may flip (bit-flip fault) or set a certain amount of bits (stuck-at fault) of data in a device under attack. This may be achieved by using optical fault injection with a laser that shoots into the register file.
- Random fault: An attacker may disturb a computation or a memory access so that the device proceeds with a random or pseudo random data element. This may be achieved by under-powering, power spiking, or clock glitching during multiplication or readout of data from RAM.
- Zeroization fault: An attacker may introduce a fault so that a zero value is processed, e.g., by enabling an internal power gate.

Intuitively, an attacker has to induce a fault such that the result modulo q is changed (to make the attack effective) and where the checksum modulo q' is the same. Thus, while storing or processing a coefficient $x \in \mathbb{Z}_{q'}$ in a suitable representation (e.g., as integer reduced in 0 to $q'-1$), the attacker needs to introduce a fault $e \in \{0,1\}^{\lceil \log_2(q') \rceil}$ in a binary bit flipping model such that $x \bmod q' = x \oplus e \bmod q'$ and $x \bmod q \neq x \oplus e \bmod q$.

For example, for $q = 7$ and $q' = 3$ we get $\hat{q} = 21$, $q^{-1} \bmod q' = 1$, and $q'^{-1} \bmod q = 5$. When combining all values $v \in \mathbb{Z}_q$ with all values $r \in \mathbb{Z}'_q$ into a value $z \in \mathbb{Z}_{\hat{q}}$ we get the representations modulo $\hat{q}$ as depicted in Table 1 in hexadecimal notation.

Table 1: Encoding of $r \in \mathbb{Z}_3$ and $v \in \mathbb{Z}_7$ with $\hat{q} = 7 \cdot 3$ using the CRT.

	$v=0$	$v=1$	$v=2$	$v=3$	$v=4$	$v=5$	$v=6$
$r=0$	0x00	0x03	0x06	0x09	0x0c	0x0f	0x12
$r=1$	0x07	0x0a	0x0d	0x10	0x13	0x01	0x04
$r=2$	0x0e	0x11	0x14	0x02	0x05	0x08	0x0b

In each class modulo q', the minimum distance is two and thus, to introduce a bit-flipping fault that will not get detected, an attacker has to flip at least two bits in each class. For example, by flipping bit 0 and bit 2 in 0x09=9 the value becomes 0x0c=12. The checksum stays the same as $9 \bmod q' = 0$ and $12 \bmod q' = 0$ while the encoded value changes from $9 \bmod q = 2$ to $12 \bmod q = 5$. The detection of a zeroization fault is more problematic. In cases where an element in $\hat{q}$ encoding $r = 0$ is faulted, an attacker likely succeeds with a zeroization fault. With such a fault, the attacker can change the 0x03 to 0x12 representations (second line of Table 1) encoding $r = 0$ to zero without detection.

We now analyze the properties of parameters sets q, q' that one would commonly encounter in lattice-based cryptosystems. For this we enumerate all representations and compute the minimum distance using brute-force. This gives the number of bits an attacker would need to flip in a bit-flipping attack. In addition, we estimate the success probability of a random value attack as $(q-1)/\hat{q}$. In this attack, the attacker has to hit one of the $q-1$ values that represent a target r with a different v out of the $\hat{q}$ possibilities[4]. If we assume no special care is taken to counter the zeroization attack and a uniform distribution of coefficients of $\mathbf{p}_i$'s and $\mathbf{r_i}$'s, then the probability of being successful is $(q-1)/\hat{q}$ as

[4]This calculation may slightly differ in practice depending on the implementation and representation of elements in $\mathbb{Z}_{\hat{q}}$ and how the implementation handles a fault (e.g., randomization of a full 32-bit register) that may cause a value larger than $\hat{q}$.

the attack succeeds when a coefficient is hit that encodes $r = 0$ and $z \neq 0$. In Table 2 we provide our results. Increasing q' does naturally result in better protection against a random value attack and zeroization attack. However, the resistance to bit flipping does not necessarily increase and depends highly on a particular choice of q'. It is also important to consider that our method only protects the function g and additional protection may be required for the combination of data and checksum as well as extraction and validation of the correct computation (e.g., skipping faults).

Table 2: Analysis of our countermeasure for different parameters

Scheme	q	q'	overhead in bits	min. dist	prob rand val.
Example	7	3	2 (40 %)	2	0.285714
Example	17	3	1 (33 %)	2	0.313725
Example	17	5	2 (43 %)	2	0.188235
Kyber [ABD+20b]	3329	257	8 (45 %)	2	0.00389
Kyber	3329	1699	11 (48 %)	4	0.000588
Kyber	3329	7681	13 (52 %)	3	0.000130
NTRU Prime [BBC+20]	4621	257	8 (42 %)	2	0.003890
NTRU Prime	4621	1699	10 (48 %)	4	0.000588
NTRU Prime	4621	7681	13 (50 %)	3	0.000130
NewHope [AAB+20]	12289	257	8 (41 %)	2	0.003891
NewHope	12289	1699	11 (44 %)	4	0.000588
NewHope	12289	7681	13 (48 %)	3	0.000130

4.3 Optimization and Control Flow Protection

With our approach it is not only possible to detect faults on single coefficients but also to check that the correct sequence of addition and multiplication operations in R_q was carried out. If the wrong computations are performed, this is reflected in a mismatch of the final checkvalue. Thus, our method can be used to prevent an attacker from using an instruction skipping fault (see [MDP+20]) to suppress, e.g., the addition of a noise vector $\mathbf{e}$ to an RLWE sample $\mathbf{a} \cdot \mathbf{s} + \mathbf{e}$ to yield a trivially solvable RLWE instance [BBK16]. To improve efficiency, constant values, e.g., public-keys or secret keys, can already be stored in the ring $R_{\hat{q}}$ during personalization of a device. This may additionally prevent an attacker from changing a pointer to a key to a data structure that contains only zero coefficients in R_q as the checksum will then not match with a very high probability.

As mentioned, it is not necessary to store the $\mathbf{r}_i$'s as a constant. To reduce memory consumption, it is possible to generate them on-the-fly based on a seed using a suitable PRNG like AES in counter mode or even a linear-feedback shift register (LFSR) or any other non-cryptography PRNG. Moreover, it is also not required to store the values $\mathbf{t}_i$. They can be compressed using an appropriate hash function h, e.g., to check that $h(\mathbf{p}_i \bmod q')$ is equal to $h(\mathbf{t}_i)$. Moreover, even a combination of all checksums can be used such that $h(\mathbf{p}_1 \bmod q', \ldots, \mathbf{p}_l \bmod q')$ is equal to $h(\mathbf{t}_1, \ldots, \mathbf{t}_l)$ so that only the short digest $h(\mathbf{t}_1, \ldots, \mathbf{t}_l)$ has to be stored.

5 Combination of RNR and CRT Countermeasures

Many lattice-based cryptography algorithms depend on efficient polynomial arithmetic using the NTT (see Section 2.2). Therefore, the technique of efficiently transforming polynomials into the NTT domain, multiplying them coefficient-wise and transforming them back to normal domain should remain applicable in a new ring $R_{\hat{q}} = R_{qq'}$ mandated by the RNR or CRT countermeasure. This can be achieved by choosing a ring $R_{q'}$ that

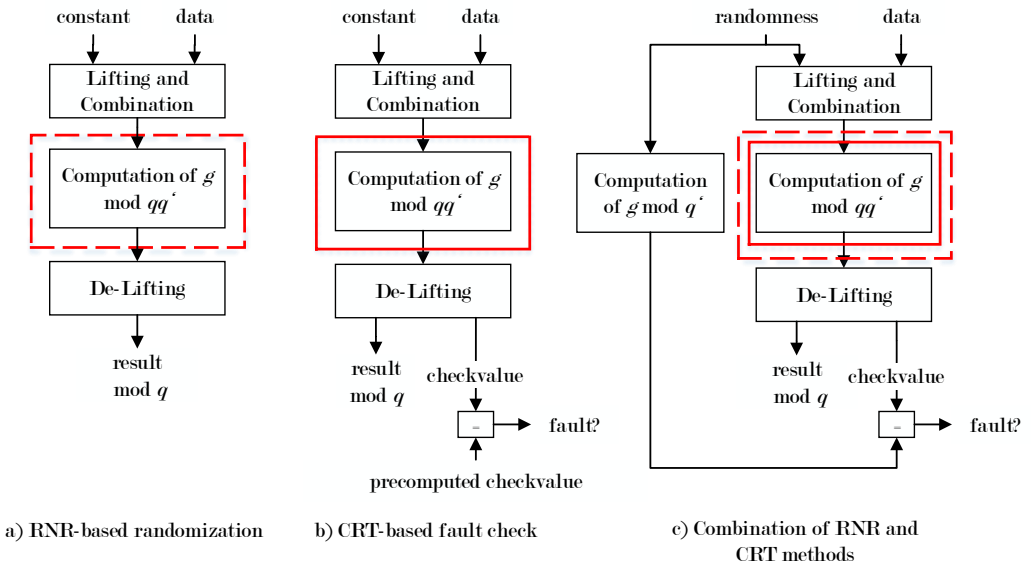


Figure 5: Overview on our RNR and CRT-based countermeasures as well as their combination with DPA protected parts within a dashed red line and fault protected parts within a solid red line.

contains a suitable primitive root of unity $\omega_{q'}$. We can then combine the roots of unity and its powers as

$$\omega_{\hat{q}} = \omega_q \cdot q' \cdot (q'^{-1} \bmod q) + \omega_{q'} \cdot q \cdot (q^{-1} \bmod q')$$
$$\omega_{\hat{q}}^{-1} = \omega_q^{-1} \cdot q' \cdot (q'^{-1} \bmod q) + \omega_{q'}^{-1} \cdot q \cdot (q^{-1} \bmod q')$$

to obtain roots of unity in $R_{\hat{q}}$.

For the NTT as defined in Kyber (see Section 2.3.1) with $q = 3329$, we need a primitive 256-th root of unity. By choosing $q' = 7681$, this property is fulfilled with $\omega_{q'} = 198$ and the Kyber NTT only needs to be adapted to the larger modulus which also results in an update of constants used in the Montgomery reduction.

To combine both countermeasures, it is helpful to consider that Equation 3 in Section 4 is just a different way of generating a redundant number representation for elements in $\mathbb{Z}_q$ by multiplying the value $\mathbf{r}[j]$ not just with q but also with the constant $q^{-1} \bmod q'$ (see a) and b) in Figure 5). The two countermeasures can thus be combined by using random inputs $\mathbf{r}_1, ..., \mathbf{r}_l$ that randomize the computation of $g_{R_{\hat{q}}}$ for each execution (see c) in Figure 5). To be able to check the result, it is of course also necessary to compute $g_{R_{q'}}(\mathbf{r}_1, ..., \mathbf{r}_l) = (\mathbf{t}_1, ..., \mathbf{t}_k)$. Moreover, the NTT constants (i.e., ζ^i) have to be taken into account as well (see Section 3). An attacker can now either attack $g_{R_{\hat{q}}}$ or $g_{R_{q'}}$ according to the analysis and assumptions in Section 4.2. However, using random constants implies that the danger of zeroing attacks might be prevented by explicitly avoiding values $\mathbf{r}[j] = 0$.

6 Evaluation

In this section we evaluate the countermeasures from Section 3, Section 4, and Section 5 with regard to their performance and effectiveness. We focus on protecting arithmetic operations of the decryption algorithm of Kyber (see Algorithm 3).

Table 3: Cycle counts of our Kyber NTT protected by the RNR on a Cortex-M4.

Implementation		Cycle Count	% of unprotected NTT
Unprotected NTT (reference)	[ABCG20]	6 829	-
Masked NTT (2× unprot.)	[ABCG20]	$2 \cdot 6829 = 13\,658$	200%
NTT with RNR	**This work**	7 737	113%

6.1 Performance of the RNR Countermeasure

We perform our evaluation on an ARM Cortex-M4 32-bit microcontroller. The ARM Cortex-M4 was chosen due to its popularity when evaluating PQC [BKS19, KRSS20, ABCG20] and due to the availability of already highly optimized code for comparison.

As development environment we use the Keil Toolchain MDK Plus 5.29/µVision 5.29 with the ARM Compiler Version 5. Our measurements for the Cortex-M4 architecture are performed using a STM32F4-DISCOVERY board with an STM32F407 that can run with up to 168 MHz with 1 Mbyte of flash memory and 192 kByte of RAM. For our measurements, we set the clock frequency to 24 MHz. The Cortex-M4 architecture features an instruction set extension - namely `ARMv7E-M` with instructions `uadd16`, `usub16`, `sasx`, and `ssax` - that makes a very fast NTT possible as proposed in [BKS19]. The target device includes the system timer (SysTick) which is used for measuring cycle counts.

In Table 3, we provide measured cycle counts of side-channel protected (see Section 3) implementations of the NTT of round 3 Kyber [ABD+20b]. The NTT implementation in [ABCG20] is constant-time but does not contain further countermeasures and is taken from the PQM4 library [KRSS20] as reference. This is a highly optimized NTT for the specific architecture. For reference, we provide cycle counts for a masked implementation in Table 3. It requires the computation of the NTT on two shares and thus consumes twice as many cycles as the reference implementation. Our implementation of the NTT protected by the RNR is realized in assembly and uses the concept of the 32-bit assembly NTT from the Dilithium implementation in [GKS20]. We adapted the Dilithium NTT, which is originally using the modulus $q = 2^{23} - 2^{13} + 1$, to the Kyber case (see Section 2.3.1 for details on the specific NTT used by Kyber) and changed the modulus to $qq' = 3329 \cdot 7681 = 25570049$.

For masking and the RNR approach, the same number of random bits are used. As a consequence, we do not include the time required for sampling these values in our cycle counts. As mentioned, we set $q' = 7681$ and use 12 bits of randomness for each coefficient. It is not necessary to randomize the NTT constants, as for our parameters no constant contains q' as a factor and thus no de-randomization happens (see Section 3.3). The RNR, therefore, requires a total of $n \cdot 12 = 3072$ random bits. For the masking, we use 12 random bits per coefficient and thus need $n \cdot 12 = 3072$ bits of randomness as well.

6.2 Performance of the CRT and Combined Countermeasure

For the evaluation of our CRT-based fault detection mechanism (see Section 4) we use the Kyber parameter $q = 3329$ and set the same $q' = 7681$ that we used to evaluate the RNR. This allows us to reuse the NTT code and Montgomery reduction. The protection level obtained by this parameter set is detailed in Table 2. We measured the cost of the combination operation from Section 4 and the NTT in the larger combined ring $R_{\hat{q}}$. The results of the fault countermeasures are listed in Table 4. We compare its performance to the state-of-the-art method of redundant computation. All in all, an NTT protected by our CRT-based method results in a smaller computational overhead compared to the state-of-the-art redundant computation. In addition, we note that the performance of the CRT approach gets more favorable when a large number of operations is performed in the lifted domain.

The evaluation of the combined countermeasure (see Section 5) is performed in a similar

Table 4: Cycle counts for our Kyber NTT protected against faults by the CRT countermeasure on a Cortex-M4.

Implementation		Cycle Count	% of unprotected NTT
Redundant NTT (2× unprot.)	[ABCG20]	$2 \cdot 6829 = 13\,658$	200%
NTT with CRT	**This work**	11 619	170%
→ Combination		3 882	
→ NTT in $R_{\hat{q}}$		7 737	

Table 5: Cycle counts for our Kyber NTT combining the RNR and CRT-based countermeasure on a Cortex-M4.

Implementation		Cycle Count	% of unprotected NTT
Redundant and masked NTT	[ABCG20]	$4 \cdot 6829 = 27\,316$	400%
NTT with CRT and RNR	**This work**	18 448	270%
→ Combination		3 882	
→ NTT in $R_{q'}$		6 829	
→ NTT in $R_{\hat{q}}$		7 737	

manner. A state-of-the-art implementation with protection against both, side-channel and fault attacks, is assumed to be realized with a redundant computation of a secret split into two shares. This leads to a larger overhead as can be seen in Table 5. The CRT and RNR methods provide a significant speedup to this state-of-the-art method. The combined method differs from the CRT method as the checksum can not be precomputed anymore. This leads to an additional computation of an NTT in $R_{q'}$.

6.3 Effectiveness against Side-Channel Attacks

To evaluate the effectiveness of the RNR and our implementation, we performed a power analysis using the ChipWhisperer [OC14] Lite platform with a STM32F303 target that is based on a Cortex-M4 processor core. We captured the power consumption of the NTT variants evaluated in Section 6.1 and Section 6.2 and use $q = 3329$ and $q' = 7681$ unless otherwise stated. Due to the usage of the ChipWhisperer platform [OC15] and its synchronous capture method, all traces are well synchronized. Thus, a lower number of traces is required in comparison to traditional side-channel setups. The main goal of our experiments is mainly to identify possible caps between theory and practice. Therefore, we apply the non-specific t-test evaluation methodology [SM15]. Hereby, we detect possible leakages that are not part of any specific leakage model. At each time point, we calculate the t-test statistic

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

where μ_0, s_0, and n_0 are the sample mean, variance, and sample size of the power traces with fixed input and μ_1, s_1, and n_1 those of the power traces with random input respectively. If the power traces of a constant-time implementation cause such a value at a specific time to exceed an absolute value of 4.5, we can detect a power difference between the usage of a fixed input or a random one and therefore the implementation is considered as not sufficiently secured.

For the initial tests, we use the down-sampling feature of the Chipwhiserer to capture the entire NTT with at most 24 400 samples for each execution. First, we verify our setup and t-test implementation by measuring our Kyber NTT protected by the RNR but with RNG off as shown in Figure 6. As expected, the test clearly shows leakage as data was processed without sufficient randomization.

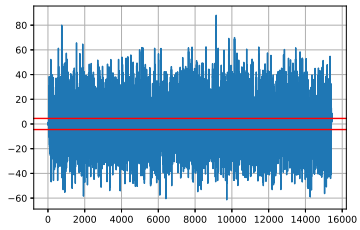


Figure 6: t-values of non-specific t-test for our Kyber NTT protected by the RNR with 1 000 traces and RNG off ($q' = 7681$)

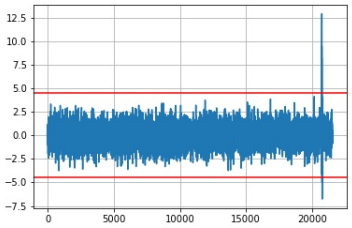


Figure 7: t-values of non-specific t-test for our Kyber NTT protected by the RNR with 1 000 traces and parameters leading to derandomization due to NTT constants (here, $q' = 257$ as discussed in Section 3.2)

In Section 3.2 we discussed that a naive instantiation of the RNR countermeasure could lead to a rather insecure implementation. To practically verify this observation we measured the Kyber NTT protected by the RNR with inappropriately chosen parameters ($q' = 257$ as in Section 3.2) and show the results in Figure 7. The peak at the end of the Figure shows the position where a multiplication by an NTT constant is performed, which is a multiple of $q' = 257$. As expected, the randomization is not sufficient and the threshold of the t-test is exceeded.

We additionally evaluated the NTT for $q' = 7681$ and where 12 random bits are used to randomize every input coefficient. The result of this test for 10 000 traces is shown in Figure 8. It does not show any leakage beyond the threshold.

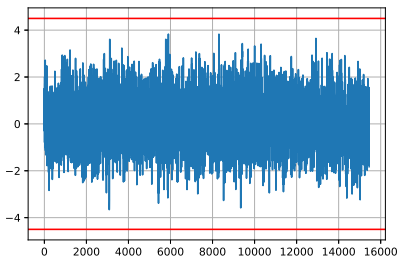


Figure 8: t-values of non-specific t-test for our Kyber NTT protected by the RNR with 10 000 traces ($q' = 7681$, 12 bits of randomness for every coefficient)

To verify our recommended implementation in more detail, we disable the downsampling feature of the Chipwhisperer to obtain more information from each trace. As a consequence,

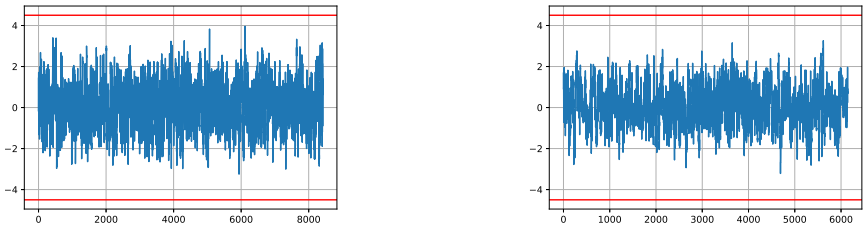


Figure 9: t-values of non-specific t-test for our Kyber NTT protected by the RNR with 10 000 traces for the first and second (left) and last round (right) of the NTT ($q' = 7681$, 12 bits of randomness for every coefficient)

we cannot capture a full NTT anymore. In Figure 9, we show the result of the t-test on the power traces of the first, second, and last round of the Cooley-Tukey NTT implementation (see Appendix A.1 where each round is equal to one iteration in the outer `for`-loop) using the same parameter setting. In none of these traces leakage beyond the threshold is captured. This verifies the assumption that the RNR can be used to randomize computations in lattice-based cryptography if parameters are chosen correctly (see Section 3.2).

We used the same setup and methodology as for the RNR to verify the combination of the RNR with the CRT as introduced in Section 5. We split the NTT into different parts and did not use the downsampling feature to obtain more detailed t-test values. However, we increased the range of random checkvalues in comparison to the RNR evaluation from $[0, 4096)$ to $[0, 7681)$. In this case, the t-test with 100 000 traces for (exemplary) the first, second, and last round of the NTT, i.e. the diagram in Figure 10, shows indeed no leakage, like its analogue from Figure 9. It should be noted that q' is chosen larger than in the theoretical analysis of Section 3.3. This is presumably the reason why even with 100 000 traces and the low noise level for the ChipWhisperer Lite, we cannot identify any leakage.

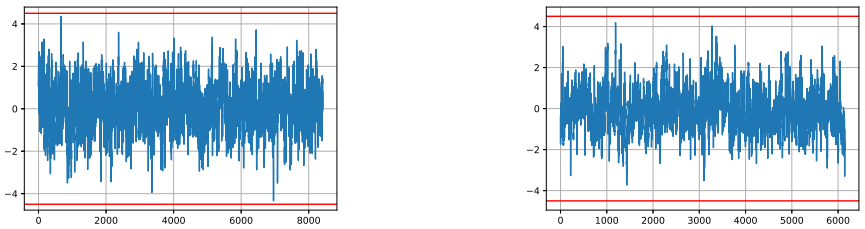


Figure 10: t-values of non-specific t-test for our Kyber NTT protected by the RNR and CRT with 100 000 traces for the first and second (left) and last round (right) of the NTT ($q' = 7681$, 13 bits of randomness for every coefficient)

6.4 Kyber Decryption

For combined protection, the overhead factor of four on the Cortex-M4, obtained from masking and redundant computations is reduced to around 2.7 for one execution of the NTT (see Table 5). We also applied the combined countermeasures to the arithmetic part

$$\mathbf{v} - \mathrm{INTT}(\hat{\mathbf{s}} \circ \mathrm{NTT}(\mathbf{u}))$$

of the Kyber768.CPA decryption in Algorithm 3. The cycle counts in Table 6 therefore only include the arithmetic operations of the decryption in the implementation [KRSS20]

on the Cortex-M4. However, we manage to reduce the computational overhead factor for leakage and fault-protected implementation from 2.9 to around 2.2 on the Cortex-M4. The pseudocode of the algorithms and the measurements is given in Appendix A.2 and A.3. If more shared operations are needed, the RNR and CRT method will gain additional performance compared to redundant computations of the shares.

Table 6: Cycle counts for $\mathbf{v} - \text{INTT}(\hat{\mathbf{s}} \circ \text{NTT}(\mathbf{u}))$ in Kyber768.CPA.Dec decryption on Cortex-M4

Implementation	Protection	Source	Cycle Count
Unprotected	none	[KRSS20, ABCG20]	79 509
Masking & Redundancy	DPA & Fault	[KRSS20, ABCG20]	229 922
RNR & CRT	DPA & Fault	**This work**	174 858

7 Conclusion and Future Work

In this work, we have analyzed the redundant number representation (RNR) and proposed the application of the Chinese remainder theorem (CRT) techniques to protect arithmetic operations in lattice-based cryptography. A combination of both methods leads to a speed-up factor of roughly 1.3 compared to the straightforward approach of masking and redundant calculation for the protection of linear parts of Kyber decryption on the ARM Cortex-M4. Both the RNR and CRT offer a security-time trade-off that can be adjusted according to the specific use-case by changing the parameter q'. Moreover, due to the low performance overhead of only 13 percent of an RNR protected NTT, the RNR approach could be used as an additional countermeasure in a masked implementation to possibly achieve higher order protection or some resistance against single-trace attacks. A combination of the methods with masking countermeasures would be an interesting topic for future work. Additional future work may consist of the optimization of the NTT for 32-bit coefficients on other architectures than the Cortex-M4.It may also be interesting to evaluate the impact of using the RNR to protect against single trace attacks as it increases the amount of possible intermediate values. Moreover, a practical evaluation of the CRT countermeasure using laser fault injection could further substantiate our theoretical model and analysis but is considered out of the scope of this work. The provided strength against multiple faults is also still open in theory and in practice. And even though ring extension methods seem more suitable for microcontroller implementations due to the fixed size of the ALU, it might still be interesting to implement them in hardware. As already shown in [ZBT19], it is for example possible on FPGAs to chose parameters that fit into the width of DPS or RAM hard macros. Another interesting avenue for future work might be the search for optimal parameters for the CRT and an improvement of the brute-force method to compute the minimum distance for larger moduli often used in lattice-based digital signatures.

Acknowledgments

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. The authors would like to thank the Chair for Communication Systems and Network Security as well as the research institute CODE at the Bundeswehr University in Munich, headed by Prof. Dreo, for their comments and improvements.

References

AAB+20. Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. NewHope: Algorithm specification and supporting documentation. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. `https://newhopecrypto.org/data/NewHope_2020_04_10.pdf`.

ABCG20. Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for {R, M} LWE schemes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):336–357, 2020.

ABD+20a. Roberto Avanzi, Joppe Bos, Láo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, , and Damien Stehlé. CRYSTALS–Kyber (version 2.0) – submission to round 2 of the nist post-quantum project. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. `https://pq-crystals.org/kyber/data/kyber-specification-round2.pdf`.

ABD+20b. Roberto Avanzi, Joppe Bos, Láo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, , and Damien Stehlé. CRYSTALS–Kyber (version 3.0) – submission to round 3 of the nist post-quantum project. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. `https://pq-crystals.org/kyber/data/kyber-specification-round3.pdf`.

ABGV08. AC Atici, Lejla Batina, Benedikt Gierlichs, and Ingrid Verbauwhede. Power analysis on NTRU implementations for RFIDs: First results. In *The 4th Workshop on RFID Security, July 9th -11th, Budapest*, 2008.

ACLZ20. Dorian Amiet, Andreas Curiger, Lukas Leuenberger, and Paul Zbinden. Defeating NewHope with a single trace. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*, volume 12100 of *Lecture Notes in Computer Science*, pages 189–205. Springer, 2020.

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 327–343. USENIX Association, 2016.

ATT+18. Aydin Aysu, Youssef Tobah, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018, Washington, DC, USA, April 30 - May 4, 2018*, pages 81–88. IEEE Computer Society, 2018.

BBC+20. Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Christine van Vredendaal, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, and Bo-Yin Yang. NTRU Prime: round 3, 2020. `https://ntruprime.cr.yp.to/nist/ntruprime-20201007.pdf`.

BBE+18. Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 354–384. Springer, 2018.

BBK16. Nina Bindel, Johannes A. Buchmann, and Juliane Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*, pages 63–77. IEEE Computer Society, 2016.

BCN+06. Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proc. IEEE*, 94(2):370–382, 2006.

BDH+19. Ciprian Baetu, F. Betül Durak, Loïs Huguenin-Dumittan, Abdullah Talayhan, and Serge Vaudenay. Misuse attacks on post-quantum cryptosystems. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 747–776. Springer, 2019.

BDK+18. Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 353–367. IEEE, 2018.

BDK+20. Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel resistant implementation of SABER. *IACR Cryptol. ePrint Arch.*, 2020:733, 2020.

BDL01. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.

BFM+18. Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. Assessing the feasibility of single trace power analysis of Frodo. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 216–234. Springer, 2018.

BKS19. Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. Memory-efficient high-speed implementation of Kyber on Cortex-M4. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje-eddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings*, volume 11627 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 2019.

BP18. Leon Groot Bruinderink and Peter Pessl. Differential fault attacks on deterministic lattice signatures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):21–43, 2018.

BPO+20. Florian Bache, Clara Paglialonga, Tobias Oder, Tobias Schneider, and Tim Güneysu. High-speed masking for polynomial comparison in lattice-based KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):483–507, 2020.

CAB19. Jérôme Courtois, Lokman A. Abbas-Turki, and Jean-Claude Bajard. Resilience of randomized RNS arithmetic with respect to side-channel leaks of cryptographic computation. *IEEE Trans. Computers*, 68(12):1720–1730, 2019.

CDH+20. Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. NTRU algorithm specifications and supporting documentation, 2020. `https://ntru.org/f/ntru-20190330.pdf`.

CHK+20. Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT multiplication for ntt-unfriendly rings. *IACR Cryptol. ePrint Arch.*, 2020:1397, 2020.

CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

CS21. Nicolas Costes and Martijn Stam. Redundant code-based masking revisited. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):426–450, 2021.

DKL+18. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

DKL+20. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS–Dilithium algorithm specifications and supporting documentation, 2020. `https://pq-crystals.org/dilithium/data/dilithium-specification-round3.pdf`.

DKRV19. Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. 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`.

EFGT18. Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Loop-abort faults on lattice-based signature schemes and key exchange protocols. *IEEE Trans. Computers*, 67(11):1535–1549, 2018.

FGM+10. Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-art of secure ECC implementations: A survey on known side-channel attacks and countermeasures. In Jim Plusquellic and Ken Mai, editors, *HOST 2010, Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 13-14 June 2010, Anaheim Convention Center, California, USA*, pages 76–87. IEEE Computer Society, 2010.

FO99. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *LNCS*, pages 537–554. Springer, 1999.

Für09. Martin Fürer. Faster integer multiplication. *SIAM J. Comput.*, 39(3):979–1005, 2009.

FV12. Junfeng Fan and Ingrid Verbauwhede. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In David Naccache, editor, *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, volume 6805 of *Lecture Notes in Computer Science*, pages 265–282. Springer, 2012.

GJN20. Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the fujisaki-okamoto transformation and its application on frodokem. *IACR Cryptol. ePrint Arch.*, 2020:743, 2020.

GKS20. Denisa O. C. Greconici, Matthias J. Kannwischer, and Daan Sprenkels. Compact Dilithium implementations on Cortex-M3 and Cortex-M4. Cryptology ePrint Archive, Report 2020/1278, 2020. `https://eprint.iacr.org/2020/1278`.

GR19. François Gérard and Mélissa Rossi. An efficient and provable masked implementation of qTESLA. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2019.

GRS99. Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese remaindering with errors. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 225–234. ACM, 1999.

HHK17. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, 2017.

HKM+19. James Howe, Ayesha Khalid, Marco Martinoli, Francesco Regazzoni, and Elisabeth Oswald. Fault attack countermeasures for error samplers in lattice-based cryptography. In *IEEE International Symposium on Circuits and Systems, ISCAS 2019, Sapporo, Japan, May 26-29, 2019*, pages 1–5. IEEE, 2019.

JKL+. Pierre-Alain Fouque Jeffrey, Hoffstein Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over NTRU. `https://falcon-sign.info/falcon.pdf`.

Joy19. Marc Joye. Protecting ECC against fault attacks: The ring extension method revisited. *IACR Cryptol. ePrint Arch.*, 2019:495, 2019.

KJJ99. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

Koc96. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

KPP20. Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):243–268, 2020.

KRSS20. Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4, 2020. `https://github.com/mupq/pqm4`, accessed 12/16/2020.

KSV13. Dusko Karaklajic, Jörn-Marc Schmidt, and Ingrid Verbauwhede. Hardware designer's guide to fault attacks. *IEEE Trans. Very Large Scale Integr. Syst.*, 21(12):2295–2306, 2013.

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015.

MD99. Thomas S. Messerges and Ezzy A. Dabbish. Investigations of power analysis attacks on smartcards. In Scott B. Guthery and Peter Honeyman, editors, *Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999*. USENIX Association, 1999.

MDH+13. Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 77–88. IEEE Computer Society, 2013.

MDP+20. Alexandre Menu, Jean-Max Dutertre, Olivier Potin, Jean-Baptiste Rigaud, and Jean-Luc Danger. Experimental analysis of the electromagnetic instruction skip fault model. In *15th Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2020, Marrakech, Morocco, April 1-3, 2020*, pages 1–7. IEEE, 2020.

MGTF19. Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*, volume 11464 of *Lecture Notes in Computer Science*, pages 344–362. Springer, 2019.

MKSDG10. LEE Mun-Kyu, Jeong Eun Song, and HAN Dong-Guk. Countermeasures against power analysis attacks for the NTRU public key cryptosystem. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 93(1):153–163, 2010.

Mon85. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.

MOP07. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

Mos18. Michele Mosca. Cybersecurity in an era with quantum computers: Will we be ready? *IEEE Secur. Priv.*, 16(5):38–41, 2018.

Nat16. National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`.

NDR+19. Hamid Nejatollahi, Nikil D. Dutt, Sandip Ray, Francesco Regazzoni, Indranil Banerjee, and Rosario Cammarota. Post-quantum lattice-based cryptography implementations: A survey. *ACM Comput. Surv.*, 51(6):129:1–129:41, 2019.

NIS20. NIST. Post-quantum cryptography - round 3 submissions, 2020. `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

OC14. Colin O'Flynn and Zhizhang (David) Chen. ChipWhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 243–260. Springer, 2014.

OC15. Colin O'Flynn and Zhizhang Chen. Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection. *J. Cryptogr. Eng.*, 5(1):53–69, 2015.

OSPG18. Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure and masked ring-LWE implementation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):142–174, 2018.

Pes16. Peter Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings*, volume 10095 of *Lecture Notes in Computer Science*, pages 153–170, 2016.

PFPB19. Louiza Papachristodoulou, Apostolos P. Fournaris, Kostas Papagiannopoulos, and Lejla Batina. Practical evaluation of protected residue number system scalar multiplication. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):259–282, 2019.

PP19. Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2019.

PPM17. Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.

RBN+15. Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

RBRC20. Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. Drop by drop you break the rock - exploiting generic vulnerabilities in lattice-based PKE/KEMs using EM-based physical attacks. *IACR Cryptol. ePrint Arch.*, 2020:549, 2020.

RdCR+16. Oscar Reparaz, Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Additively homomorphic ring-lwe masking. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 233–244. Springer, 2016.

RJH+19. Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. Exploiting Determinism in Lattice-based Signatures: Practical Fault Attacks on PQM4 Implementations of NIST Candidates. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, pages 427–440. ACM, 2019.

RPBC20. Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On configurable SCA countermeasures against single trace attacks for the NTT - A performance evaluation study over kyber and dilithium on the ARM Cortex-M4. *IACR Cryptol. ePrint Arch.*, 2020:1038, 2020.

RRCB20. Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based PKE and kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):307–335, 2020.

RRdC+16. Oscar Reparaz, Sujoy Sinha Roy, Ruan de Clercq, Frederik Vercauteren, and Ingrid Verbauwhede. Masking ring-LWE. *J. Cryptographic Engineering*, 6(2):139–153, 2016.

RRVV15. Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-lwe implementation. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 683–702. Springer, 2015.

Saa18. Markku-Juhani O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures - engineering a side-channel resistant post-quantum signature scheme with compact signatures. *J. Cryptographic Engineering*, 8(1):71–84, 2018.

Sch06. F. W. Scholz. *Maximum Likelihood Estimation.* American Cancer Society, 2006. ISBN: 9780471667193, Available: `https://onlinelibrary.wiley.com/doi/abs/10.1002/0471667196.ess1571.pub2`.

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.

SM15. Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.

SPOG19. Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, volume 11443 of *LNCS*, pages 534–564. Springer, 2019.

TE15. Mostafa Taha and Thomas Eisenbarth. Implementation attacks on post-quantum cryptographic schemes. *IACR Cryptol. ePrint Arch.*, 2015:1083, 2015.

VKS11. Ingrid Verbauwhede, Dusko Karaklajic, and Jörn-Marc Schmidt. The fault attack jungle - A classification model to guide you. In Luca Breveglieri, Sylvain Guilley, Israel Koren, David Naccache, and Junko Takahashi, editors, *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*, pages 3–8. IEEE Computer Society, 2011.

VOGR18. Felipe Valencia, Tobias Oder, Tim Güneysu, and Francesco Regazzoni. Exploring the vulnerability of R-LWE encryption to fault attacks. In John Goodacre, Mikel Luján, Giovanni Agosta, Alessandro Barenghi, Israel Koren, and Gerardo Pelosi, editors, *Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems, CS2 2018, Manchester, United Kingdom, January 24, 2018*, pages 7–12. ACM, 2018.

WZW13. An Wang, Xuexin Zheng, and Zongyue Wang. Power analysis attacks and countermeasures on NTRU-based wireless body area networks. *TIIS*, 7(5):1094–1107, 2013.

ZBT19. Timo Zijlstra, Karim Bigou, and Arnaud Tisserand. FPGA implementation and comparison of protections against SCAs for RLWE. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*, volume 11898 of *Lecture Notes in Computer Science*, pages 535–555. Springer, 2019.

ZWW13. Xuexin Zheng, An Wang, and Wei Wei. First-order collision attack on protected NTRU cryptosystem. *Microprocessors and Microsystems - Embedded Hardware Design*, 37(6-7):601–609, 2013.

A Appendix

A.1 Pseudo Code of Cooley-Tukey FFT Algorithm

Input: `Array` p `representing a polynomial with` n `coefficients where` n `is a power of 2`
Output: `Discrete Fourier Transform of` p

```
1  j, k = 1;
2  for len = 128; len >= 2; len >>= 1 do
3  |  for start = len; start < 256; start = j + len do
4  |  |  ζ ← zetas[k++] ;
5  |  |  for j = start; j < start + len; ++j do
6  |  |  |  t ← ζ · p[j + len] mod q ;
7  |  |  |  p[j + len] ← p[j] − t ;
8  |  |  |  p[j] ← p[j] + t ;
9  |  |  end
10 |  end
11 end
12 return p
```

Algorithm 4: Cooley-Tukey FFT in PQCLEAN

A.2 Pseudo Code of Kyber.CPA.Dec with RNR and CRT

Input: Ciphertext c, randomized secret key sk with random polynomial r_{secret}
Output: Message m

```
1  Generate KYBER_K random polynomials in random[KYBER_K];
2  /*Start cycle count*/;
3  poly_unpackdecompress(mp, c, 0);
4  combined = poly_combine(random[0], mp);
5  poly_nttQPrime(random[0]);
6  poly_nttQHat(combined);
7  poly_frombytes_mulQPrime(random[0],r_secret);
8  poly_frombytes_mulQHat(combined,sk);
9  for i = 1; i < KYBER_K; i++ do
10 |  poly_unpackdecompress(bp, c, i);
11 |  combined2 = poly_combine(random[i], bp);
12 |  poly_nttQPrime(random[i]);
13 |  poly_nttQHat(combined2);
14 |  poly_frombytes_mulQPrime(random[i],
   |   r_secret + i · KYBER_POLYBYTES);
15 |  poly_frombytes_mulQHat(combined2, sk + i · KYBER_POLYBYTES);
16 |  poly_addQPrime(random[0], random[0], random[i]);
17 |  poly_addQHat(combined, combined, combined2);
18 end
19 poly_invnttQPrime(random[0]);
20 poly_invnttQHat(combined);
21 poly_decompress(v, c + KYBER_POLYVECCOMPRESSEDBYTES);
22 poly_subQHat(combined, v, combined);
23 poly_subQPrime(random[0],v, random[0]);
24 /*Stop cycle count*/;
25 Check for faults;
26 Calculate output message m;
27 return m
```

A.3 Pseudo Code of Kyber.CPA.Dec with Redundancy and Masking

Input: Ciphertext c, Redundantly shared secret key $sk = sk_i[0] + sk_i[1]$
Output: Message m

for $j = 0;\ j < 2;j++$ **do**
  /*Start cycle count*/;
  poly_unpackdecompress(mp, c, 0);
  mp[0]= mp[1] = poly_ntt(mp);
  poly_frombytes_mul(mp[0],$sk_j[0]$);
  poly_frombytes_mul(mp[1],$sk_j[1]$);
  for $i = 1;\ i < KYBER\_K;\ i++$ **do**
    poly_unpackdecompress(bp, c, i);
    bp[0] = bp[1] = poly_ntt(bp);
    poly_frombytes_mul(bp[0], $sk_j[0] + i \cdot KYBER\_POLYBYTES$);
    poly_frombytes_mul(bp[1], $sk_j[1] + i \cdot KYBER\_POLYBYTES$);
    poly_add(mp[0], mp[0], bp[0]);
    poly_add(mp[1], mp[1], bp[1]);
  end
  poly_invntt(mp[0]);
  poly_invntt(mp[1]);
  poly_decompress(v,
   $c + KYBER\_POLYVECCOMPRESSEDBYTES$);
  poly_sub(mp[0], v, mp[0]);
  poly_sub(mp[1],v, mp[1]);
  /*Stop cycle count*/;
  Calculate output messages m_j;
end
 Check for faults;
return m