Round5:
KEM and PKE based on (Ring) Learning with Rounding
Thursday 28th March, 2019

Hayo Baan[1], Sauvik Bhattacharya[1], Scott Fluhrer[5], Oscar Garcia-Morchon[1], Thijs Laarhoven[4], Rachel Player[6], Ronald Rietman[1], Markku-Juhani O. Saarinen[2], Ludo Tolhuizen[1], José Luis Torre-Arce[1], and Zhenfei Zhang[3]

[1]Philips (Netherlands)
[2]PQShield (UK)
[3]Algorand (US)
[4]TU/e (Netherlands)
[5]Cisco (US)
[6]RHUL (UK)

Contents

1 Cover Sheet

Name of the proposed cryptosystem:	The proposed cryptosystem is called *Round5*. Round5 consists of a key-encapsulation mechanism (KEM) named *r5_cpa_kem*, and a public-key encryption (PKE) scheme named *r5_cca_pke*.
Principal submitter:	Oscar Garcia-Morchon, oscar.garcia-morchon@philips.com, +31611073603, Philips International B.V., High Tech Campus 5, 5656 AE Eindhoven, The Netherlands.
Names of the auxiliary-submitters :	Baan, H., (Philips, NL) Bhattacharya, S., (Philips, NL) Fluhrer, S., (Cisco, US) Laarhoven, T., (TU Eindhoven, NL) Player, R., (RHUL, UK) Rietman, R., (Philips, NL) Saarinen, M.J.O., (PQShield, UK) Tolhuizen, L., (Philips, NL) Torre-Arce, J.L., (Philips, NL) Zhang, Z., (Algorand, US).
Names of the inventors-developers :	Baan, H., (Philips, NL) Bhattacharya, S., (Philips, NL) Fluhrer, S., (Cisco, US) Garcia-Morchon, O., (Philips, NL) Laarhoven, T., (TU Eindhoven, NL) Player, R., (RHUL, UK) Rietman, R., (Philips, NL) Saarinen, M.J.O., (PQShield, UK) Tolhuizen, L., (Philips, NL) Torre-Arce, J.L., (Philips, NL) Zhang, Z., (Algorand, US).
Name of the owner:	Koninklijke Philips N.V.
Signature of the principal submitter:	
Backup point of contact:	Ludo Tolhuizen, ludo.tolhuizen@philips.com, +31615543195, Philips International B.V., High Tech Campus 34, 5656 AE Eindhoven, The Netherlands.

2 Algorithm Specifications

This submission proposes Round5 that consists of algorithms for the key encapsulation mechanism r5_cpa_kem and the public-key encryption scheme r5_cca_pke.

The proposed algorithms fall under the category of lattice-based cryptography. Round5 is a merger of the submissions Round2 [14, 15] and HILA5 [90, 92]. Like with Round2, the security relies on the General Learning With Rounding (GLWR) problem, which allows for a unified design for instantiations based on Learning with Rounding (LWR) or Ring Learning with Rounding (RLWR). In some of its parameter sets, Round5 uses an error-correcting code based on the one from HILA5 to decrease the decryption failure probability, and thus, achieve smaller keys and better performance.

The main contents of the submission are structured as follows:

- Section 2.1 reviews the design rationale of Round5. Section 2.3 defines the General Learning with Rounding problem.
- Section 2.4 and Figure 1 detail the Round5 algorithms and building blocks in a formal way. Section 2.4.7 and 2.9 detail the parameter sets and performance results.

  The Round5 cryptosystem proposes a unified scheme that can be configured to fit the needs of different applications. This specification details 18 main parameters sets: six ring parameter sets with error correction, six ring parameter sets without error correction, and six non-ring parameter sets. Each set of six parameter sets is for r5_cpa_kem and r5_cca_pke, and for NIST security levels 1, 3 and 5. The parameter sets are provided in Tables 3, 4 and 5 for r5_cpa_kem and Tables 11, 11 and 12 for r5_cca_pke.

  Round5 has three additional "specific use-case" parameter sets with the only purpose of demonstrating the flexibility of Round5 and its applicability to a number of diverse, specialized usage scenarios. These parameter sets can be found in Tables 6 and 14.

  Finally, Section 2.11 details the implementation of Round5 from an implementation perspective, including the specification of the bit ordering, of hash functions, and of functions used for generating randomness.
- Section 2.6 and Figure 2 show that the Round5 family of algorithms stems from a provable secure design. More details can be found in Appendix A.

  Section 2.7 analyzes the concrete security of Round5 from the perspective of best known attacks.

  Section 2.8 reviews the correctness of Round5, in particular, it contains an analysis of the probability of a decapsulation or decryption error.

  Finally, Section 2.10 discusses advantages and limitations of Round5.

2.1 Design Rationale

We design Round5's design rationale is characterized by 3 features: a) a unified design; b) design choices for optimized performance; c) ring choice.

2.1.1 A Unified Design

A key feature of Round5 is that it has been designed so that it can be instantiated with the Learning With Rounding (LWR) problem and the Ring Learning With Rounding (RLWR) problem in a seamless and unified way. This is done by defining the General LWR problem, on which Round5 is based, that can instantiate LWR or RLWR depending on the input parameters. The reasons behind this choice are as follows:

Round5 is adaptive and can be applied to multiple environments. On the one hand, LWR-based algorithms are required in environments in which performance is less of an issue, but security is the priority. In those cases, it is often preferred to not have an additional ring structure (as in ideal lattices [73, 56]). On the other hand, RLWR-based algorithms achieve the best performance in terms of bandwidth and computation so they are better suited for constrained environments with stricter bandwidth requirements, e.g., due to the complexity of message fragmentation or small MTUs. It is possible to derive especially small parameter sets applicable to IoT scenarios, or non-ring parameter sets with balanced or unbalanced public-key and ciphertext sizes.

Round5 reduces code analysis and maintenance since the unified scheme definitions of r5_cpa_kem and r5_cca_pke instantiate different underlying problems, LWR and RLWR, with the same code.

The unified design enables a migration strategy from ring-based schemes to non-ring schemes from day one of deployment. This makes sure that if vulnerabilities in ring-based problems were to be found in future, then an alternative secure solution would already be available and could be deployed directly. Indeed, a series of results [41, 40, 24] and most recently [83] indicate that Ideal Approximate-SVP [99], a problem typically considered to justify the hardness of the Ring-LWE problem may be a weaker problem than Approximate-SVP for arbitrary lattices. Although this still does not call into question the hardness of Ring-LWE or Ring-LWR, it does raise a need to be conservative and offer competitive alternatives that do not rely on the hardness of lattice problems on structured lattices.

2.1.2 Design choices for optimized performance

Round5's design aims at optimizing performance.

- The usage of GLWR, i.e., LWR and RLWR, rather than their LWE counterparts, leads to lower bandwidth requirements in Round5, since fewer bits need to be transmitted per coefficient.
- Rounding avoids sampling of noise, and thus reduces the amount of random data to be generated. The generation of noise may be vulnerable to

both implementation errors and side-channel attacks, see [87],[59] and the references therein, which is a further advantage of not having to generate noise samples.

- Round5 relies on a definition of GLWR with sparse ternary secrets. This simplifies implementation and reduces the probability of decryption/decapsulation errors.

- The ring instantiation of Round5 relies on the RLWR problem over the cyclotomic ring $\mathbb{Z}_q[x]/\Phi_{n+1}(x)$ with $n+1$ prime. This problem is well studied: there exist reductions [19] from the RLWE problem [73] to RLWR, and the former is well-studied for the ring in question. Operations over this ring can be mapped to an NTRU [56] ring $\mathbb{Z}_q[x]/(x^{n+1}-1)$ to improve performance.

- For some of its parameter sets, Round5 uses an f-bit error correcting block code XEf to decrease the failure rate, see Section 2.4.1. The code is built using the same strategy as codes used by TRUNC8 [92] (2-bit correction) and HILA5 [91] (5-bit correction). The main advantage of XEf codes is that they avoid table look-ups and conditions altogether, and are therefore resistant to timing attacks. The usage of XEf requires performing the ciphertext computations in the NTRU ring and requires sparse ternary secrets that are balanced, i.e., contain equally many ones as minus ones. This leads to independent bit failures so that error correction can be applied (Section 2.4.1).

- The moduli q and p are powers of two. This simplifies the implementation of the rounding function, as it can be realized by ignoring the least significant bits. Similarly, the modular computations can be realized by ignoring the most significant bits.

- Preventing pre-computation attacks requires refreshing the master public parameter $\boldsymbol{A}$. However, this can be computationally costly, in particular in the case of LWR. Round5 provides several alternatives for efficiently refreshing $\boldsymbol{A}$ that are applicable to different use cases.

2.1.3 The Choice of the Ring

In the literature, a common choice of the ring to instantiate an RLWE or RLWR scheme is $\mathbb{Z}_q[x]/\Phi_{2n}(x)$ where n is a power of two, so that the $2n$-th cyclotomic polynomial $\Phi_{2n}(x) = x^n + 1$. Examples of RLWE/RLWR key exchange schemes based on the above ring are [29] and [8]. However, requiring that n be a power of two severly the choice of n: it has to be a power of two, and it has to be at least 512 for the proper security level so that the underlying lattice problem is hard to solve in practice. While having $n = 512$ sometimes does not deliver the target security level, the $n = 1024$ choice would be considered an overkill. A sweet spot is $n \approx 700$, which was a common choice made by many proposals, including Kyber [30], NTRUEncrypt [56], NTRU-KEM [61] and more.

The following observations can be made:

- Kyber [30] uses three RLWE instances, each of dimension 256, to achieve $n = 768$ in total. This still limits the choice of n as it has to be a multiple of 256.

- NTRUEncrypt uses the reduction polynomial $x^n - 1$ which is slightly different from a cyclotomic ring, although the NTRU problem remains hard for this ring, the decisional RLWE problem over this ring seems to be easy [89].

This leads us to use as reduction polynomial the $(n+1)$-th cyclotomic polynomial $\Phi_{n+1}(x) = x^n + \cdots + x + 1$ with $n + 1$ a prime as in NTRU-KEM [61]. With this reduction polynomial, there is a wide range of n to choose from, for various security levels. In addition, as shown in [82], decisional RLWE over this ring remains hard for any modulus; this gives us confidence in the underlying design and security of Round5. Note that although operating over the same ring as the NTRU-KEM scheme, Round5 achieves better performance because its key generation algorithm is significantly faster than the NTRU key generation algorithm.

In addition, since decisional RLWE is hard over a prime cyclotomic ring with any modulus [82], we can use any modulus that optimizes our performance. Common choices of the modulus are

- A number theoretical transform (NTT) friendly prime number, such as 12289 in [8] (note that NTT is not compatible with rounding operations naively);

- A composite number that fits in a data type for modern computers, such as $2^{32} - 1$ in [29];

- A power of two that makes modulo operations and integer multiplications efficient, such as 2^{11} in NTRUEncrypt [56].

In our proposal, we consider a prime cyclotomic polynomial ring with a modulus q that is a power of two such that the Φ_{n+1} is irreducible modulo two. As Φ_{n+1} then is irreducible modulo q, the ring $\mathbb{Z}_q[x]/\Phi_{n+1}(x)$ does not have any proper subrings. This choice allows for a smooth implementation of the ring and non-ring cases in the unified scheme. All coefficients of our polynomials and matrices are integers modulo q less than 2^{16}. That is, each coefficient fits in a `uint16_t` type. For intermediate values during computations, overflows can be ignored, as overflown bits are “mod-ed out” once the element is lifted back to $\mathbb{Z}_q$. In particular, when multiplying two `uint16_t` elements, only the lower 16 bits of the product need to be computed; the higher 16 bits have no effect on the final result.

The use of a composite modulus in [29], as well as the result from [82] suggest that the particular modulus does not have much effect on the hardness of the problem; the size of the modulus is more important from this point of view. Consequently, any modulus of similar size should deliver a similar security level, and therefore depending on the use case we choose the one that is the most efficient.

We finally remark that in the parameter sets of Round5 that use error correction, operations on one component of the ciphertext, as well as decryption use reduction modulo $x^{n+1}-1$. The reason for doing so, as explained in Section 2.8, is to avoid correlated errors, which would make error correction much less effective.

2.2 Preliminaries

For each positive integer a, we denote the set $\{0,1,\dots,a-1\}$ by $\mathbb{Z}_a$. For a set A, we denote by $a \xleftarrow{\$} A$ that a is drawn uniformly from A. If χ is a probability distribution, then $a \leftarrow \chi$ means that a is drawn at random according to the probability distribution χ. Logarithms are in base 2, unless specified otherwise.

Modular reductions. For a positive integer α and $x \in \mathbb{Q}$, we define $\{x\}_\alpha$ as the unique element x' in the interval $(-\alpha/2, \alpha/2]$ satisfying $x' \equiv x \pmod{\alpha}$. Moreover, we define $\langle x \rangle_\alpha$ as the unique element x' in the interval $[0,\alpha)$ for which $x \equiv x' \pmod{\alpha}$.

Rounding. For $x \in \mathbb{Q}$, we denote by $\lfloor x \rfloor$ and $\lfloor x \rceil$ rounding downwards to the next integer and rounding to the closest integer (with rounding up in case of a tie) respectively. For positive integers a, b and $h \in \mathbb{Q}$, the rounding function $\mathrm{R}_{a\to b,h}$ is defined as

$$\mathrm{R}_{a\to b,h}(x) = \left\langle \left\lfloor \frac{b}{a}(x+h) \right\rfloor \right\rangle_b . \tag{1}$$

If $h = \frac{a}{2b}$, then $\mathrm{R}_{a\to b,h}(x) = \langle \lfloor \frac{b}{a} x \rceil \rangle_b$. This special case of rounding will be used in the underlying problem for Round5, so the following notation will come in handy: for positive integers a, b, the function $\mathrm{R}_{a\to b}$ is defined as

$$\mathrm{R}_{a\to b} = \mathrm{R}_{a\to b,a/2b}. \tag{2}$$

Ring choice. Let $n+1$ be prime. The $(n+1)$-th cyclotomic polynomial $\Phi_{n+1}(x)$ then equals $x^n + x^{n-1} + \cdots + x + 1$. We denote the polynomial ring $\mathbb{Z}[x]/\Phi_{n+1}(x)$ by $\mathcal{R}_n$. When n equals 1, then $\mathcal{R}_n = \mathbb{Z}$. For each positive integer a, we write $\mathcal{R}_{n,a}$ for $\mathbb{Z}_a[x]/\Phi_{n+1}(x)$. We call a polynomial in $\mathcal{R}_n$ *ternary* if all its coefficients are 0, 1 or -1. Throughout this document, regular font letters denote elements from $\mathcal{R}_n$, and bold lowercase letters represent vectors with coefficients in $\mathcal{R}_n$. All vectors are column vectors. Bold uppercase letters are matrices. The transpose of a vector $\boldsymbol{v}$ or a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{v}^T$ or $\boldsymbol{A}^T$. A vector or matrix of polynomials, in which all polynomial entries have all their coefficients equal to 1, is denoted by $\boldsymbol{j}$ or $\boldsymbol{J}$, respectively.

Distributions. For each $v \in \mathcal{R}_n$, the **Hamming weight** of v is defined as its number of non-zero coefficients. The Hamming weight of a vector in $\mathcal{R}_n^k$ equals the sum of the Hamming weighs of its components. We denote with $\mathcal{H}_{n,k}(h)$ the set of all vectors $\boldsymbol{v} \in \mathcal{R}_n^k$ of ternary polynomials of Hamming weight h, where $h \leq nk$. By considering the coefficients of a polynomial in $\mathcal{R}_n$ as a vector of length n, a polynomial in $\mathcal{H}_{n,k}(h)$ corresponds to a ternary vector of length nk with non-zeros in h positions, so that $\mathcal{H}_{n,k}(h)$ has $\binom{nk}{h}2^h$ elements. When $k = 1$, we omit it from the notation, and $\mathcal{H}_n(h)$ denotes the set of all ternary polynomials in $\mathcal{R}_n$ of Hamming weight h, corresponding to the set of all vectors $\boldsymbol{v} \in \{-1, 0, 1\}^n$ with Hamming weight h.

Secret keys consist of matrices that contain (column) vectors in $\mathcal{H}_{n,k}(h)$. Functions f_R and f_S are used to generate secrets from a seed in the encryption (Algorithm 2) and decryption (Algorithm 3), respectively.

2.3 The General Learning With Rounding Problem

The problem underlying the security of Round5 is the **General Learning With Rounding (GLWR) problem** formally defined as follows:

Definition 2.3.1 (General LWR (GLWR))**.** *Let d, n, p, q be positive integers such that $q \geq p \geq 2$, and $n \in \{1, d\}$. Let $\mathcal{R}_{n,q}$ be a polynomial ring, and let D_s be a probability distribution on $\mathcal{R}_{n,q}^{d/n}$.*

The search version of the GLWR problem $\mathtt{sGLWR}_{d,n,m,q,p}(D_s)$ is as follows: given m samples of the form $R_{q\to p}(\boldsymbol{a}_i^T \boldsymbol{s})$ with $\boldsymbol{a}_i \in \mathcal{R}_{n,q}^{d/n}$ and a fixed $\boldsymbol{s} \leftarrow D_s$, recover $\boldsymbol{s}$.

The decision version of the GLWR problem $\mathtt{dGLWR}_{d,n,m,q,p}(D_s)$ is to distinguish between the uniform distribution on $\mathcal{R}_{n,q}^{d/n} \times \mathcal{R}_{n,p}$ and the distribution $(\boldsymbol{a}_i, b_i = R_{q\to p}(\boldsymbol{a}_i^T \boldsymbol{s}))$ with $\boldsymbol{a}_i \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n}$ and a fixed $\boldsymbol{s} \leftarrow D_s$.

For brevity, when $D_s = \mathcal{U}\left(\mathcal{H}_{n,d/n}(h)\right)$, we denote $\mathtt{GLWR}_{d,n,m,q,p}\left(\mathcal{U}\left(\mathcal{H}_{n,d/n}(h)\right)\right)$ as $\mathtt{GLWR}_{\text{spt}}$. When the secret distribution D_s is the uniform one over $\mathcal{R}_{n,q}^{d/n}$, it shall be omitted in the above problem notation.

Instantiating GLWR as LWR. When $n = 1$, the GLWR problem is equivalent to the LWR problem [19] with dimension d, large modulus q and rounding modulus p. Setting the distribution $D_s = \mathcal{U}(\mathcal{H}_{1,d}(h))$ further specializes the GLWR problem to the LWR problem with sparse-ternary secrets $\mathtt{LWR}_{\text{spt}}$. In [37] it is claimed that the hardness of $\mathtt{LWR}_{\text{spt}}$ can be obtained from that of LWE with similar secret distributions since the reduction from LWE to LWR is independent of the secret's distribution [25]. We extend this claim and make it explicit by proving that for appropriate parameters there exists a polynomial-time reduction from the (decision) Learning with Errors (LWE) problem with secrets chosen uniformly from $\mathbb{Z}_q^d$ and errors chosen from a Gaussian distribution D_α, to the decision version of $\mathtt{LWR}_{\text{spt}}$. See Section A.8 and Theorem A.8.1 for more details.

Instantiating GLWR as RLWR. When $n = d > 1$ is such that $n + 1$ is prime, and $\mathcal{R}_{n,q} = \mathbb{Z}_q[x]/\left(\Phi_{n+1}(x)\right)$ for the $n+1$-th cyclotomic polynomial $\Phi_{n+1}(x) = 1 + x + \ldots + x^n$, the GLWR problem is equivalent to the RLWR problem with reduction polynomial $\Phi_{d+1}(x)$, dimension d, large modulus q and rounding modulus p. Setting $D_s = \mathcal{U}\left(\mathcal{H}_{d,1}(h)\right)$ further specializes it to the RLWR problem with sparse-ternary secrets $\texttt{RLWR}_{\text{spt}}$.

Possible instantiation as Module Learning with Rounding (MLWR) The NIST submission CRYSTALS-Kyber [13] has as underlying problem the Module LWE (MLWE) problem [71]. In prior work [33], MLWE was first introduced as the General Learning with Errors problem, but only instantiated with the ring case and the non-ring cases. The NIST submission SABER [43] has the corresponding Module LWR(MLWR) problem as underlying hard problem. In both submissions, the rationale is that a single ring needs to be optimized, and that different security levels can be obtained by choosing a different module rank.

In contrast, Round5 can easily choose from many different possible rings $\mathcal{R}_{n,q}$. We note that if the GLWR condition $n \in \{1, d\}$ is relaxed by stipulating that n only needs to be a divisor of d, this would lead to the MLWR problem with module rank equal to d/n. In this configuration, it would be possible to vary over d, n and the moduli p, q and t, allowing for even a more fine-grained optimization.

2.4 Round5

Our proposal is called Round5. It includes an IND-CPA secure KEM called r5_cpa_kem and a IND-CCA secure PKE called r5_cca_pke. A public-key encryption scheme called r5_cpa_pke is used as a building block for r5_cpa_kem. A key-encapsulation mechanism called r5_cca_kem, is used as a building block for r5_cca_pke. All algorithms in these schemes use random choices, and scheme-specific mappings that are described in the following sections with each scheme.

In Section 2.4.1, the error-correcting codes applied in some parameter sets of Round5 are described. These codes are built using the same strategy as codes used by TRUNC8 [92] (2-bit correction) and HILA5 [91] (5-bit correction). In each protocol instantiation, Round5 uses a fresh public parameter $\boldsymbol{A}$. In Section 2.4.2, three methods for generating $\boldsymbol{A}$ are described. r5_cpa_kem is described in Section 2.4.4, preceded by the description of its building block r5_cpa_pke in Section 2.4.3. r5_cca_pke is described in Section 2.4.6, preceded by its building block r5_cca_kem in Section 2.4.5. Figure 1 provides an overview of our proposal. It also shows the different instantiations of the proposed schemes based on the underlying GLWR problem.

2.4.1 Internal building block: error correction code

In [50], it is analyzed how error-correcting codes can be used the enhance the error resilience of protocols like [7] [28],[30], and it is shown that the usage of

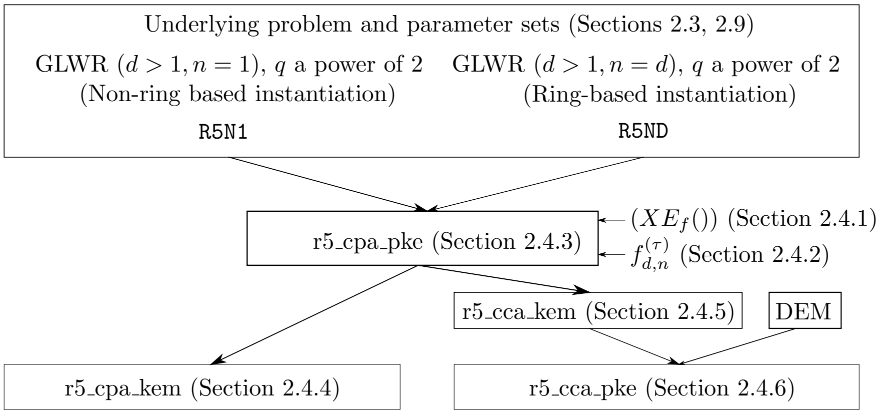
Underlying problem and parameter sets (Sections 2.3, 2.9)
GLWR ($d > 1, n = 1$), q a power of 2
(Non-ring based instantiation)
R5N1
GLWR ($d > 1, n = d$), q a power of 2
(Ring-based instantiation)
R5ND
r5_cpa_pke (Section 2.4.3)
$(XE_f())$ (Section 2.4.1)
$f_{d,n}^{(\tau)}$ (Section 2.4.2)
r5_cca_kem (Section 2.4.5)
DEM
r5_cpa_kem (Section 2.4.4)
r5_cca_pke (Section 2.4.6)

Figure 1: Submission overview

error correcting codes can significantly increase the estimated bit-security and decrease the communication overhead. For some of its parameter sets, Round5 uses an f-bit error correcting block code XEf to decrease the failure rate. The code is built using the same strategy as codes used by TRUNC8 [92] (2-bit correction) and HILA5 [91] (5-bit correction).

XEf. The XEf code is described by $2f$ "registers" r_i of size $|r_i| = l_i$ with $i = 0, \ldots, 2f - 1$. We view the κ-bits payload block m as a binary polynomial $m_{\kappa-1}x^{\kappa-1} + \cdots + m_1 x + m_0$ of length κ. Registers are defined via cyclic reduction

$$r_i = m \bmod x^{l_i} - 1, \tag{3}$$

or equivalently by

$$r_i[j] = \sum_{k \equiv j \bmod l_i} m_k \tag{4}$$

where $r_i[j]$ is bit j of register r_i. A transmitted message consists of the payload m concatenated with register set r (a total of $\mu = \kappa + xe$ bits, where $xe = \sum l_i$).

Upon receiving a message $(m' \mid r')$, one computes the register set r'' corresponding to m' and compares it to the received register set r' – that may also have errors. Errors are in coefficients m'_k where there are parity disagreements $r'_i[k \mod l_i] \neq r''_i[k \mod l_i]$ for multitude of registers r_i. We use a majority rule and flip bit m'_k if

$$\sum_{i=0}^{2f-1} ((r'_i[\langle k \rangle_{l_i}] - r''_i[\langle k \rangle_{l_i}]) \bmod 2) \geq f + 1 \tag{5}$$

where the sum is taken as the number of disagreeing register parity bits at k.

XEf codes can include a special register – that is marked with symbol $(*)$ such that $\ell^{(*)}$ divides κ, and the parity bits in register $r^{(*)}$ are computed as the sum of $\kappa/\ell^{(*)}$ consecutive bits. That is, for $0 \leq j \leq l^{(*)} - 1$, bit $r^{(*)}[j]$ is defined as

$$r^{(*)}[j] = \sum_{i=0}^{\frac{\kappa}{\ell^{(*)}}-1} m_{j \cdot \frac{\kappa}{\ell^{(*)}}+i}.$$

In HILA5[91]'s XE5 code, r_0 is such a special register. When using this special register, decoding is done as explained above, but with the term corresponding to the special register $r^{(*)}$ replaced by $\left(r'^{(*)}[\lfloor \frac{k\ell^{(*)}}{\kappa} \rfloor] - r"^{(*)}[\lfloor \frac{k\ell^{(*)}}{\kappa} \rfloor]\right) \mod 2$.

As shown in HILA5[91], if all length pairs satisfy $\text{lcm}(l_i, l_j) \geq \kappa$ when $i \neq j$, then this code always corrects at least f errors. If the XEf code includes a special register $r^{(*)}$, say $r^{(*)} = r_0$, then it is required that $\frac{\kappa}{\ell_0} \leq l_i$ for all $j \geq 1$ and $\text{lcm}(l_i, l_j) \geq \kappa$ for $i, j \geq 1$ and $i \neq j$.

The main advantage of XEf codes is that they avoid table look-ups and conditions altogether and are therefore resistant to timing attacks.

Requirements for using XEf in Round5. A basic requirement for using XEf error correction code is that the errors it aims to correct are independent. As explained in Section 2.4.3, Round5 can use two reduction polynomials $\xi(x) = \Phi_{n+1}(x) = \frac{x^{n+1}-1}{x-1}$ or $\xi(x) = N_{n+1}(x) = x^{n+1} - 1$ in the computation of the ciphertext and in the decryption/decapsulation. As explained in more detail in Section 2.8, using $\xi(x) = \Phi_{n+1}(x)$ leads to correlated errors. The basic reason is that the difference polynomial $d(x)$ governing the error behavior is obtained as $d(x) = \langle \sum_{i=0}^{n} f_i x^i \rangle_{\Phi_{n+1}(x)} = \sum_{i=0}^{n} f_i x^i - f_n \Phi_{n+1}(x) = \sum_{i=0}^{n-1} (f_i - f_n) x^i$. As a result, if f_n is large, then many coefficients of $d(x)$ are likely to be large, leading to errors in the corresponding bit positions. As a consequence of this correlation between errors, the XEf code cannot be directly employed with the reduction polynomial $\xi(x) = \Phi_{n+1}(x)$ as used in Round2.

As detailed in Section 2.8, Round5 aims to obtain independent errors by imposing two requirements:

- Ciphertext computation and decryption use the reduction polynomial $\xi(x) = N_{n+1}(x)$.
- The secret ternary polynomials $s(x)$ and $r(x)$ are balanced, that is, both have as many coefficients equal to 1 as to -1.

The latter requirement implies that $(x - 1)$ is a factor of both $s(x)$ and $r(x)$. As a consequence, for any polynomial $a(x)$, it holds that $\langle s(x)a(x) \rangle_{\Phi_{n+1}(x)} r(x) \equiv s(x) \rangle a(x) r(x) \rangle_{\Phi_{n+1}(x)} \pmod{N_{n+1}(x)}$. This equivalence relation is essential for operational correctness, cf. Section 2.8.

These two requirements are aligned with features of the original Round2 submission [14] since (i) Round2 already internally performed all operations in $\xi(x) = N_{n+1}(x)$ and (ii) Round2's implementation used balanced secrets.

2.4.2 Internal building block: $f_{d,n}^{(\tau)}$ for generating a public matrix

In each protocol instantiation, Round5 uses a fresh public parameter $\boldsymbol{A}$ (cf. Section 2.4.3), which is a $d/n \times d/n$ matrix with entries from $\mathbb{Z}_q[x]/\Phi_{n+1}(x)$. The refreshing of $\boldsymbol{A}$ prevents backdoor and pre-computation attacks, cf. Section 2.7.8. The public matrix $\boldsymbol{A}$ is computed from a seed σ using function $f_{d,n}^{(\tau)}$. Round5 has three options for $f_{d,n}^{(\tau)}$:

- $f_{d,n}^{(0)}$ generates the entries of $\boldsymbol{A}$ by means of a deterministic random bit generator (DRBG) initialized by a seed σ specific for the protocol exchange. The DRBG must be called $(d/n)^2 \times n = d^2/n$ times; this can lead to performance penalties in non-ring settings.
- $f_{d,n}^{(1)}$, which only applies for $n = 1$, generates $\boldsymbol{A}$ by permuting a public and system-wide matrix $\boldsymbol{A_{master}}$. The permutation is derived from a seed σ. It is described by a vector $\boldsymbol{p}_1 \in Z_d^d$, and the fresh $\boldsymbol{A}$ is defined as

  $$\boldsymbol{A}[i,j] = \boldsymbol{A_{master}}[i, \boldsymbol{p1}[i] + j\rangle_d].$$

  The entries of $\boldsymbol{p}_1$ are obtained from a DRBG initialized with σ and with uniform output in $\mathbb{Z}_d$. With this approach, only d calls to a DRBG are required, instead of d^2 as with $f_{d,n}^{(0)}$.
- $f_{d,n}^{(2)}$, which only applies for $n = 1$, generates $\boldsymbol{A}$ by first generating a vector $\boldsymbol{a_{master}} \in \mathbb{Z}_q^{len\_tau\_2}$ from a seed σ and then permuting it. This requires len_tau_2 calls to the DRBG. The entries of the permutation vector $\boldsymbol{p}_2$ are obtained from a DRBG initialized with σ and with uniform output in $\mathbb{Z}_{len\_tau\_2}$; rejection sampling is used to ensure that the entries of $\boldsymbol{p}_2$ are distinct. The matrix $\boldsymbol{A}$ is defined as

  $$A[i,j] = \boldsymbol{a}_{master}[\langle \boldsymbol{p2}[i] + j\rangle_{len\_tau\_2}].$$

  With this approach, it is possible to efficiently obtain a fresh $\boldsymbol{A}$ without having to compute a significant amount of pseudorandom data and without having to store a potentially large matrix $\boldsymbol{A_{master}}$. The default value for len_tau_2 is set to 2^{11}, the smallest power of two greater than d for any of the Round5 configurations.

$f_{d,n}^{(0)}$ is the standard approach followed by many other protocols [28],[30],[7], and for which security proofs are available (Section 2.6 and Appendix A.1). A distinguishability proof cannot be provided for $f_{d,n}^{(1)}$ and $f_{d,n}^{(2)}$; despite this, Round5 considers these two options since they can successfully deal with backdoor and pre-computation attacks (as argued in Section 2.7.8) and they provide clear performance advantages (Section 2.9.10). For instance, $f_{d,n}^{(1)}$ can increase the efficiency of a server that has to handle many connections; $f_{d,n}^{(2)}$ reduces memory needs for both initiator and responder, which can be especially useful on small devices. The specific implementation of $f_{d,n}^{(\tau)}$ is detailed in Section 2.11.6 when AES and CSHAKE are used as the underlying DRBG.

2.4.3 Internal building block: r5_cpa_pke

This section describes r5_cpa_pke, the CPA-secure public key encryption that is a building block in both r5_cpa_kem and r5_cca_pke. It consists of algorithms 1 (key-generation), 2 (encryption) and 3 (decryption), and various cryptosystem parameters, *viz* positive integers n, d, h, p, q, t, b, $\overline{n}$, $\overline{m}$, μ, f, τ, and a security parameter κ. The system parameters are listed in Table 1. In the proposed parameter sets, $n \in \{1, d\}$, and b, q, p, t are powers of 2, such that $b|t|p|q$. It is required that

Table 1: List of symbols and their meaning

n	Degree of reduction polynomial;
	also indicates if a ring ($n > 1$) or a non-ring ($n = 1$) instantiation is used
d	Number of polynomial coefficients per row of public matrix
h	Number of non-zero polynomial coefficient per column of secret matrices
b, p, q, t	Rounding moduli, all powers of two, satisfying $b < t < p < q$
b_bits	Number of extracted bits per symbol in ciphertext component; $b = 2^{b\_bits}$.
$\overline{n}$	Number of columns of the secret matrix of the initiator
$\overline{m}$	Number of columns of the secret matrix of the responder
κ	Security parameter; number of information bits in error-correcting code
xe	Number of parity bits of error correcting code
μ	Number of symbols in ciphertext component: $\mu = \lceil \frac{\kappa + xe}{b_{bits}} \rceil$.
Sample$_\mu$	Function for picking up μ polynomial coefficients from a matrix
f	Number of bit errors correctable by error-correcting code
τ	Index for method of determining public matrix from a seed in $\{0,1\}^\kappa$
$f_{d,n}^{(\tau)}$	Method for determining public matrix from a seed in $\{0,1\}^\kappa$
f_S	Function for determining secret matrix for initiator from a seed in $\{0,1\}^\kappa$
f_R	Function for determining secret matrix for responder from a seed in $\{0,1\}^\kappa$
z	$z = \max(p, \frac{tq}{p})$. Relevant in reduction proofs and definition of rounding constants
h_1	$h_1 = \frac{q}{2p}$ is a rounding constant
h_2	$h_2 = \frac{q}{2z}$ is a rounding constant
h_3	$h_3 = \frac{p}{2t} + \frac{p}{2b} - \frac{q}{2z}$ is a constant for reducing bias in decryption
h_4	$h_4 = \frac{q}{2p} - \frac{q}{2z}$ is a constant for reducing bias in decryption
$\Phi_{n+1}(x)$	$\Phi_{n+1}(x) = \frac{x^{n+1}-1}{x-1}$. Reduction polynomial for computation of public keys; $n+1$ a prime.
$\xi(x)$	Reduction polynomial for computation of ciphertext. $\xi(x) \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$

$$\mu \leq \overline{n} \cdot \overline{m} \cdot n \text{ and } \mu * b\_bits \geq \kappa \text{ where } b = 2^{b\_bits}.$$

In this section, options like the choice of the DRBG underlying $f_{d,n}^{(\tau)}$ are hidden.

The function $\text{Sample}_\mu : \boldsymbol{C} \in \mathcal{R}_{n,p}^{\overline{n}\times\overline{m}} \to \mathbb{Z}_p^\mu$ outputs μ polynomial coefficients of the $\overline{n} \cdot \overline{m} \cdot n$ polynomial coefficients present in $\boldsymbol{C}$. These μ coefficients are used in the actual encryption algorithm. This function is fully specified in Appendix 2.11.4.

The integer f denotes the error-correction capability of a code $XE_{\kappa,f} \subset \mathbb{Z}_b^\mu$. We have an encoding function $xef\_compute_{\kappa,f} : \{0,1\}^\kappa \to XE_{\kappa,f}$ and a decoding function $xef\_decode_{\kappa,f} : \mathbb{Z}_b^\mu \to \{0,1\}^\kappa$ such that for each $m \in \{0,1\}^\kappa$ and each error $e = (e_0, \ldots, e_{\mu-1})$ with at most f symbols e_i different from zero,

$$xef\_decode_{\kappa,f}(xef\_compute_{\kappa,f}(m) + e) = m. \tag{6}$$

The functions $xef\_compute()$ and $xef\_decode()$, based on Xef codes, are detailed from an implementation perspective in Section 2.11.4.

Algorithm r5_cpa_pke_encrypt employs a deterministic function f_R for generating a secret matrix $\boldsymbol{R} \in \left(\mathcal{H}_{n,d/n}(h)\right)^{1\times\overline{m}}$ from an input ρ. Defining ρ as an explicit input to r5_cpa_pke_encrypt allows us to reuse this *same* algorithm as a building block for both IND-CPA and IND-CCA secure cryptographic constructions.

Furthermore, r5_cpa_pke uses four rounding constants, namely

$$h_1 = \frac{q}{2p},\ h_2 = \frac{q}{2z}, h_3 = \frac{p}{2t} + \frac{p}{2b} - \frac{q}{2z} \text{ and } h_4 = \frac{q}{2p} - \frac{q}{2z},\ \text{with } z = \max(p, \frac{tq}{p}) \tag{7}$$

The constant h_1 leads to rounding to the closest integer. The choice of h_2 ensures that Round5's ciphertext $(\boldsymbol{U}, \boldsymbol{v})$ is provably pseudorandom under the GLWR assumption. Details are provided in the proof of IND-CPA security for r5_cpa_pke and r5_cpa_kem (Section A.4). The choices of h_3 and h_4 are made to avoid bias in decryption, see Section 2.8.

Note that the rounding constants are added explicitly here for completeness, but with the specific parameter choices in the different Round5 configurations 2.4.7 some of them can be simplified: $h_1 = h_2 = q/2p$ leading to standard rounding as in Round2 [14, 15] implemented by means of flooring. Furthermore, $h_4 = 0$. Thus, the only difference with respect to Round2 is h_3, which is present to avoid bias in decryption.

Algorithm 1: r5_cpa_pke_keygen()

```
parameters: Integers $p, q, n, h, d, \overline{n}, \kappa, \tau$
input     : -
output    : $pk \in \{0,1\}^\kappa \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, sk \in \{0,1\}^\kappa$
$\sigma \xleftarrow{\$} \{0,1\}^\kappa$
$\boldsymbol{A} = \boldsymbol{f}_{d,n}^{(\tau)}(\sigma)$
$sk \xleftarrow{\$} \{0,1\}^\kappa$
$\boldsymbol{S} = f_S(sk)$
$\boldsymbol{B} = \mathrm{R}_{q \to p, h_1}(\langle \boldsymbol{A}\boldsymbol{S} \rangle_{\Phi_{n+1}})$
$pk = (\sigma, \boldsymbol{B})$
return $(pk, sk)$
```

Algorithm 2: r5_cpa_pke_encrypt(pk, m, ρ)

```
parameters: Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, b, \kappa, f, \tau$; $\xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$
input     : $pk = (\sigma, \boldsymbol{B}) \in \{0,1\}^\kappa \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, m, \rho \in \{0,1\}^\kappa$
output    : $ct = (\tilde{\boldsymbol{U}}, \boldsymbol{v}) \in \mathcal{R}_{n,p}^{\overline{m} \times d/n} \times \mathbb{Z}_t^\mu$
$\boldsymbol{A} = \boldsymbol{f}_{d,n}^{(\tau)}(\sigma)$
$\boldsymbol{R} = f_R(\rho)$
$\boldsymbol{U} = \mathrm{R}_{q \to p, h_2}(\langle \boldsymbol{A}^T \boldsymbol{R} \rangle_{\Phi_{n+1}})$
$\tilde{\boldsymbol{U}} = \boldsymbol{U}^T$
$\boldsymbol{v} = \langle \mathrm{R}_{p \to t, h_2}(\mathrm{Sample}_\mu(\langle \boldsymbol{B}^T \boldsymbol{R} \rangle_\xi)) + \frac{t}{b} xef\_compute_{\kappa,f}(m) \rangle_t$
$ct = (\tilde{\boldsymbol{U}}, \boldsymbol{v})$
return $ct$
```

Algorithm 3: r5_cpa_pke_decrypt(sk, ct)

```
parameters: Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, b, \kappa, f$; $\xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$
input     : $sk \in \{0,1\}^\kappa, ct = (\tilde{\boldsymbol{U}}, \boldsymbol{v}) \in \mathcal{R}_{n,p}^{\overline{m} \times d/n} \times \mathbb{Z}_t^\mu$
output    : $\hat{m} \in \{0,1\}^\kappa$
$\boldsymbol{v}_p = \frac{p}{t} \boldsymbol{v}$
$\boldsymbol{S} = f_s(sk)$
$\boldsymbol{U} = \tilde{\boldsymbol{U}}^T$
$\boldsymbol{y} = \mathrm{R}_{p \to b, h_3}(\boldsymbol{v}_p - \mathrm{Sample}_\mu((\boldsymbol{S}^T(\boldsymbol{U} + h_4 \boldsymbol{J}))_\xi))$
$\hat{m} = xef\_decode_{\kappa,f}(\boldsymbol{y})$
return $\hat{m}$
```

2.4.4 Submission proposal: r5_cpa_kem

This section describes r5_cpa_kem, an IND-CPA-secure key encapsulation method. It builds on r5_cpa_pke (Section 2.4.3). In addition to the parameters and functions from r5_cpa_pke, it uses a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$. In order to improve readability, in Algorithms 5 and 6 the conversion of the ciphertext ct into a binary string before it is fed to H is not made explicit. With $||$, we denote concatenation.

```
Algorithm 4: r5_cpa_kem_keygen()
    parameters: Integers $p, q, n, h, d, \overline{n}, \kappa, \tau$
    input     : -
    output    : $pk \in \{0,1\}^\kappa \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, sk \in \{0,1\}^\kappa$
    $(pk, sk)$ = r5_cpa_pke_keygen()
    return $(pk, sk)$
```

```
Algorithm 5: r5_cpa_kem_encapsulate($pk$)
    parameters: Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, b, \kappa, f, \tau$; $\xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$
    input     : $pk \in \{0,1\}^\kappa \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$
    output    : $(ct, k) \in (\mathcal{R}_{n,p}^{\overline{m} \times d/n} \times \mathbb{Z}_t^\mu) \times \{0,1\}^\kappa$
    $m \xleftarrow{\$} \{0,1\}^\kappa$
    $\rho \xleftarrow{\$} \{0,1\}^\kappa$
    $ct$ = r5_cpa_pke_encrypt($pk, m, \rho$)
    $k = H(m||ct)$
    return $(ct, k)$
```

```
Algorithm 6: r5_cpa_kem_decapsulate($ct, sk$)
    parameters: Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, b, \kappa, f$; $\xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$
    input     : $ct \in \mathcal{R}_{n,p}^{\overline{m} \times d/n} \times \mathbb{Z}_t^\mu, sk \in \{0,1\}^\kappa$
    output    : $k \in \{0,1\}^\kappa$
    $m$ = r5_cpa_pke.decrypt($sk, ct$)
    $k = H(m||ct)$
    return $k$
```

2.4.5 Internal building block: r5_cca_kem

This section describes r5_cca_kem that is a building block for r5_cca_pke. It consists of the algorithms 7, 8, 9, and several system parameters and functions in addition to those from r5_cpa_pke and r5_cpa_kem. In addition to the hash function H from r5_cpa_kem, it uses another hash function $G : \{0,1\}^* \rightarrow \{0,1\}^\kappa \times \{0,1\}^\kappa \times \{0,1\}^\kappa$. To improve readability, conversion of data types to bitstrings before being fed to G and H is not made explicit.

r5_cca_kem is actively secure as it is obtained by application of the Fujisaki-Okamoto transform [58] to r5_cpa_pke, similarly as in [30, Sec. 4]. On decapsulation failure, i.e. if the condition in line 4 of Algorithm 9 is not satisfied, a pseudorandom key is returned, causing later protocol steps to fail implicitly. Explicit failure notification would complicate analysis, especially in the quantum random oracle (QROM) case.

Algorithm 7: r5_cca_kem_keygen()

```
parameters: Integers $p, q, n, h, d, \overline{n}, \kappa, \tau$
input      : -
output     : $pk \in \{0,1\}^\kappa \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, sk \in \{0,1\}^\kappa \times \{0,1\}^\kappa \times (\{0,1\}^\kappa \times \mathcal{R}_{n,p}^{d/n \times \overline{n}})$
$(pk, sk_{CPA-PKE}) = $ r5_cpa_pke_keygen()
$y \xleftarrow{\$} \{0,1\}^\kappa$
$sk = (sk_{CPA-PKE}, y, pk)$
return $(pk, sk)$
```

Algorithm 8: r5_cca_kem_encapsulate(pk)

```
parameters: Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, b, \kappa, f, \tau$; $\xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$
input      : $pk \in \{0,1\}^\kappa \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$
output     : $ct = (\tilde{\boldsymbol{U}}, \boldsymbol{v}, g) \in \mathcal{R}_{n,p}^{\overline{m} \times d/n} \times \mathbb{Z}_t^\mu \times \{0,1\}^\kappa, k \in \{0,1\}^\kappa$
$m \xleftarrow{\$} \{0,1\}^\kappa$
$(L, g, \rho) = G(m||pk)$
$(\tilde{\boldsymbol{U}}, \boldsymbol{v}) = $ r5_cpa_pke_encrypt$(pk, m, \rho)$
$ct = (\tilde{\boldsymbol{U}}, \boldsymbol{v}, g)$
$k = H(L||ct)$
return $(ct, k)$
```

Algorithm 9: r5_cca_kem_decapsulate(ct, sk)

```
parameters: Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, b, \kappa, f, \tau$; $\xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$
input      : $ct = (\tilde{\boldsymbol{U}}, \boldsymbol{v}, g) \in \mathcal{R}_{n,p}^{\overline{m} \times d/n} \times \mathbb{Z}_t^\mu \times \{0,1\}^\kappa, sk =$
             $(sk_{CPA-PKE}, y, pk) \in \{0,1\}^\kappa \times \{0,1\}^\kappa \times (\{0,1\}^\kappa \times \mathcal{R}_{n,p}^{d/n \times \overline{n}})$
output     : $k \in \{0,1\}^\kappa$
$m' = $ r5_cpa_pke_decrypt$(sk_{CPA-PKE}, (\tilde{\boldsymbol{U}}, \boldsymbol{v}))$
$(L', g', \rho') = G(m'||pk)$
$(\tilde{\boldsymbol{U'}}, \boldsymbol{v'}) = $ r5_cpa_pke_encrypt$(pk, m', \rho')$
$ct' = (\tilde{\boldsymbol{U'}}, \boldsymbol{v'}, g')$
if $(ct = ct')$ then
    return $k = H(L'||ct)$
else
    return $k = H(y||ct)$
end if
```

2.4.6 Submission proposal: r5_cca_pke

The IND-CCA [86] public key encryption scheme r5_cca_pke consists of algorithms 10, 11 and 12. It combines r5_cca_kem with a data encapsulation mechanism (DEM), in the canonical way proposed by Cramer and Shoup [42]. r5_cca_kem is used to encapsulate a key k that is then used by the DEM to encrypt an arbitrary-length plaintext, optionally adding integrity protection. In decryption, r5_cca_kem is used to decapsulate k, which is then used by the DEM to decrypt and authenticate the plaintext. The expression $\lceil(\overline{m} \times d \times \log_2(p) + \mu \log_2(t) + \kappa)/8\rceil$ in Algorithms 11 and 12 denotes the length (in bytes) of the binary representation of elements of $(\mathcal{R}_{n,p}^{\overline{m}\times d/n} \times Z_t^{\mu} \times \{0,1\}^{\kappa})$.

Algorithm 10: r5_cca_pke_keygen()

```
parameters: Integers $p, q, n, h, d, \overline{n}, \kappa, \tau$
input     : -
output    : $pk \in \{0,1\}^{\kappa} \times \mathcal{R}_{n,p}^{d/n\times\overline{n}}, sk \in \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \times (\{0,1\}^{\kappa} \times \mathcal{R}_{n,p}^{d/n\times\overline{n}})$
$(pk, sk)$ = r5_cca_kem_keygen()
return $(pk, sk)$
```

Algorithm 11: r5_cca_pke_encrypt$(mlen, m, pk)$

```
parameters: Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, b, \kappa, f, \tau$; $\xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$
input     : $m \in \mathbb{Z}_{256}^{mlen}, mlen \in \mathbb{Z}, pk \in \{0,1\}^{\kappa} \times \mathcal{R}_{n,p}^{d/n\times\overline{n}}$
output    : $ct = (c1||c2) \in (\mathcal{R}_{n,p}^{\overline{m}\times d/n} \times Z_t^{\mu} \times \{0,1\}^{\kappa}) \times \mathbb{Z}_{256}^{c2len}, clen =$
            $\lceil(\overline{m} \times d \times \log_2(p) + \mu\log_2(t) + \kappa)/8\rceil + c2len \in \mathbb{Z}$
$(c1, k)$ = r5_cca_kem_encapsulate$(pk)$
$(c2, c2len) = \mathrm{DEM}(k, m)$
$ct = (c1||c2)$
$clen = \lceil(\overline{m} \times d \times \log_2(p) + \mu\log_2(t) + \kappa)/8\rceil + c2len$
return $(ct, clen)$
```

Algorithm 12: r5_cca_pke_decrypt$(ct, clen, sk)$

```
parameters: Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, b, \kappa, f, \tau$; $\xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$
input     : $ct = (c1||c2) \in (\mathcal{R}_{n,p}^{\overline{m}\times d/n} \times Z_t^{\mu} \times \{0,1\}^{\kappa}) \times \mathbb{Z}_{256}^{c2len}$,
            $clen = \lceil(\overline{m} \times d \times \log_2(p) + \mu\log_2(t) + \kappa)/8\rceil + c2len \in \mathbb{Z}, sk \in$
            $\{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \times (\{0,1\}^{\kappa} \times \mathcal{R}_{n,p}^{d/n\times\overline{n}})$
output    : $m \in \mathbb{Z}_{256}^{mlen}, mlen \in \mathbb{Z}$
$k$ = r5_cca_kem_decapsulate$(c1, sk)$
$(m, mlen) = \mathrm{DEM}^{-1}(k, c2)$
return $(m, mlen)$
```

2.4.7 Proposed parameter sets

Round5 proposes a unified scheme with a large design scheme – Table 1 – that can configured to fit the needs of different applications. In particular, Round5 optimizes over the above design space to define the following parameter sets:

- Six ring parameter sets (r5_cpa_kem and r5_cca_pke for NIST security levels 1,3 and 5) without error correction, see Tables 3 and 11. These parameter choices can be considered to be conservative, as they are only based on the Round2 design that has received public review since its submission.

- Six ring parameter sets (r5_cpa_kem and r5_cca_pke for NIST security levels 1,3 and 5) with XEf error correction code, see Tables 4 and 12. These parameter choices are based on the merge of HILA5 with Round2 and lead to the smallest public key and ciphertext sizes.

- Six non-ring parameter sets (r5_cpa_kem and r5_cca_pke for NIST security levels 1,3 and 5) without error correction, see Tables 5 and 13. These parameter choices rely on same design choices as the original Round2 submission. In particular, it uses $\overline{n} \approx \overline{m}$ to minimize total size of public-key plus ciphertext.

Round5 further details three additional specific use-case parameter sets with the only purpose of demonstrating its flexibility:

- A ring-based KEM parameter set with XE2 error correction, addressing IoT use cases. This parameter set has low bandwidth and computational requirements, yet still provides 88 bits of (quantum) security. See Table 6.

- A ring-based NIST level 1 KEM parameter set with XE4 error correction, for which the encapsulated key is 192-bit long instead of just 128-bit so that the difficulty of attacking the encapsulated key (by Grover) equals the difficulty of quantum lattice attack to Round5. See Table 6.

- A non-ring-based PKE parameter set with NIST level 3 with a ciphertext size of only 988 Bytes, with fast encryption, by taking $\overline{m} = 1$, at the cost of a larger public key. This parameter set targets applications in which the public-key can remain static for a long period, e.g., a fixed VPN between two end points. In such applications, the bandwidth footprint depends on the size of the ciphertext. Hence, this parameter set, despite enjoying the more conservative security assumptions for unstructured lattices, has a bandwidth requirement comparable to ring variants. See Table 14.

In contrast to the original Round2 and HILA5 submissions, Round5 does not include parameter sets suitable for NTT. The reason is that non-NTT parameters allow achieving better CPU-performance, as the advantages of using modular arithmetic with powers of two outweigh the advantages of using an NTT.

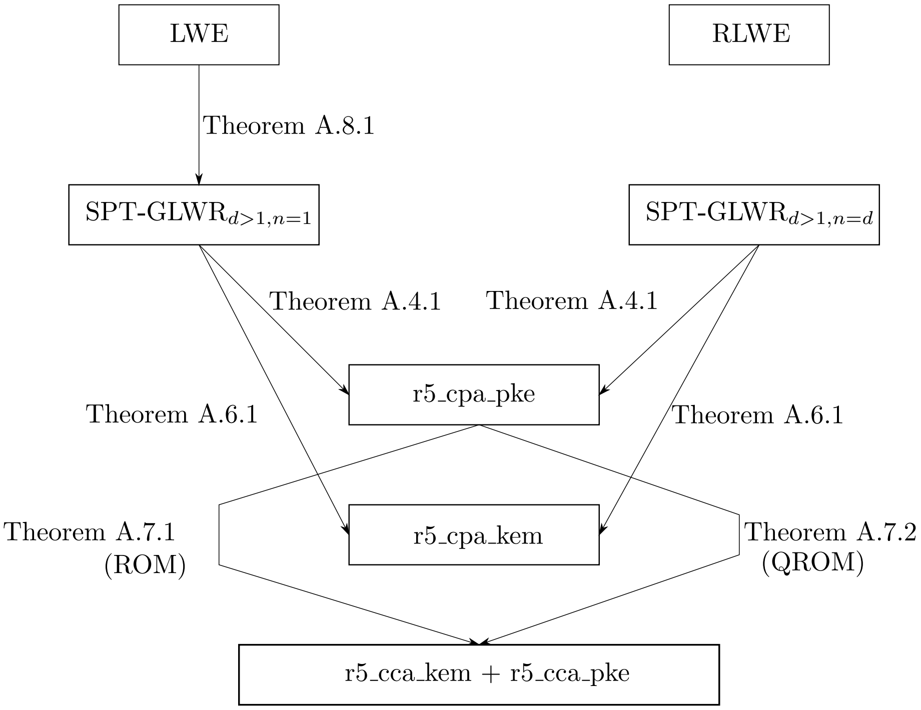

Figure 2: Summary of reductions involved in the security proofs for Round5 algorithms. "SPT" refers to a variant of the problem in question where the secret is sparse-ternary, instead of uniform in $\mathbb{Z}_q^d$.

2.5 Known Answer Test Values

The Known Answer Test Values for all algorithm variants and NIST levels can be found on the digital media at the location described in Section 3.4.

Note that for generating intermediate output, the code needs to be compiled with `-DNIST_KAT_GENERATION` (this option is enabled by default when making use of the provided `Makefile`s).

2.6 Expected Security Strength

This section summarizes results on the expected security of Round5 algorithms, its building blocks and underlying hard problem. An overview of these results is also given in Figure 2. For proofs, we refer to Appendix A.

1. **r5_cpa_pke** is an IND-CPA secure public-key encryption scheme if the decision General Learning with Rounding (GLWR) problem with sparse - ternary secrets is hard for the polynomial ring $\mathbb{Z}[x]/\Phi_{n+1}(x)$. Theorem A.4.1 gives a tight, classical reduction against classical or quantum adversaries in the standard model, a concise version of which is as follows:

Theorem 2.6.1. *When the parameter $n = 1$, or the reduction polynomial $\xi(x) = \Phi_{n+1}(x)$, for every adversary $\mathcal{A}$, there exist distinguishers $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$ such that*

$$\begin{aligned} Adv^{IND\text{-}CPA}_{CPA\text{-}PKE}(\mathcal{A}) \leq Adv^{f^{\tau}_{n,d}}_{\mathcal{U}(\mathcal{R}^{d/n \times d/n}_{n,q})}(\mathcal{B}) + Adv^{f_S(s)}_{\chi_S^{\overline{n}}}(\mathcal{C}) + \\ Adv^{f_R(\rho)}_{\chi_S^{\overline{m}}}(\mathcal{D}) + \overline{n} \cdot Adv^{dGLWR_{spt}}_{d,n,d/n,q,p}(\mathcal{E}) + Adv^{dGLWR_{spt}}_{d,n,d/n+\overline{n},q,z}(\mathcal{F}) \end{aligned} \tag{8}$$

where $z = \max(p, tq/p)$.

The full proof of IND-CPA security, given in Appendix A.4, follows a similar approach as [44] to equalize the noise ratios q/p and p/t in the two ciphertext components $\boldsymbol{U}$ and $\boldsymbol{v}$ that allows their combination into a single GLWR sample with noise ratio q/z. For the reduction to be meaningful, it is crucial to have $\mathrm{Adv}^{f^{\tau}_{n,d}}_{\mathcal{U}(\mathcal{R}^{d/n \times d/n}_{n,q})}(\mathcal{B})$ negligible; this is only proved for $\tau = 0$.

2. Appendix A.5 presents a proof of IND-CPA security for a Ring LWE based variant of r5_cpa_pke with reduction polynomial $\xi(x) = N_{n+1}(x)$, if the decision Ring LWE problem for $\mathbb{Z}[x]/\Phi_{n+1}(x)$ is hard; this results gives confidence for the RLWR case. Theorem A.5.1 gives a tight, classical reduction against classical and quantum adversaries in the standard model, a concise version of which is as follows:

   Theorem 2.6.2. *For every adversary $\mathcal{A}$, there exist distinguishers $\mathcal{C}$ and $\mathcal{E}$ such that*

   $$Adv^{IND\text{-}CPA}_{CPA\text{-}PKE,N_{n+1}(x)}(\mathcal{A}) \leq Adv^{RLWE(\mathbb{Z}[x]/\phi(x))}_{m=1}(\mathcal{C}) + Adv^{RLWE(\mathbb{Z}[x]/\phi(x))}_{m=2}(\mathcal{E}). \tag{9}$$

   where m denotes the number of available RLWE samples available.

   This section also gives indications on sufficient conditions for the proof to work. One of them is replacing coordinate-wise rounding as done in Round5 by a more complicated form of rounding which ensures that the sum of the errors induced by rounding sum to zero. We chose not to use this form of rounding as it adds complexity and seems not to improve practical security. Moreover, the Sample$_\mu$ function in Round5 stops a well- known distinguishing attack against schemes based on rings with reduction polynomial $x^{n+1} - 1$.

3. **r5_cpa_kem** is an IND-CPA secure KEM if r5_cpa_pke is an IND-CPA secure public-key encryption scheme, and that H is a secure pseudo-random function. Theorem A.6.1 gives a tight, classical reduction against classical or quantum adversaries in the standard model; the construction and proof technique are standard.

4. **r5_cca_pke** is constructed from r5_cca_kem and a one-time data encapsulation mechanism in the canonical way proposed by Cramer and Shoup [42].

Therefore, if r5_cca_kem is IND-CCA secure, and the data encapsulation mechanism is (one-time) secure against chosen ciphertext attacks and has a keylength fitting the security level of r5_cca_kem, then r5_cca_pke is an IND-CCA secure PKE. Section A.7 contains details.

5. **r5_cca_kem** is constructed using a KEM variant of the Fujisaki-Okamoto transform [58] from r5_cpa_pke. Therefore, assuming that r5_cpa_pke is an IND-CPA secure public-key encryption scheme, and G and H are modeled as random oracles, r5_cca_kem is an IND-CCA secure KEM. Theorem A.7.1 gives a tight, classical reduction against classical adversaries in the classical random oracle model. Theorem A.7.2 gives a non-tight, classical reduction against quantum adversaries in the quantum random oracle model.

6. Appendix A.1 gives an overview of the security reduction when replacing the GLWR public parameter $\boldsymbol{A}$ sampled from a truly uniform distribution with one generated in a pseudorandom fashion using the function $f_{d,n}^{(0)}$. The reduction models AES(128 or 256) as an ideal cipher, and SHAKE(128 or 256) as a random oracle.

7. The decision Learning with Rounding (LWR) problem with sparse-ternary secrets is hard if the decision Learning with Errors (LWE) problem with uniform secrets and Gaussian errors is hard. Theorem A.8.1 gives a classical reduction against classical or quantum adversaries in the standard model, under the condition that the noise rate in the LWE problem decreases linearly in the security parameter n of the LWE and LWR problems. A concise version of this theorem is presented below:

   Theorem 2.6.3. *Let p divide q, and $k \geq \frac{q}{p} \cdot m$. Let $\epsilon \in (0, \frac{1}{2})$, and $\alpha, \delta > 0$ such that*

   $$\alpha \geq q^{-1}\sqrt{(2/\pi)\ln(2n(1+\epsilon^{-1}))},\quad \binom{n}{h}2^h \geq q^{k+1} \cdot \delta^{-2},\ \textit{and } m = O(\frac{\log n}{\alpha\sqrt{10h}})$$

   Then there is a reduction from $\mathtt{dLWE}_{n,\frac{q}{p},q,D_{\alpha\sqrt{10h}}}(\mathcal{U}(\mathcal{H}_n(h)))$ to $\mathtt{dLWR}_{n,m,q,p}(\mathcal{U}(\mathcal{H}_n(h)))$.

The above summary of the proofs shows that the overall design of Round5 family of algorithms stems from a provable secure design. These proofs do not give any guidance to parameter choices. As is common practice in practical lattice based cryptography, we derive parameters from best known attacks.

2.7 Analysis with respect to known attacks

In this section, we analyze the concrete security of Round5. As the lattice-based attacks all rely on estimating the costs of solving certain lattice problems, we start with preliminaries on this topic in Sections 2.7.1 and 2.7.2. We then consider attacks using lattice basis reduction in Sections 2.7.3, 2.7.4 and 2.7.5, followed by specialized attacks that exploit sparse-ternary secrets used in Round5 in Sections 2.7.6 and 2.7.7. Finally, in Section 2.7.8 we consider precomputation and back-door attacks against the Round5 GLWR public parameter $\boldsymbol{A}$.

2.7.1 Preliminaries: SVP Complexities

On arbitrary lattices, we define the Shortest Vector Problem (SVP) as the problem of finding a vector in a lattice whose Euclidean norm is minimal among all non-zero vectors in the lattice. This problem is known to be NP-hard for randomized reductions [2, 66], and we do not expect any polynomial time algorithms for solving this problem in the worst-case to be found any time soon.

Various algorithms have been studied for solving SVP both in the worst-case and for average-case instances. The algorithm with the current best worst-case (asymptotic) complexity for solving SVP is the discrete Gaussian sampling approach [1], with an asymptotic cost in dimension n of $2^{n+o(n)}$ time and space. For large dimensions, this beats other exponential time and space algorithms based on computing the Voronoi cell [98, 76] or running a lattice sieve [3, 85], and enumeration algorithms requiring superexponential time in the lattice dimension [84, 63, 97, 51, 11]. These worst-case analyses however seem to suffer heavily from the (potential) existence of exotic, highly dense lattices such as the Leech lattice [38], and for average-case instances one can often solve SVP much faster, as also shown by various experiments on random lattices [45].

By making heuristic assumptions on the expected average-case behavior of SVP algorithms on random lattices, various works have demonstrated that some algorithms can solve SVP much faster than the above worst-case bounds suggest. For a long time, lattice enumeration [51, 11] was considered the fastest method in practice for solving SVP in high dimensions, and the crossover point with sieving-based approaches appeared to be far out of reach, i.e. well beyond cryptographically relevant parameters [77]. Recently, improvements to lattice sieving have significantly weakened this view, and the current fastest implementation based on sieving [6] has surpassed the best previous enumeration implementation both in terms of speed and in terms of the highest dimension reachable in practice. Although sieving is hindered by its exponential memory footprint, sieving also scales better in high dimensions compared to enumeration in terms of the time complexity ($2^{\Theta(n)}$ for sieving versus $2^{\Theta(n \log n)}$ for enumeration).

Conservative estimates. To estimate the actual computational complexity of solving SVP, we will use the best asymptotics for sieving [20], which state that solving SVP in dimension n takes time $2^{0.292n+o(n)}$. Clearly the hidden order term in the exponent has a major impact on the actual costs of these methods, and one might therefore be tempted to choose this term according to the actual, experimental value. However, recent improvements have shown that although the leading term $0.292n$ in the exponent is unlikely to be improved upon any time soon [10, 55], the hidden order term may well still be improved with heuristic tweaks such as those discussed in [46, 69, 6]. A conservative and practical cost estimate is therefore to estimate the costs of solving SVP in dimension n as $2^{0.292n}$, ignoring the (positive) hidden order term which may still be reduced over the next decades. Observe that this estimate is also well below the lowest estimates for enumeration or any other SVP method in high dimensions.

Note that some work has also suggested that SVP is slightly easier to solve on structured, ideal lattices when using lattice sieving [62, 32, 21, 101]. However, these are all improvements reducing the time and/or space complexities by a small polynomial factor in the lattice dimension. Moreover, in our cost estimates we will actually be using an SVP algorithm as a subroutine within a blockwise lattice reduction algorithm, and even if the original lattice on which we run this lattice basis reduction algorithm has additional structure, these low-dimensional sublattices do not. Therefore, even for ideal lattices we do not expect these polynomial speedups to play a role.

Quantum speedups. For making Round5 post-quantum secure, we also take into account any potential quantum speedups to attacks an adversary might run against our scheme. Concretely, for the hardness of SVP on arbitrary lattices this may reduce the cost of lattice sieving to only $2^{0.265n+o(n)}$, assuming the attacker can efficiently run a Grover search [52] on a dynamically changing, exponential-sized database of lattice vectors stored in quantum RAM [70, 68]. Although it may not be possible to ever carry out such an attack efficiently, a conservative estimate for the quantum hardness of SVP in dimension n is therefore to assume this takes the attacker at least $2^{0.265n}$ time [68].

Note that recent work has also indicated that enumeration may benefit from a quantum speedup when applying a Grover-like technique to improve the backtracking search of lattice enumeration [12]. Disregarding significant polynomial overhead as well as simply the overhead of having to do operations quantumly, a highly optimistic estimate for the time costs of quantum enumeration in dimension n is $2^{\frac{1}{2}(0.187n\log n-1.019n+16.1)}$ [61]. Even with this model, the attack costs for all our parameter sets are higher than $2^{0.265n}$.

2.7.2 Lattice basis reduction: the core model

The above discussion focused on algorithms for solving exact SVP in high dimensions. The concrete security of lattice-based cryptographic schemes like Round5 however relies on the hardness of unique/approximate SVP, where a break constitutes finding a vector not much longer than the actual shortest non-zero vector in the lattice. This problem has also been studied extensively, and the fastest method known to date for approximate SVP is the BKZ algorithm [95, 96, 35, 78, 6]. Given an arbitrary basis of a lattice, this algorithm (a generalization of the LLL algorithm [72]) performs HKZ reduction [67] on (projected) parts of the basis of a certain block size b. To perform HKZ reduction, the algorithm makes calls to an exact SVP oracle on b-dimensional lattices, and uses the result to improve the basis. Both the quality and time complexity can be tuned by b: running BKZ with larger b gives better bases, but also takes longer to finish.

Although the exact overall time complexity of BKZ remains somewhat mysterious [35, 54, 78, 6], it is clear that the dominant cost of BKZ for large block sizes is running the SVP oracle on b-dimensional lattices. The BKZ algorithm further makes tours through the entire basis on overlapping blocks of basis vectors, but the number of tours only contributes a small multiplicative term, and SVP calls on overlapping blocks do not necessarily add up to independent SVP calls - for multiple overlapping blocks, it may well be possible to solve SVP in almost the same time as solving only one instance of SVP when using a sieving-based SVP solver [6].

This all gives rise to the core SVP model, where the complexity of BKZ with block size b is modeled by just *one* call to an SVP oracle in a b-dimensional lattice. This is clearly a lower bound for the actual cost of BKZ, and as further advances may be made to sieving-based BKZ algorithms reusing information between blocks, this is also the largest lower bound we can use without risking that predicted future advances in cryptanalysis will reduce the security level of our scheme in the near future.

2.7.3 Lattice-based attacks

We consider lattice-reduction based attacks, namely, the *primal* or decoding attack [17] and the *dual* or distinguishing attack [4], and how they can be adapted to exploit the shortness of secrets in our schemes. We begin by detailing how an attack on Round5 can be formulated as a lattice-reduction based attack on the LWR problem. We then analyze the concrete security of Round5 against the primal attack in order to estimate secure parameters, in Section 2.7.4. We do the same for the dual attack in Section 2.7.5.

The attacker can use the public keys $\boldsymbol{B} = \langle \lfloor \frac{p}{q} \langle \boldsymbol{AS} \rangle_q \rceil \rangle_p$ of the public-key encryption scheme or the key-encapsulation scheme to obtain information on the secret key $\boldsymbol{S}$. We work out how this is done. Let $1 \leq i \leq d$ and $1 \leq j \leq \overline{n}$. If we denote the i-th row of $\boldsymbol{A}$ by $\boldsymbol{a}_i^T$ and the j-th column of $\boldsymbol{S}$ by $\boldsymbol{s}_j$, then

$$\boldsymbol{B}_{i,j} = \langle \lfloor \frac{p}{q} \langle \langle \boldsymbol{a}_i^T \boldsymbol{s}_j \rangle_q \rangle_q \rceil \rangle_p = \langle \lfloor \frac{p}{q} \langle \boldsymbol{a}_i^T \boldsymbol{s}_j \rangle_q \rceil \rangle_p.$$

By the definition of the rounding function $\lfloor \cdot \rceil$, we have that

$$\boldsymbol{B}_{i,j} \equiv \frac{p}{q} \langle \boldsymbol{a}_i^T \boldsymbol{s}_j \rangle_q + e_{i,j} \pmod{p} \text{ with } e_{i,j} \in (-1/2, 1/2].$$

As $\langle \boldsymbol{a}_i^T \boldsymbol{s}_j \rangle_q = \boldsymbol{a}_i^T \boldsymbol{s}_j + \lambda q$ for some integer λ, we infer that

$$\frac{q}{p} \boldsymbol{B}_{i,j} \equiv \boldsymbol{a}_i^T \boldsymbol{s}_j + \frac{q}{p} e_{i,j} \pmod{q}. \tag{10}$$

So we have d equations involving $\boldsymbol{s}_j$. Unlike conventional LWE, the errors $\frac{q}{p} e_{i,j}$ reside in a bounded interval, namely $(-\frac{q}{2p}, \frac{q}{2p}]$. In what follows, we will only consider the case that p divides q.

2.7.4 Primal Attack

In (10), we write $\boldsymbol{s}$ for $\boldsymbol{s}_j$, denote by $\boldsymbol{b}$ the vector of length m with j-th component $\frac{q}{p}\boldsymbol{B}_{i,j}$, and with $\boldsymbol{A}_m$ the matrix consisting of the m top rows of $\boldsymbol{A}$. We then have, for $\boldsymbol{e} \in (-\frac{q}{2p}, \frac{q}{2p}]^m$

$$\boldsymbol{b} \equiv \boldsymbol{A}_m \boldsymbol{s} + \boldsymbol{e} \pmod q \tag{11}$$

so that $\boldsymbol{v} = (\boldsymbol{s}^T, \boldsymbol{e}^T, 1)^T$ is in the lattice Λ defined as

$$\Lambda = \{\boldsymbol{x} \in \mathbb{Z}^{d+m+1} : (\boldsymbol{A}_m | \boldsymbol{I}_m | -\boldsymbol{b})\boldsymbol{x} = \boldsymbol{0} \pmod q\} \tag{12}$$

of dimension $d' = d + m + 1$ and volume q^m [28, 8].

Lattice Rescaling: The attacker wishes to recover the lattice vector $\boldsymbol{v} = (\boldsymbol{s}, \boldsymbol{e}, 1)$, which is unbalanced in that $\|\boldsymbol{s}\| \ll \|\boldsymbol{e}\|$. For exploiting this fact, a rescaling technique originally due to [17], and analyzed further in [37] and [4], is applied. Multiplying the first d columns of Λ's basis (see Eq. 12) with an appropriate scaling factor ω yields the following weighted or rescaled lattice,

$$\Lambda_\omega = \{\boldsymbol{x} \in \mathbb{Z}^{d+m+1} : \big((\omega \cdot \boldsymbol{A}_m^T) | \boldsymbol{I}_m | -\boldsymbol{b}\big)\boldsymbol{x} = \boldsymbol{0} \text{ (mod q)}\} \tag{13}$$

in which the attacker then searches for a short vector, that he hopes to be equal to $\boldsymbol{v}_\omega = (\omega \cdot \boldsymbol{s}^T, \boldsymbol{e}^T, 1)^T$. This search is typically done by using a lattice reduction algorithm to obtain a reduced basis of the lattice, the first vector of which will be the shortest of that basis. We explain later in this section how to choose an appropriate value for ω in order to maximize the chances of the attack's success.

If the quality of the lattice reduction is good enough, the reduced basis will contain $\boldsymbol{v}_\omega$. The attack success condition is as in [8] assuming that BKZ [35, 96] with block-size b is used as the lattice reduction algorithm. The vector $\boldsymbol{v}_\omega$ will be detected if its projection $\widetilde{\boldsymbol{v}}_b$ onto the vector space of the last b Gram-Schmidt vectors of Λ is shorter than the expected norm of the $(d'-b)^{th}$ Gram-Schmidt vector $\widetilde{\boldsymbol{b}}_{d'-b}$, where d' is the dimension of Λ [8, Sec. 6.3],[28]. The condition to be satisfied for the primal attack to succeed is therefore $\|\widetilde{\boldsymbol{v}}_b\| < \|\widetilde{\boldsymbol{b}}_{d'-b}\|$, i.e.

$$\|\widetilde{\boldsymbol{v}}_b\| < \delta^{2b-d'-1} \cdot (\mathrm{Vol}(\Lambda))^{\frac{1}{d'}}, \text{ where } \delta = ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi \mathbf{e}})^{\frac{1}{2(b-1)}}. \tag{14}$$

The attack success condition (14) yields the following *security* condition that must be satisfied by the parameters of our public-key encryption and key-encapsulation schemes to remain secure against the primal attack:

$$\sqrt{(\omega^2 \cdot h + \sigma'^2 m) \cdot \frac{b}{d+m}} \geq \delta^{2b-d'-1} \cdot (q^m \omega^d)^{\frac{1}{d'}}, \tag{15}$$

$$\text{where } \delta = ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi \mathbf{e}})^{\frac{1}{2(b-1)}}, \sigma'^2 = ((q/p)^2 - 1)/12, \text{and } d' = d + m + 1.$$

For finding an appropriate value for ω, we rewrite (15) as

$$\delta^{2b-d'-1}b^{-1/2} \leq \sqrt{\omega^2 h + m\sigma'^2 \cdot \frac{1}{m+d}}\omega^{-d/d'}q^{-(d-d'-1)/d}. \tag{16}$$

Given d, m, h and σ', the attacker obtains the least stringent condition on the block size b by maximizing the right hand side 16 over ω, or equivalently, by maximizing

$$\frac{1}{2}\log(\omega^2 h + m\sigma'^2) - \frac{d}{d'}\log \omega.$$

By differentiating with respect to ω, we find that the optimizing value for ω satisfies

$$\omega^2 = \frac{dm\sigma'^2}{h(d'-d)} = \frac{dm\sigma'^2}{h(m+1)} \approx \frac{d}{h}\sigma'^2.$$

2.7.5 Dual Attack

The dual attack against LWE/LWR [8],[4] employs a short vector $(\boldsymbol{v}, \boldsymbol{w}) \in \mathbb{Z}^m \times \mathbb{Z}^d$ in the dual lattice

$$\Lambda^* = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}^m \times \mathbb{Z}^d : \boldsymbol{A}_m^T \boldsymbol{x} = \boldsymbol{y} \pmod q\} \tag{17}$$

The attacker constructs the distinguisher using $z = \{\boldsymbol{v}^T\boldsymbol{b}\}_q$. If $\boldsymbol{b}$ is an LWR samples, then $z = \{\boldsymbol{v}^T(\boldsymbol{A}_m\boldsymbol{s} + \boldsymbol{e})\}_q = \{\boldsymbol{w}^T\boldsymbol{s} + \boldsymbol{v}^T\boldsymbol{e}\}_q$. Note that $\|\boldsymbol{s}\| \ll \|\boldsymbol{e}\|$ in our case, the attacker can enforce that $\|\boldsymbol{w}\| \gg \|\boldsymbol{v}\|$ to ensure that $|\boldsymbol{w}^T\boldsymbol{s}| \approx |\boldsymbol{v}^T\boldsymbol{e}|$ similar to [4]. He does so by choosing $\omega = \sigma'\sqrt{m/h}$ (for the LWR rounding error with variance $\sigma'^2 = ((q/p)^2 - 1)/12$), and considering the lattice:

$$\Lambda_\omega^* = \{(\boldsymbol{x}, \boldsymbol{y}/\omega) \in \mathbb{Z}^m \times (\frac{1}{\omega} \cdot \mathbb{Z}^d) : \boldsymbol{A}_m^T\boldsymbol{x} = \boldsymbol{y} \pmod q\} \tag{18}$$

A short vector $(\boldsymbol{v}, \boldsymbol{w}) \in \Lambda_\omega^*$ gives a short vector $(\boldsymbol{v}, \omega\boldsymbol{w}) \in \Lambda^*$ that is used to construct the distinguisher z. If $\boldsymbol{b}$ is uniform modulo q, so is z. If $\boldsymbol{b}$ is an LWR sample, then $z = \{\boldsymbol{v}^T\boldsymbol{b}\}_q = \{(\omega\boldsymbol{w})^T\boldsymbol{s} + \boldsymbol{v}^T\boldsymbol{e}\}_q = \{\boldsymbol{w}^T(\omega\boldsymbol{s}) + \boldsymbol{v}^T\boldsymbol{e}\}$ has a distribution approaching a Gaussian of zero mean and variance $\|(\boldsymbol{v}, \boldsymbol{w})\|^2 \cdot \sigma'^2$ as the lengths of the vectors increase, due to the Central limit theorem. Note that ω has been chosen such that $\|\omega\boldsymbol{s}\| \approx \|\boldsymbol{e}\|$. The maximal statistical distance between this and the uniform distribution modulo q is bounded by $\epsilon \approx 2^{-1/2}\exp(-2\pi^2(\|(\boldsymbol{v}, \boldsymbol{w})\|^2 \cdot \sigma'/q)^2)$ [23, Appendix B]. Lattice reduction with root-Hermite factor δ yields a shortest vector of length $\delta^{d'-1} \cdot \mathrm{Vol}(\Lambda_\omega^*)^{1/d'}$, where $d' = m + d$ and $\mathrm{Vol}(\Lambda_\omega^*) = (q/\omega)^d$ are Λ_ω^*'s dimension and volume, respectively.

However, finding only one such short vector is not enough, as the resulting distinguishing advantage ϵ is too small to distinguish a final key which is hashed. The attack needs to be repeated approximately ϵ^{-2} times to achieve constant advantage. The attacker can utilize the fact that when sieving algorithms are used in the lattice reduction algorithm (e.g., BKZ with block size b) to obtain short vectors, each run of such algorithms provides $2^{0.2075b}$ vectors [8]. Under the conservative assumption that each of these vectors is short enough to guarantee a distinguishing advantage ϵ, the lattice reduction algorithm must therefore be run at least $\max(1, 1/2^{0.2075b} \cdot \epsilon^2)$ times [8], when considering BKZ with block size b. The cost of the weighted dual attack on LWR (with dimension d, large modulus q, rounding modulus p) using m samples thus is the cost of running one instance of BKZ lattice reduction with block size b times the number of repetitions required, i.e.,

$$(b \cdot 2^{cb}) \cdot \max(1, 1/(\epsilon^2 \cdot 2^{0.2075 \cdot b})), \text{where} \tag{19}$$

$$\epsilon = 2^{-1/2} \cdot \mathbf{e}^{-2\pi^2\left(\left(\|(\boldsymbol{v},\boldsymbol{w})\|^2 \cdot \sigma'\right)/q\right)^2},\ \|(\boldsymbol{v},\boldsymbol{w})\|^2 = \delta^{m+d-1} \cdot (q/\omega)^{d/(m+d)},$$

$$\delta = ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi\mathbf{e}})^{\frac{1}{2(b-1)}},\ \omega = \sigma' \cdot \sqrt{m/h},\ \text{and } {\sigma'}^2 = ((q/p)^2 - 1)/12.$$

The first term $(b \cdot 2^{cb})$ in the overall attack cost is that of running BKZ with block-size b. The BKZ sieving exponent c depends on the security model, e.g., $c = 0.265$ considering sieving algorithms sped up with Grover's quantum search algorithm [68, 70], and $c = 0.292$ with no such speedup.

For Round5's ring-based instantiations (i.e., when the parameters $n = d$), we follow a conservative approach similar to [8] and omit the b factor in the cost of running one instance of BKZ, accounting for the possibility that techniques in [94, 32] can be adapted to more advanced sieving algorithms, yielding the cost 2^{cb} instead of $b \cdot 2^{cb}$.

2.7.6 Hybrid Attack

In this section, we consider a hybrid lattice reduction and meet-in-the-middle attack (henceforth called *hybrid attack*) originally due to [60] that targeted the NTRU [56] cryptosystem. We first describe the hybrid attack, using notation similar to that of [100], in a general form. Subsequently, we specialize the attack to our scheme. Finally, we describe a scaling approach to make the hybrid attack exploit the fact that the secret is small and sparse.

The hybrid attack will be applied to the lattice

$$\Lambda' = \{\boldsymbol{x} \in \mathbb{Z}^{m+d+1} \mid (\boldsymbol{I}_m | \boldsymbol{A}_m | -\boldsymbol{b})\boldsymbol{x} \equiv 0 \bmod q\}$$

for some $m \in \{1, \ldots, d\}$. We first find a basis $\boldsymbol{B'}$ for Λ' of the form

$$\boldsymbol{B'} = \begin{pmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{I}_r \end{pmatrix} \tag{20}$$

where $0 < r < d$ is the meet-in-the-middle dimension and $\boldsymbol{I}_r$ is the r-dimensional identity matrix. We aim to find the short vector $\boldsymbol{v} = (\boldsymbol{e}^T, \boldsymbol{s}^T, 1)^T$ in Λ'. We write $\boldsymbol{v} = (\boldsymbol{v}_l^T \boldsymbol{v}_g^T)^T$ where $\boldsymbol{v}_g$ has length r. We recover the vector $\boldsymbol{v}_g$ of length $r < d$ consisting of the $(r-1)$ bottom entries of $\boldsymbol{s}$ followed by a '1' by guessing. As the columns of $\boldsymbol{B'}$ generate Λ', there exists a $\boldsymbol{x} \in \mathbb{Z}^{d+m+1-r}$ such that

$$\boldsymbol{v} = \begin{pmatrix} \boldsymbol{v}_l \\ \boldsymbol{v}_g \end{pmatrix} = \boldsymbol{B'} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{v}_g \end{pmatrix} = \begin{pmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{I}_r \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{v}_g \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}\boldsymbol{x} + \boldsymbol{C}\boldsymbol{v}_g \\ \boldsymbol{v}_g \end{pmatrix} \tag{21}$$

As $\boldsymbol{v}_l$ is short, $\boldsymbol{C}\boldsymbol{v}_g$ is close to $-\boldsymbol{B}\boldsymbol{x}$, a vector from the lattice $\Lambda(\boldsymbol{B})$. As explained in [100], the idea is that if we correctly guess $\boldsymbol{v}_g$, we can hope to find $\boldsymbol{v}_l$ by using Babai's Nearest Plane (NP) algorithm [16]. This algorithm, given a basis $\tilde{\boldsymbol{B}}$, finds for every target vector $\boldsymbol{t} \in \mathbb{R}^{d+m+1-r}$ a vector $\boldsymbol{e} = \mathrm{NP}_{\tilde{\boldsymbol{B}}}(\boldsymbol{t})$ such that $\boldsymbol{t} - \boldsymbol{e} \in \Lambda(\tilde{\boldsymbol{B}})$.

The cost for the hybrid attack thus is the sum of two terms: the cost of finding a good basis $\tilde{\boldsymbol{B}}$ for $\Lambda(\boldsymbol{B})$, and the cost of generating $\mathrm{NP}_{\tilde{\boldsymbol{B}}}(\boldsymbol{C}\boldsymbol{y})$ for all $\boldsymbol{y}$ from a set of vectors of length r that (with high probability) contains $\boldsymbol{v}_g$. The latter cost may be reduced by using a Meet-in-the-middle approach [60] which reduces the number of calls to the Nearest Plane algorithm to the square root of the number of calls with a brute-force approach.

As $r < d$, the vector $\boldsymbol{v}_g$ is a ternary. The attacker can benefit from the fact that $\boldsymbol{s}$ has h non-zero entries in the generation of candidates for $\boldsymbol{v}_g$: candidates with high Hamming weight are not very likely. Also, as the $(d{-}r)$ bottom entries of $\boldsymbol{v}_l$, being the $(d-r)$ top elements of $\boldsymbol{s}$, are ternary and sparse. In order to benefit from this fact, the $d{-}r$ rightmost columns of the matrix $\boldsymbol{B}$ are multiplied with an appropriate scaling factor ω. Calculated similarly to Section 2.7.4 by equalizing the *per-component* expected norms of the secret $\boldsymbol{s}$ and LWR rounding error $\boldsymbol{e}$, we arrive at the same scaling factor $\omega = \sqrt{\frac{(q/p)^2-1}{12} \cdot \frac{d}{h}}$. This then scales up the volume of the $(d - r + m + 1$ dimensional) lattice Λ generated by $\boldsymbol{B}$ by a factor ω^{d-r}.

We analyze the hybrid attack similarly as in [56]. For each pair (r, m) with $1 \leq r, m < d$, we stipulate that the quality of the reduced basis $\tilde{\boldsymbol{B}}$ is so high that $\mathrm{NP}_{\tilde{\boldsymbol{B}}}(\boldsymbol{v}_g) = \boldsymbol{v}_l$ with high probability. The condition, derived from [60, Lemma 1]] is that the norm of the last Gram-Schmidt vector of $\tilde{\boldsymbol{B}}$ is at least twice $||\boldsymbol{v}_l||_\infty$, see also [56] We use the Geometric Series Assumption to approximate the norms of these vectors in terms of the Hermite constant δ. The cost for obtaining a reduced basis with Hermite constant δ is estimated as $b2^{cb}$, where b is such that $\delta = \left((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi e}\right)^{\frac{1}{2b-1}}$, and similar to our conservative approach in the previous attacks we ignore the additional b factor in the cost accounting for potential speedups in sieving algorithms for ring-based instantiations. c is the BKZ sieving exponent as before. Moreover, we estimate the cost for the lattice decoding part to be equal to the number of invocations of the Nearest Plane Algorithm, which, following [56], we set to $2^{\frac{1}{2}r \cdot H}$, where H is the entropy of the distribution of each of the coordinates of the guessed vector $\boldsymbol{v}_g$. The number $2^{r \cdot H}$ approximates the number of typical vectors of length r; the factor $\frac{1}{2}$ is due to either the usage of the MITM technique, or the use of Grover's algorithm in the quantum case. Finally, we minimize the cost over all feasible pairs (r, m).

Our analysis of the hybrid attack allows us to obtain a rough estimate of its cost. With the goal of providing a more accurate estimate, Wunderer [100] gives an extensive runtime analysis of the hybrid attack. One of the aspects he takes into account is that the attacker chooses a larger value of δ. This leads to a smaller cost (running time) for lattice reduction to obtain $\tilde{\boldsymbol{B}}$, but decreases the probability that $\mathrm{NP}_{\tilde{\boldsymbol{B}}}(\boldsymbol{v}_g) = \boldsymbol{v}_l$, thereby increasing the expected cost (running time) of the part of the attack dealing with solving BDD problems. Also, he takes into account that the guesses for $\boldsymbol{v}_g$ are generated such that the most likely candidates for $\boldsymbol{v}_g$ occur early, thus reducing the expected number of calls to the nearest plane algorithm.

2.7.7 Attacks against Sparse Secrets

In Sections 2.7.4 and 2.7.5, we considered attacks [17, 4, 37] against LWE and/or LWR variants with unnaturally small secrets. In this section, we consider attacks against *sparse* secrets with the goal of choosing an appropriate Hamming weight providing both optimal performance and security. The best-known attacks against such sparse secrets are (to the best of our knowledge) the Hybrid attack described in Section 2.7.6, and another one due to Albrecht *et al* [4]. The Hybrid attack performs better than Albrecht's attack against our schemes, it is therefore the primary attack considered in our analysis. Recall that the hybrid attack's overall cost is the sum of two components: that of finding a good basis and that of solving Babai's Nearest Planes for a large set of vectors using a Meet-in-the-middle approach. Recall also that this second cost component depends on the entropy H of the distribution of each secret coordinate (in our case), which in turn depends on the Hamming weight of the secret. We therefore optimize over the Hamming weight to choose the smallest value for which the overall hybrid attack cost is at least a targeted minimum (depending on the security level).

For completeness, we also describe Albrecht's attack [4] against LWE/LWR variants with sparse secrets. Since most rows of the public matrix $\boldsymbol{A}$ become irrelevant in the calculation of the product $\boldsymbol{As}$ for such secrets, Albrecht's attack ignores a random set of $k \leq d$ components of $\boldsymbol{s}$ and brings down the lattice dimension (and hence attack cost) during lattice reduction. As $\boldsymbol{s}$ has $d-h$ zeroes, there are $\binom{d-h}{k}$ choices (out of $\binom{d}{k}$) for the k ignored components such that these ignored components only contain zeroes. We therefore repeat the attack $\binom{d}{k}/\binom{d-h}{k}$ times. We estimate the cost (in bits) for a given Hamming weight $h \leq d$, as the number of repetitions each low cost attack is performed times the cost of the low-cost attack on a lattice of dimension $d-k$:

$$\frac{\binom{d}{k}}{\binom{d-h}{k}} \times \mathrm{Cost}_{\text{Lattice Reduction}}(d,k,h)$$

Here $\mathrm{Cost}_{\text{Lattice Reduction}}(d,k,h)$ is defined as

$$\min\{b\cdot 2^{cb} \mid b \in \mathbb{N},\ \text{there exists } m \in \mathbb{N} \text{ such that } m \leq d \text{ and (22) is satisfied}\}.$$

$$\sqrt{(\omega^2 \cdot h + \sigma'^2 m)\cdot \frac{b}{d+m}} < \delta^{2b-d'-1}\cdot\left(q^{d'-(d-k)-1}\omega^{d-k}\right)^{\frac{1}{d'}} \tag{22}$$

$$\text{where } \delta = \left((\pi b)^{\frac{1}{b}}\cdot\frac{b}{2\pi\mathbf{e}}\right)^{\frac{1}{2(b-1)}}, \omega = \sigma'\cdot\sqrt{(d-k)/h},$$

$$\sigma'^2 = ((q/p)^2-1)/12,\ \text{and } d' = m+d-k+1.$$

The term $b \cdot 2^{cb}$ represents the cost of running BKZ lattice reduction with blocksize b (the first term or factor b in the cost is ignored for ring-based instantiations as in the previous attacks). The attack runs on a LWE problem of dimension $d - k$ with $m \leq d$ samples. Condition 22, which is essentially Condition 14, ensures that such an attack has chances of succeeding. Note that although the above applies to the primal attack, a similar analysis is possible for the dual attack, in which case $\text{Cost}_{\text{Lattice Reduction}}(d, k, h)$ is calculated as in Eq. 19, with the parameter d replaced by $d - k$.

This specialized attack only gives an advantage if an attacker is able to choose a k for which the total attack cost is less than a standard lattice-reduction attack on a lattice of dimension d. Similar to the case of the hybrid attack (Section 2.7.6), we optimize over the Hamming weight to choose the smallest value such that Albrecht et al.'s attack results in at least a minimum targeted security level (both for the standard attack embodiment mentioned above and an adaptive embodiment described in [4]).

Furthermore, to ensure that an exhaustive, brute-force search of each secret-key vector in the secret-key using Grover's quantum search algorithm [52] has a cost of at least λ (in bits), the chosen Hamming weight should satisfy:

$$\sqrt{\binom{d}{h} \cdot 2^h} > 2^\lambda \tag{23}$$

Note that for a typical security level of $\lambda = 128$, a dimension of at least $d = 512$ would be secure against Grover's quantum search, for any Hamming weight h that is at least $0.1d$.

2.7.8 Use of $f_{d,n}^{(\tau)}$ to stop Pre-computation and Back-door Attacks

Works prior to New Hope [8], like [29] and [81], defined key exchange protocols or key encapsulation protocols with a fixed public parameter $\boldsymbol{A}$ in the case of unstructured lattices, $\boldsymbol{a}$ in the case of structured lattices. NewHope [8] proposed generating $\boldsymbol{a}$ on the fly from a seed that needs to be exchanged. The rationale for having a fresh $\boldsymbol{A}$ or $\boldsymbol{a}$ includes preventing pre-computation and backdoor attacks. In a pre-computation attack on a system with fixed public parameter $\boldsymbol{A}$, the attacker performs lattice reduction on $\boldsymbol{A}$ over a long period of time. A back-door attack can be mounted if the fixed $\boldsymbol{A}$ has deliberately been chosen so as to result in a lattice with weak security.

Round5 addresses these attacks and performance issues by generating a fresh $\boldsymbol{A}$ based on function $f_{d,n}^{(\tau)}$ (see Section 2.4.3) with $\tau \in \{0, 1, 2\}$. For its ring instantiations, Round5 only defines $f_{d,n}^{(0)}$, as this solution is sufficiently fast and memory efficient. For the non-ring instatiations, all three options $f_{d,n}^{(\tau)}$ with $\tau \in \{0, 1, 2\}$ can be applied.

$\boldsymbol{f_{d,n}^{(0)}}$, Non-ring and ring instantiations. With this function, a fresh $\boldsymbol{A}$ is generated in each protocol instantiation, similar to [28] and [8]. This prevents both pre-computation attacks and backdoors since the whole matrix is derived from a fresh, random seed by means of a deterministic random bit generator (DRBG). This option performs well in the ring case, but for the non-ring case it requires the generation of a high amount of pseudorandom data, between 0.75 MByte and 2.5 MByte for typical parameters. This motivates the usage of $f_{d,n}^{(1)}$ and $f_{d,n}^{(2)}$.

$\boldsymbol{f_{d,n}^{(1)}}$, Non-ring instantiation. With $f_{d,n}^{(1)}$, the matrix $\boldsymbol{A}$ is derived from a fixed long-term system-wide matrix $\boldsymbol{A}_{\text{master}} \in \mathbb{Z}_q^{d\times d}$. This is done by applying to $\boldsymbol{A}_{\text{master}}$ a fresh permutation chosen by the initiator of the protocol at the start of each protocol exchange. The permutation consists in random, cyclic shifts of the rows of $\boldsymbol{A}_{\text{master}}$. Computing this permutation is cheap, and thus, this approach is efficient in terms of CPU overhead. Memory-wise, only $\boldsymbol{A}_{\text{master}}$ needs to be kept in memory and simple permutations of it can be used for connecting to multiple clients. This prevents any pre-computation attacks since the possible number of permuted versions of $\boldsymbol{A}_{\text{master}}$ obtained in this way equals d^d. Backdoors are avoided since $\boldsymbol{A}_{\text{master}}$ is derived by means of a pseudo-random function from a randomly generated seed, in other words, by using $f_{d,n}^{(\tau)}$ for $\tau = 0$. Proving formal security in terms of indistinguishability is not feasible if $(\boldsymbol{A}, \boldsymbol{B})$ is jointly considered as the public key.

$\boldsymbol{f_{d,n}^{(2)}}$, Non-ring instantiation. With $f_{d,n}^{(2)}$, $\boldsymbol{A}$ is obtained from a vector $\boldsymbol{a}_{\text{master}} \in \mathbb{Z}_q^{len\_tau\_2}$ that is randomly generated by means of a DBRG from a seed determined by the initiator in each protocol interaction. Each row in $\boldsymbol{A}$ is obtained from $\boldsymbol{a}_{\text{master}}$ by means of a random permutation that is also determined by the initiator and is specific to each protocol interaction, and ensures that $\boldsymbol{A}$ has distinct rows. Each row of $\boldsymbol{A}$ consists of d consecutive entries of $\boldsymbol{a}_{\text{master}}$. Since only a few – len_tau_2 – elements need to be generated and kept in memory, $f_{d,n}^{(2)}$ is efficient both in terms of memory and CPU consumption.

Since elements in $\boldsymbol{A}$ might correspond to the same entry of $\boldsymbol{a}_{\text{master}}$, the security of Round5 when employing $f_{d,n}^{(2)}$ cannot be based on the results from Section A.8. However, we argue that pre-computation and back-door attacks are avoided since the seed that determines $\boldsymbol{A}$ is new in each protocol interaction, and this approach allows computing many $\boldsymbol{A}$'s: $\boldsymbol{A}$ is obtained by permuting – there are a total of $\binom{len\_tau\_2}{d}$ permutations – a fresh vector $\boldsymbol{a}_{\text{master}}$ for which there are $q^{len\_tau\_2}$ choices.

We now analyze $f_{d,n}^{(2)}$ from a different perspective. If $len\_tau\_2 = d$, then $f_{d,n}^{(2)}$ can be considered as an equivalent description of an ideal lattice and the permutations used to compute $\boldsymbol{A}$ are equivalent to those that define a (anti-) cyclic matrix. Next, assume that $len\_tau\_2 > d$. We say that two rows of $\boldsymbol{A}$ overlap if they have have at least one entry originating from the same entry of $\boldsymbol{a}_{\text{master}}$. We now consider two rows of $\boldsymbol{A}$.

If the two rows do not overlap, then we argue that for those two rows, $f_{d,n}^{(2)}$ is equivalent to $f_{d,n}^{(0)}$ since all entries of those rows have been obtained by applying a DRBG to a seed.

Now we consider two overlapping rows of $\boldsymbol{A}$ that share $d-1-k$ elements, namely $a = \langle a_0, a_1, a_2, \ldots, a_{d-1}\rangle$ and $b = \langle b_0, b_1, \ldots, b_{k-1}, a_0, a_1, a_2, \ldots, a_{d-k-1}\rangle$. If we define $a(x) := a_0 + a_1x + a_2x^2 + \cdots + a_{d-1}x^{d-1}$ and $b(x) := b_0 + b_1x + \cdots + b_{k-1}x^{k-1} + a_0x^k + a_1x^{k+1} + \cdots + a_{d-k-1}x^{d-1}$, then we have $b(x) = a(x)x^k + (a_{d-k} + b_0) + (a_{n-k+1} + b_1)x + \cdots + (a_{d-1} + b_{k-1})x^{k-1} \bmod x^d + 1$. Effectively, that is, each row can be seen as using the x^d+1 ring with a random shifting (due to k) and additional random noises, those $(a_{d-k}+b_0)+(a_{d-k+1}+b_1)x+\cdots+(a_{d-1}+b_{k-1})x^{k-1}$. The random shift is due to the the random permutation and the random noises are due to the random generation of $\boldsymbol{a}_{\text{master}}$. We argue that this fact makes it harder to exploit any ring structure in the resulting $\boldsymbol{A}$ (as can be found in circulant or anti-circulant matrices for ideal lattices, for example). We also argue that this approach of generating $\boldsymbol{A}$ with $f_{d,n}^{(2)}$ is at least as secure as using the ring instantiation of Round5.

2.8 Correctness of Round5

The following subsections we first derive the condition for correct decryption in Round5. Subsequently, we work out this condition for non-ring parameters, ring parameters with reduction polynomial $\xi(x) = \Phi_{n+1}(x)$, and ring parameters with $\xi(x) = x^{n+1}-1$. The analysis for ring parameters with $\xi(x) = \Phi_{n+1}(x)$ show that decryption errors (before error correction) are correlated. For this reason, error correction in combination with reduction polynomials $\xi(x) = x^{n+1}-1$ is not effective. Finally, results from the decryption failure analysis are compared with simulations for scaled-down versions of Round5.

2.8.1 Decryption failure analysis

In this section, the decryption failure behavior of r5_cpa_pke is analyzed. In decryption, the vector $\boldsymbol{x'} = \langle\lfloor\frac{b}{p}\boldsymbol{\zeta}\rceil\rangle_b$ is computed, where

$$\boldsymbol{\zeta} = \left\langle \frac{p}{t}\boldsymbol{v} + h_3\boldsymbol{j} - \mathrm{Sample}_\mu\left(\left\langle \boldsymbol{S}^T(\boldsymbol{U} + h_4\boldsymbol{J})\right\rangle_\xi\right)\right\rangle_p .$$

First, a sufficient condition is derived so that $\boldsymbol{x'}$ and $\boldsymbol{x} = \mathit{xef\_compute}_{\kappa,f}(m)$ agree in a given position, where $\boldsymbol{x}$ is considered as a vector of μ symbols, each consisting of $\mathit{b\_bits}$ bits. We have that

$$\boldsymbol{v} \equiv \langle\frac{t}{p}\mathrm{Sample}_\mu(\langle\boldsymbol{B}^T\boldsymbol{R}\rangle_\xi + h_2\boldsymbol{j}) - \frac{t}{p}\boldsymbol{I}_v\rangle_p + \frac{t}{b}\boldsymbol{x} \pmod{t},$$

where $\frac{t}{p}\boldsymbol{I}_v$ is the error introduced by the rounding downwards, with each component of $\boldsymbol{I}_v$ in $\mathbb{Z}_{p/t}$. As a result,

$$\boldsymbol{\zeta} \equiv \frac{p}{b}\boldsymbol{x} + \boldsymbol{\Delta} \pmod{p} \text{ with } \boldsymbol{\Delta} = (h_2+h_3)\boldsymbol{j} - \boldsymbol{I}_v + \mathrm{Sample}_\mu(\langle\boldsymbol{B}^T\boldsymbol{R} - \boldsymbol{S}^T(\boldsymbol{U}+h_4\boldsymbol{J})\rangle_\xi). \tag{24}$$

As $\boldsymbol{x'} = \lfloor \frac{b}{p}\boldsymbol{\zeta} \rfloor = \lfloor \frac{b}{p}\boldsymbol{\zeta} - \frac{1}{2} \rceil$, we have that

$$\boldsymbol{x'} \equiv \boldsymbol{x} + \lfloor \frac{b}{p}\boldsymbol{\Delta} - \frac{1}{2}\boldsymbol{J} \rceil \equiv \boldsymbol{x} + \lfloor \frac{b}{p}\{\boldsymbol{\Delta} - \frac{p}{2b}\boldsymbol{J}\}_p \rceil \pmod{b}.$$

Here $\{y\}_p$ denotes the integer in $(-p/2, p/2]$ that is equivalent to y modulo p. As a consequence, $x_i = x'_i$ whenever $|\{\boldsymbol{\Delta}_i - \frac{p}{2b}\}_p| < \frac{p}{2b}$. We multiply both sides of the above inequality with $\frac{q}{p}$, and infer that $x_i = x'_i$ whenever

$$\mid \{\frac{q}{p}\boldsymbol{\Delta}_i - \frac{q}{2b}\}_q \mid < \frac{q}{2b}. \tag{25}$$

Equivalently, as $\frac{q}{p}\boldsymbol{\Delta}_i$ has integer components, if $x_i \neq x'_i$, then

$$\langle \frac{q}{p}\boldsymbol{\Delta}_i \rangle_q \in \left[\frac{q}{2b}, q - \frac{q}{2b}\right] \tag{26}$$

The probability that $\boldsymbol{x}$ and $\boldsymbol{x'}$ differ in position i thus is at most the probability that (26) is satisfied. In order to analyze this probability, we work out $\frac{q}{p}\boldsymbol{\Delta} - \frac{q}{2b}\boldsymbol{J}$, using (24).

We write $\boldsymbol{J}_v = \frac{q}{p}(h_2\boldsymbol{j} + h_3\boldsymbol{j} - \boldsymbol{I}_v - \frac{p}{2b}\boldsymbol{J})$. The definitions of h_2 and h_3 imply that $\boldsymbol{J}_v = \frac{q}{p}(\frac{p}{2t} - \boldsymbol{I}_v)$. Each component of $\boldsymbol{I}_v$ is in $\mathbb{Z}_{p/t}$. The value of h_3 thus ensures that the absolute value of each coefficient of $\frac{p}{2t} - \boldsymbol{I}_v$ is at most $\frac{p}{2t}$.
We now analyse $\frac{q}{p}\langle \boldsymbol{B}^T\boldsymbol{R} - \boldsymbol{S}^T(\boldsymbol{U} + h_4\boldsymbol{J})\rangle_\xi$. Similarly to the expression for $\boldsymbol{v}$, we write

$$\boldsymbol{B} = \left\langle \frac{p}{q}\left(\langle \boldsymbol{A}\boldsymbol{S}\rangle_{\Phi_{n+1}} + h_1\boldsymbol{J}\right) - \frac{p}{q}\boldsymbol{I}_B \right\rangle_p \text{ and } \boldsymbol{U} = \left\langle \frac{p}{q}(\langle \boldsymbol{A}^T\boldsymbol{R}\rangle_{\Phi_{n+1}} + h_2\boldsymbol{J}) - \frac{p}{q}\boldsymbol{I}_U \right\rangle_p,$$

with all components of $\boldsymbol{I}_B$ and $\boldsymbol{I}_U$ in $\mathbb{Z}_{q/p}$. We thus can write

$$\frac{q}{p}(\boldsymbol{B}^T\boldsymbol{R} - \boldsymbol{S}^T(\boldsymbol{U} + h_4\boldsymbol{J}) \equiv \langle \boldsymbol{S}^T\boldsymbol{A}^T\rangle_{\Phi_{n+1}}\boldsymbol{R} - \boldsymbol{S}^T\langle \boldsymbol{A}^T\boldsymbol{R}\rangle_{\Phi_{n+1}} + \boldsymbol{J}_B^T\boldsymbol{R} - \boldsymbol{S}^T\boldsymbol{J}_U \pmod{q} \tag{27}$$

where

$$\boldsymbol{J_B} = h_1\boldsymbol{J} - \boldsymbol{I}_B, \text{ and } \boldsymbol{J_U} = (h_2 + h_4)\boldsymbol{J} - \boldsymbol{I}_U. \tag{28}$$

As $h_1 = h_2 + h_4 = \frac{q}{2p}$, all entries of $\boldsymbol{J}_B$ and of $\boldsymbol{J}_U$ are from the set $I := (-\frac{q}{2p}, \frac{q}{2p}] \cap \mathbb{Z}$. The value of h_4 thus ensures that the absolute value of the entries of $\boldsymbol{J}_B$ and $\boldsymbol{J}_U$ are at most $\frac{q}{2p}$.

Clearly, if $\xi = \Phi_{n+1}$, then $\langle \boldsymbol{S}^T\boldsymbol{A}^T\rangle_{\Phi_{n+1}}\boldsymbol{R} \equiv \boldsymbol{S}^T\langle \boldsymbol{A}^T\boldsymbol{R}\rangle_{\Phi_{n+1}}$, so that

$$\frac{q}{p}\boldsymbol{\Delta} \equiv \boldsymbol{J}_v + \text{Sample}_\mu\left(\langle \boldsymbol{J}_B^T\boldsymbol{R} - \boldsymbol{S}^T\boldsymbol{J}_U\rangle_{\Phi_{n+1}}\right) \pmod{q} \tag{29}$$

In Section 2.8.4, the special case $\xi(x) = x^{n+1} - 1$ will be analyzed.

The probability that (26) is satisfied will be analyzed under the following assumptions: the entries $\boldsymbol{J}_v$ are drawn independently, and each such entry is distributed as $\frac{q}{p}y$ with y uniform on $(-\frac{p}{2t}, \frac{p}{2t}] \cap \mathbb{Z}$. The entries of $\boldsymbol{J}_B$ and $\boldsymbol{J}_U$ are drawn independently and uniformly from $I = (-\frac{q}{2p}.\frac{q}{2p}] \cap \mathbb{Z}$. In our analysis, we also assume that all columns of the secret matrices $\boldsymbol{S}$ and $\boldsymbol{R}$ have $h/2$ coefficients equal to 1 and $h/2$ coefficients equal to -1. This is true for the implementations of f_S and f_R.

2.8.2 Failure probability computation: non-ring parameters

In the non-ring case, each entry of $\boldsymbol{J}_B^T\boldsymbol{R}$ and of $\boldsymbol{S}^T\boldsymbol{J}_U$ is the inner product of a row of $\boldsymbol{J}_B^T$ (resp. a column of $\boldsymbol{J}_U$) and a ternary vector with $h/2$ entries equal to one and $h/2$ entries equal to minus one. Hence each entry of $\boldsymbol{J}_B^T\boldsymbol{R}-\boldsymbol{S}^T\boldsymbol{J}_U$ is distributed as the sum of h uniform variables on I minus the sum of h uniform variables on I. Assuming independence, the latter distribution (modulo q) can easily be computed explicitly (using repeated convolutions). Indeed, if c and d are two independent variables on $\{0,1,\ldots,q-1\}$ with probability distributions p_c and p_d, then the probability distribution p_e of $\langle c+d\rangle_q$ is given by

$$p_e(k)=\sum_{i=0}^{q-1}p_c(i)p_d(\langle k-i\rangle_q) \text{ for } 0\leq k\leq q-1.$$

Assuming independence between the i-th components of $\boldsymbol{J}_v$ and of $\text{Sample}_\mu(\boldsymbol{J}_B^T\boldsymbol{R}-\boldsymbol{S}^T\boldsymbol{J}_U)$, the probability that (26) is satisfied can be computed by using another convolution. By the union bound, the probability that at least one of the symbols of $\boldsymbol{x}$ is not retrieved correctly is at most μ times the probability that (26) is satisfied.

2.8.3 Failure probability computation: ring parameters with $\xi(x)=\Phi_{n+1}(x)$

Round5 parameters without error correction use $\xi(x)=\Phi_{n+1}(x)$. As $N(x)$ is a multiple of $\Phi_{n+1}(x)$, we have that $\langle f\rangle_{\Phi_{n+1}}=\langle\langle f\rangle_N\rangle_{\Phi_{n+1}}$. Moreover, if $g(x)=\sum_{i=0}^n g_ix^i$, then $\langle g\rangle_{\Phi_{n+1}}=g-g_n\Phi_{n+1}$. In particular, for all polynomials s,e,

$$\text{if } \langle se\rangle_N=\sum_{k=0}^{n}c_k(s,e)x^k, \text{ then } \langle se\rangle_{\Phi_{n+1}}=\sum_{k=0}^{n-1}(c_k(s,e)-c_n(s,e))x^k, \tag{30}$$

with $c_k(s,e)$ as defined in (33). If the k-th symbol is not retrieved correctly, then

$$\langle (j_v(x))_k+c_k(j_B,r)-c_n(j_B,r)+c_k(s,j_u)+c_n(s,j_u)\rangle_q\in\left[\frac{q}{2b},q-\frac{q}{2b}\right] \tag{31}$$

Assuming independence, and taking into account that r and s contain $h/2$ ones and $h/2$ minus ones, $c_k(j_B,r)-c_n(j_B,r)-c_n(j_B,r)+c_k(s,j_u)$ is distributed as the difference of $2h$ independent random variables on I, minus the sum of $2h$ independent random variables on I. The probability that (31) is satisfied thus can be computed explicitly. By the union bound, the probability that at least one of the μ symbols is not retrieved correctly is at most μ times the probability that (31) is satisfied.

Remark The condition in (31) for decryption error in position k shows the term $-c_n(j_B,r)+c_n(s,j_u)$ that is common to all positions k. This is the reason that using error correction in conjunction with $\xi(x)=\Phi_{n+1}(x)$ is not effective. The union bound can be used as it also applies to dependent random variables.

2.8.4 Failure probability computation: ring parameters with $\xi(x) = x^{n+1} - 1$

Round5 parameters using error correction are restricted to the ring case. So in (27) we have

$$\langle \boldsymbol{S}^T \boldsymbol{A}^T \rangle_{\Phi_{n+1}} \boldsymbol{R} - \boldsymbol{S}^T \langle \boldsymbol{A}^T \boldsymbol{R} \rangle_{\Phi_{n+1}} = \lambda_s(x)\Phi_{n+1}(x)r(x) - s(x)\lambda_r(x)\Phi_{n+1}(x)$$

for some $\lambda_s, \lambda_r \in \mathbb{Z}[x]$.

For the Round5 parameters with error correction, it is required that $\xi(x) = (x-1)\Phi_{n+1}(x) = x^{n+1} - 1$, and that $s(x)$ and $r(x)$ both are balanced, that is, have $h/2$ coefficients equal to 1, $h/2$ coefficients equal to -1, and the other coefficients equal to zero. Then $x - 1$ divides both $r(x)$ and $s(x)$, and so $\xi(x)$ divides both $\Phi_{n+1}(x)r(x)$ and $s(x)\Phi_{n+1}(x)$. As a result, (29) reads as

$$\frac{q}{p}\Delta(x) \equiv j_v(x) + \text{Sample}_\mu \left(\langle \boldsymbol{j}_B(x)r(x) - s(x)j_U(x) \rangle_{x^{n+1}-1} \right) \pmod q. \quad (32)$$

For any two polynomials $e(x) = \sum_{i=0}^{n} e_i x^i$ and $s(x) = \sum_{i=0}^{n} s_i x^i$,

$$\langle s(x)e(x) \rangle_{x^{n+1}-1} = \sum_{k=0}^{n} c_k(s,e)x^k \text{ where } c_k(s,e) = \sum_{i=0}^{n} e_i s_{\langle k-i \rangle_{n+1}}. \quad (33)$$

If the k-th symbol is not retrieved correctly, then

$$\langle (j_v(x))_k + c_k(j_B, r) - c_k(s, j_u) \rangle_q \in \left[\frac{q}{2b}, q - \frac{q}{2b} \right]. \quad (34)$$

Assuming independence, using that r and s both contain $h/2$ and $h/2$ minus ones, $c_k(j_B, r) - c_k(s, j_u)$ is distributed as the sum of h independent uniform variables on I, minus the sum of h uniform variables on I. The probability p_n that (33) is satisfied thus can be computed explicitly.

Now let the error-correcting code be capable of correcting f symbol errors. Assuming that $c_k(s,e)$ and $c_\ell(s,e)$ are independent whenever $k \neq \ell$, the probability of not decoding correctly is at most $\sum_{e \geq f+1} \binom{\mu}{e} p_n^e (1-p_n)^{\mu-e}$.

2.8.5 Experimental results

Figure 3 compares the estimated probabilities of at least one error occurring and that of at least two errors occurring, when $\xi = N_{n+1}$ (as in r5_cpa_pke) and when $\xi = \Phi_{n+1}$, respectively. These estimates are computed by explicitly convolving probability distributions. Parameters are simulated without error correction, and are $n = 800$, $q = 2^{11}$, $p = 2^7$, $t = 2^4$, $\mu = \kappa = 128$, while the Hamming weight h varies between 100 and 750 in order to show its effect on both the bit failure rate and error correlation.

For $\xi = \Phi_{n+1}$, the probabilities are computed as follows. For any a, (31) can be used to compute $p(k \mid a)$, the probability that bit k is not retrieved correctly, given that $-c_n(j_b, r) + c_n(s, j_u) \equiv a \pmod q$. We assume that having

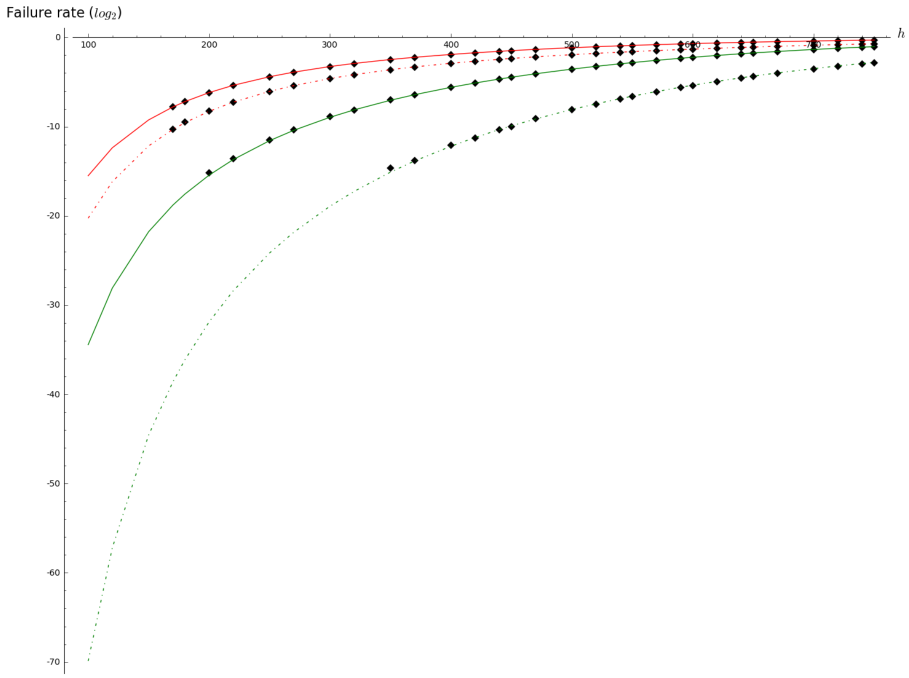

Figure 3: Probabilities of at least one (continuous lines) and at least two errors (dotted lines) in Round5 ring parameters, plotted against the Hamming weight of secrets (X-axis). Red and green curves respectively represent predicted values for the above in parameters using $\Phi_{n+1}(x)$ and $N_{n+1}(x)$ as reduction polynomial, respectively. Diamonds represent corresponding probabilities computed from actual Round5 simulations for the same parameters, with 10^6 runs per datapoint. Scripts for analyzing and reproducing these results can be found at `www.round5.org`.

a bit error in position k, given that $c_n(s, j_u) - c_n(j_b, r) \equiv a$, is independent of having a bit error in another position j, given that $c_n(s, j_u) - c_n(j_{b}, _r\ ) \equiv a$. The probability of having exactly e bit errors, given that $c_n(s, j_u) - c_n(j_b, r) \equiv a$, then equals $\binom{\mu}{k}(p(0 \mid a)^e(1 - p(0 \mid a))^{\mu - e}$. By summing these probabilities over a, weighted with the probability that $c_n(s, j_u) - c_n(j_b, r) \equiv a$, the probability of having exactly e bit errors is obtained.

Clearly, the probability of at least two errors is much higher when multiplications are done modulo Φ_{n+1} instead of N_{n+1}, and in the latter case, this probability is significantly lower than the probability of at least one error. Figure 3 also shows corresponding probabilities of at least one and at least two errors, obtained from simulations of scaled-down r5_cpa_pke parameters, showing that the actual behavior closely matches that predicted by the failure analysis.

As further evidence, Table 2 reports experimental results for actual, proposed parameters for the ring instantiation of Round5's IND-CPA secure key-

Table 2: Experimental and predicted bit failure probabilities $\hat{p}_{b,\text{expt}}$, $\hat{p}_{b,\text{pred}}$, for proposed parameters of Round5's error correction using, ring-based instantiation (Section 2.9). Also shown are overall failure rates $p_{>f,\text{expt}}$, $p_{>f,\text{pred}}$ that are computed using $p_{b,\text{expt}}$ and predicted by the failure analysis, respectively.

Parameters	$\boldsymbol{S}$	$\boldsymbol{n_1}$	$\boldsymbol{n_2}$	$\boldsymbol{n_3}$	$\hat{\boldsymbol{p}}_{\boldsymbol{b},\text{expt}}$	$\hat{\boldsymbol{p}}_{\boldsymbol{b},\text{pred}}$	$\frac{\boldsymbol{n_2}}{\boldsymbol{S}}_{\text{expt}}$	$\frac{\boldsymbol{n_2}}{\boldsymbol{S}}_{\text{pred}}$	$\boldsymbol{p}_{>\boldsymbol{f},\text{expt}}$	$\boldsymbol{p}_{>\boldsymbol{f},\text{pred}}$
R5ND_1KEM_5d	8.5×10^9	226639	6	0	$2^{-22.19}$	$2^{-21.35}$	$2^{-30.4}$	$2^{-31.4}$	$2^{-92.85}$	$2^{-87.79}$
R5ND_3KEM_5d	2.2×10^9	4120	0	0	$2^{-26.61}$	$2^{-26.61}$	N/A	$2^{-39.06}$	$2^{-117.14}$	$2^{-117.13}$
R5ND_5KEM_5d	2.8×10^9	2685625	1314	1	$2^{-18.02}$	$2^{-17.99}$	$2^{-21.02}$	$2^{-21.06}$	$2^{-64.07}$	$2^{-63.85}$

encapsulation mechanism that use error correction, i.e., R5ND_1KEM_5d, R5ND_3KEM_5d and R5ND_5KEM_5d (presented in Section 2.9). These results are compared with those predicted by the failure probability analysis from Section 2.8.4. For each of the parameter sets, n_i, the number of simulated message exchanges for which the XEf decoder flipped exactly i bits was measured. The total number of flipped bits in the simulations thus equals $\sum_i in_i$. As the XEf decoder can only flip the κ message bits, and does not change the parity bits, only errors in κ message bits are taken into account, and the experimental bit failure rate $\hat{p}_{b,\text{expt}}$ equals

$$\hat{p}_{b,\text{expt}} = \frac{\sum_i in_i}{\kappa S}, \tag{35}$$

where S equals the number of simulated message exchanges. Table 2 shows that $\hat{p}_{b,\text{expt}}$ closely fits the value of $\hat{p}_{b,\text{pred}}$ for the corresponding parameter set predicted by the failure analysis from Section 2.8.4.

In case of independent bit failures with probability $\hat{p}_b$, the probability of having exactly two bit failures in κ information bits equals

$$\binom{\kappa}{2}\hat{p}_b^2(1-\hat{p}_b)^{\kappa-2} \tag{36}$$

The fraction of simulated exchanges with exactly two bit errors equals n_2/S. Table 2 tabulates the values $(n_2/S)_{\text{expt}}$ from simulations, and shows that they fit the expected $(n_2/S)_{\text{pred}} = \binom{\kappa}{2}\hat{p}_{b,\text{expt}}^2(1-\hat{p}_{b,\text{expt}})^{\kappa-2}$ for R5ND_5KEM_5d, for which n_2 is sufficiently large to draw conclusions. This supports the assumption of independent bit failures. Finally, the probability of having more than f errors (overwhelming the f-bit error correction code) is computed using $\hat{p}_{b,\text{expt}}$ and compared with the failure rate predicted by the correctness analysis for each parameter set.

To conclude, the effect of dependence between polynomial coefficients due to polynomial multiplication modulo Φ_{n+1}, is made negligible by the combined use of polynomial multiplication modulo N_{n+1} and balanced secrets, allowing the use of forward error correction, resulting in better security and performance.

2.9 Round5: Parameter Sets and Performance

Round5 offers a large design space allowing for a unified and efficient implementation of Round5, independently of the fact that it instantiates LWR or RLWR. r5_cpa_kem and r5_cca_pke can be configured to instantiate the LWR and RLWR underlying problems depending on the input parameter n. As described in detail later, the security level depends on the input value d and also the choice of q and p. All moduli, q, p, and t are chosen to be powers-of-two. Usage of error correction to better deal with decryption failures can also be configured by setting up parameter f. If $f > 0$, then secrets must be balanced and the reduction polynomial $\xi(x)$ for the ciphertext component v must be $x^{n+1} - 1$. A final restriction is that $\Phi_{n+1}(x)$ is irreducible modulo two.

The Round5 parameter sets are described in Sections 2.9.1 and Section 2.9.2. Section 2.9.3 describes the implementations in the submission package. Section 2.9.5 details the parameter sets for r5_cpa_kem. Section 2.9.6 summarizes the performance of r5_cpa_kem. Section 2.9.7 details the parameter sets of r5_cca_pke. Section 2.9.8 summarizes the performance of r5_cca_pke. Section 2.9.9 reports performance on other platforms.

2.9.1 Main parameter sets

The Round5 cryptosystem has 18 main parameters sets: six ring parameter sets without error correction, six ring parameter sets with error correction, and six non-ring parameter sets. Each set of six parameter sets is for r5_cpa_pke and r5_cca_pke and NIST levels 1,3, and 5. A parameter set is denoted as R5N{1,D}_{1,3,5}{KEM,PKE}_{0,5}{version} where:

- 1,D refers whether it is a non-ring (1) or ring (D) parameter set.
- 1,3,5 refers to the NIST security level that is strictly fulfilled.
- KEM,PKE refers to the cryptographic algorithm it instantiates.
- 0,1,2,3,4,5 identifies the amount of corrected bits, 0 means no-errors are corrected and this description is equivalent to the original Round2; 5 means that up to 5 errors can be corrected as in the original HILA5 submission.
- version is a letter to indicate the version of published parameters to account for potential differences in published parameters.

R5ND_{1,3,5}KEM_5d: Merged parameters for the ring variant ($n = d$) of Round5 key-encapsulation mechanism, for NIST security levels 1, 3 and 5, resulting from the merge of the NIST PQC first round candidate cryptosystems Round2 and HILA5. XE5 forward error correction is used to decrease decryption failure rates (hence the 5 at the end of the parameter designator), and improve bandwidth and security.

R5ND_{1,3,5}PKE_5d: Merged parameters for the ring variant ($n = d$) of Round5 public-key encryption, for NIST security levels 1, 3 and 5, resulting from the merge of the NIST PQC first round candidate cryptosystems Round2 and HILA5. XE5 forward error correction is used to decrease decryption failure rates (hence the 5 at the end of the parameter designator), and improve bandwidth and security.

R5ND_{1,3,5}KEM_0d: Parameters for the ring variant ($n = d$) of Round5 INDCPA key-encapsulation mechanism, for NIST security levels 1, 3 and 5. No forward error correction is used, hence the 0 at the end of the parameter designator. All polynomial multiplications are done modulo the prime-order cyclotomic polynomial. This choice is considered more conservative since it uses the same design principles as Round2.

R5ND_{1,3,5}PKE_0d: Parameters for the ring variant ($n = d$) of Round5 INDCCA public-key encryption, for NIST security levels 1, 3 and 5. No forward error correction is used, hence the 0 at the end of the parameter designator. All polynomial multiplications are done modulo the prime-order cyclotomic polynomial. This choice is considered more conservative since it uses the same design principles as Round2.

R5N1_{1,3,5}KEM_0d: Parameters for the non-ring/unstructured variant ($n = 1$) of Round5 key-encapsulation mechanism, for NIST security levels 1, 3 and 5. No forward error correction is used, hence the 0 at the end of the parameter designator. Since the generation of the public-parameter $\boldsymbol{A}$ is expensive, we include performance results for three alternative ways of generating it denoted as $T0, T1, T2$ and discuss performance trade-offs between the usage of SHAKE256 and AES128.

R5N1_{1,3,5}PKE_0d: Parameters for the non-ring/unstructured variant ($n = 1$) of Round5 public-key encrpytion, for NIST security levels 1, 3 and 5. No forward error correction is used, hence the 0 at the end of the parameter designator.

2.9.2 Parameter sets for specific use-cases

Round5 has three additional "specific use-case" parameter sets with the only purpose of demonstrating the flexibility of Round5 and its applicability to a number of diverse, specialized usage scenarios.

R5ND_0KEM_2iot: A small parameter set targeting the Internet of Things use-cases, with lower bandwidth and computational footprints. We set as security targets the encapsulation of a 128-bit key and a security strength of at least 80/96-bits of quantum/classical security (core sieving) with a failure probability at least as low as 2^{-40}. We believe that these settings are suitable for low security IoT applications and the lower bandwidth needs can facilitate adoption. It turns out that these targets fit a classical security level of 128-bit XE2 forward error correction is used to improve failure rate, bandwidth and security.

R5ND_1KEM_4longkey: An alternative to the NIST level 1, this ring-variant key-encapsulation parameter set encapsulates a 192-bit key despite targeting the NIST security level 1. This ensures that the (quantum) cost of atacking the encapsulated key (e.g., by using Grover's quantum search algorithm) is as much as the (quantum) cost of attacking the underlying cryptosystem, i.e., Round5.

R5N1_3PKE_0smallCT: An alternative to the NIST level 3, this non-ring variant public-key encryption parameter set has exceptionally small ciphertexts. This parameter set targets use cases for which the public-key is static and hence is exchanged rarely, implying that bandwidth footprint depends on the size of the ciphertext. Hence this parameter set, despite enjoying the more conservative security assumption based on unstructured lattices, has a bandwidth requirement comparable to ring variants.

2.9.3 Implementations

We include the following implementations of Round5.

- Reference: This is a reference implementation of Round5 capable of running all parameter sets after compilation. Matrix and polynomial operations are explicit so that it is simple to verify the correctness of the implementation. This implementation is detailed in Section 2.11.
- Optimized: This is an optimized implementation of Round5 capable of running a single parameter set, fixed at compile time. This implementation can also fix – at compile time – the way to generate $\boldsymbol{A}$, i.e., $f^{(0)}_{d,n}$, $f^{(1)}_{d,n}$, or $f^{(2)}_{d,n}$ and whether to use SHAKE or AES. Trade-offs are discussed in the following sections. This implementation can also select multiple ways of computing the secrets, including index-based, cache memory side-channel resistant, and AVX2 optimized for unstructured parameter sets, i.e., R5N1*.
- Additional implementations: We include and/or report on results of additional implementations.
  - Configurable: This is an optimized implementation of Round5 that can be configured and can execute all parameter sets at run time. This implementation shows the feasibility of an implementation running very different parameter sets with a common code base that can give flexibility, e.g., if the user needs to choose parameter sets with different security levels or an structured/unstructured problem, *post compilation*. The reason why this is feasible is the specific choices in Round5 that allow performing the core operations in very similar code. The most important feature is that for both ring and non-ring configurations, the core operations can be represented as a matrix multiplication of dimension $d \times d$ times a vector of length d.
  - AVX2: This implementation optimizes the core matrix multiplication operations in the optimized implementation for the Round5 non-ring parameters, namely $\boldsymbol{B} = \boldsymbol{AS}$, $\boldsymbol{U} = \boldsymbol{R}^T\boldsymbol{A}$, and $\boldsymbol{S}^T\boldsymbol{U}$.

2.9.4 Development Environment

The performance numbers for the optimized implementation of Round5 have been gathered on a MacBookPro15.1 with an Intel Core i7 2.6GHz, running macOS 10.13.6. The code has been compiled with `gcc -march=native -mtune=native -O3 -fomit-frame-pointer -fwrapv`, using Apple LLVM version 10.0.0 (clang-1000.10.44.4). Cache attack countermeasures (`-DCM_CACHE`) were enabled.

All tests were run 1000 times, the measurements shown are the median values of all test runs and are for the optimized implementation of the algorithm.

For the memory requirements, we have provided an indication based on a possible implementation. Note that the actual memory usage depends, of course, on the specifics of a particular implementation (e.g., an implementation might require matrices to exist (also) in transposed form, need room for storing intermediate results, etc.).

2.9.5 r5_cpa_kem: Parameters

This section contains specific parameter values for r5_cpa_kem fitting the Round5 parameter sets described at the beginning of this section.

Tables 3, 4, and 5 show the parameter sets for NIST security levels 1, 3, and 5. Table 6 present the parameter sets for specific use cases.

In all of the above tables, the values under *Security Levels* represent the cost of known attacks (discussed in Section 2.7) against the underlying LWR or RLWR problem.

The main conclusions from these tables are as follows:

- All parameter sets strictly fulfill NIST security levels and the hybrid attack is the one that has the highest impact on Round5.
- Parameter sets using XE5 (R5ND_1,3,5KEM_5d) have around 25% lower bandwidth requirements compared with those without error correction (R5ND_1,3,5KEM_0d). Since Round5 proposes a KEM that offers CPA security, no active attacks are applicable.
- The application-specific IoT parameter set (R5ND_0KEM_2iot) only requires 736 bytes and it still offers a classical/quantum security level of 96/88 bits.
- Many security protocols that require a handshake to exchange a session key do not require active security and a relatively high failure probability is good enough. This allows finding parameters that offer smaller keys and ciphertext at the price of a slightly higher failure rate. For instance, for IoT deployment the failure rate of a wireless communication link will be likely worse than the failure rate of R5ND_0KEM_2iot, and thus, a failure probability of 2^{-41} is sufficient. Alternatively, it is possible to further increase this failure probability so that the sizes of public-key and ciphertext further drop.

Table 3: R5ND_{1,3,5}KEM_0d parameters

	Security Level		
	NIST1	**NIST3**	**NIST5**
Parameters			
$d/n/h$	618/618/104	786/786/384	1018/1018/428
$q/p/t/b$	$2^{11}/2^{8}/2^{4}/2^{1}$	$2^{13}/2^{9}/2^{4}/2^{1}$	$2^{14}/2^{9}/2^{4}/2^{1}$
$\overline{n}/\overline{m}$	1/1	1/1	1/1
$f/xe/\kappa/\mu$	0/0/128/128	0/0/192/192	0/0/256/256
Performance			
Total bandwidth (bytes)	1316	1890	2452
Public key (bytes)	634	909	1178
Ciphertext (bytes)	682	981	1274
Failure rate	2^{-65}	2^{-71}	2^{-64}
Classical/Quantum security (bits) – Core Sieving			
Primal attack	144/131	192/175	256/233
Dual attack	147/134	193/175	258/235
Hybrid attack	128/122	194/183	261/248
Sparse-secrets attack	144/130	192/174	256/232
Classical/Quantum optimal block size BKZ – Core Sieving			
Primal attack	494/494	659/659	878/878
Dual attack	504/504	662/662	885/885
Hybrid attack	437/461	665/691	895/934
Classical/Quantum security (bits) – Enumeration			
Primal attack	340/170	500/250	729/365
Dual attack	350/175	503/252	737/368
Hybrid attack	160/133	293/222	397/309
Sparse-secrets attack	340/170	500/250	729/364
Classical/Quantum optimal block size BKZ – Enumeration			
Primal attack	494/494	659/659	878/878
Dual attack	504/504	661/662	885/885
Hybrid attack	284/412	442/603	554/773

Table 4: R5ND_{1,3,5}KEM_5d parameters

	Security Level		
	NIST1	**NIST3**	**NIST5**
Parameters			
$d/n/h$	490/490/162	756/756/242	940/940/414
$q/p/t/b$	$2^{10}/2^{7}/2^{3}/2^{1}$	$2^{12}/2^{8}/2^{2}/2^{1}$	$2^{12}/2^{8}/2^{2}/2^{1}$
$\overline{n}/\overline{m}$	1/1	1/1	1/1
$f/xe/\kappa/\mu$	5/190/128/318	5/218/192/410	5/234/256/490
Performance			
Total bandwidth (bytes)	994	1639	2035
Public key (bytes)	445	780	972
Ciphertext (bytes)	549	859	1063
Failure rate	2^{-88}	2^{-117}	2^{-64}
Classical/Quantum security (bits) – Core Sieving			
Primal attack	130/118	194/176	256/233
Dual attack	132/120	197/179	259/235
Hybrid attack	128/122	193/183	263/249
Sparse-secrets attack	130/118	194/176	256/232
Classical/Quantum optimal block size BKZ – Core Sieving			
Primal attack	446/446	666/666	878/878
Dual attack	452/452	674/674	887/887
Hybrid attack	440/462	660/691	900/940
Classical/Quantum security (bits) – Enumeration			
Primal attack	297/148	507/254	729/365
Dual attack	301/151	515/258	739/369
Hybrid attack	170/135	270/215	390/307
Sparse-secrets attack	296/148	507/253	729/364
Classical/Quantum optimal block size BKZ – Enumeration			
Primal attack	446/446	666/666	878/878
Dual attack	451/451	673/674	887/887
Hybrid attack	297/416	416/588	547/770

Table 5: R5N1_{1,3,5}KEM_0d parameters

	Security Level		
	NIST1	**NIST3**	**NIST5**
Parameters			
$d/n/h$	594/1/238	881/1/238	1186/1/712
$q/p/t/b$	$2^{13}/2^{10}/2^{7}/2^{3}$	$2^{13}/2^{10}/2^{7}/2^{3}$	$2^{15}/2^{12}/2^{7}/2^{4}$
$\overline{n}/\overline{m}$	7/7	8/8	8/8
$f/xe/\kappa/\mu$	0/0/128/43	0/0/192/64	0/0/256/64
Performance			
Total bandwidth (bytes)	10450	17700	28552
Public key (bytes)	5214	8834	14264
Ciphertext (bytes)	5236	8866	14288
Failure rate	2^{-66}	2^{-65}	2^{-77}
Classical/Quantum security (bits) – Core Sieving			
Primal attack	132/121	201/184	257/234
Dual attack	131/120	202/184	256/233
Hybrid attack	128/121	192/182	256/241
Sparse-secrets attack	130/119	201/183	256/233
Classical/Quantum optimal block size BKZ – Core Sieving			
Primal attack	422/422	658/658	847/847
Dual attack	418/418	659/659	843/844
Hybrid attack	408/423	626/652	844/871
Classical/Quantum security (bits) – Enumeration			
Primal attack	275/138	499/250	696/348
Dual attack	272/136	500/250	692/346
Hybrid attack	177/130	268/210	420/304
Sparse-secrets attack	271/135	499/249	691/345
Classical/Quantum optimal block size BKZ – Enumeration			
Primal attack	422/422	658/658	847/847
Dual attack	418/418	659/659	843/843
Hybrid attack	306/404	414/577	578/765

Table 6: Specific use case Round5 KEM parameters

	Parameter Set	
	R5ND_0KEM_2iot	**R5ND_1KEM_4longkey**
Parameters		
$d/n/h$	372/372/178	490/490/162
$q/p/t/b$	$2^{11}/2^7/2^3/2^1$	$2^{10}/2^7/2^3/2^1$
$\overline{n}/\overline{m}$	1/1	1/1
$f/xe/\kappa/\mu$	2/53/128/181	4/163/192/355
Performance		
Total bandwidth (bytes)	736	1016
Public key (bytes)	342	453
Ciphertext (bytes)	394	563
Failure rate	2^{-41}	2^{-71}
Classical/Quantum security (bits) – Core Sieving		
Primal attack	98/89	130/118
Dual attack	98/89	132/120
Hybrid attack	96/90	128/122
Sparse-secrets attack	97/88	130/118
Classical/Quantum optimal block size BKZ – Core Sieving		
Primal attack	335/335	446/446
Dual attack	336/336	452/452
Hybrid attack	329/341	440/462
Classical/Quantum security (bits) – Enumeration		
Primal attack	201/100	297/148
Dual attack	202/101	301/151
Hybrid attack	129/96	170/135
Sparse-secrets attack	200/100	296/148
Classical/Quantum optimal block size BKZ – Enumeration		
Primal attack	335/335	446/446
Dual attack	336/336	451/451
Hybrid attack	243/325	297/416

2.9.6 r5_cpa_kem: CPU and Memory Requirements

This section contains the CPU and memory requirements for r5_cpa_kem described in Section 2.9.5.

Table 7 shows the performance and memory usage figures for the optimized implementation of the R5ND_{1,3,5}KEM_0d algorithm. Table 8 shows the performance and memory usage figures for the optimized implementation of the R5ND_{1,3,5}KEM_5d algorithm. Table 9 shows the performance and memory usage figures for the optimized implementation of the R5N1_{1,3,5}KEM_0d $\tau = 2$ algorithm. Table 10 shows the performance of the parameter sets for specific use cases of the optimized implementation of r5_cpa_kem.

The main conclusions from these tables are as follows:

- CPU performance for ring parameter sets does not seem to be an issue for most applications since it requires fractions of a millisecond without using any type of hardware instructions.
- CPU performance for non-ring parameter sets can be sufficient for many applications, except those that require very high throughput.

Table 7: Optimized R5ND_{1,3,5}KEM_0d performance and memory usage

	Security Level		
	NIST1	**NIST3**	**NIST5**
API Parameters			
`CRYPTO_SECRETKEYBYTES`	16	24	32
`CRYPTO_PUBLICKEYBYTES`	634	909	1,178
`CRYPTO_BYTES`	16	24	32
`CRYPTO_CIPHERTEXTBYTES`	682	981	1,274
Performance: Elapsed time (ms)			
KEM Generate Key Pair	0.02	0.06	0.06
KEM Encapsulate	0.04	0.08	0.10
KEM Decapsulate	0.02	0.04	0.05
Total	0.08	0.2	0.2
Performance: CPU Clock Cycles			
KEM Generate Key Pair	57.6k	140.1k	159.9k
KEM Encapsulate	94.9k	212.3k	248.0k
KEM Decapsulate	45.0k	114.5k	132.5k
Total	197.5k	466.8k	540k
Memory usage indication			
KEM Generate Key Pair	4,374B	5,673B	7,350B
KEM Encapsulate	7,784B	10,182B	12.9KiB
KEM Decapsulate	3,458B	4,581B	5,954B

Table 8: Optimized R5ND_{1,3,5}KEM_5d performance and memory usage

	Security Level		
	NIST1	NIST3	NIST5
API Parameters			
`CRYPTO_SECRETKEYBYTES`	16	24	32
`CRYPTO_PUBLICKEYBYTES`	445	780	972
`CRYPTO_BYTES`	16	24	32
`CRYPTO_CIPHERTEXTBYTES`	549	859	1,063
Performance: Elapsed time (ms)			
KEM Generate Key Pair	0.02	0.06	0.06
KEM Encapsulate	0.04	0.09	0.10
KEM Decapsulate	0.02	0.04	0.06
Total	0.08	0.2	0.2
Performance: CPU Clock Cycles			
KEM Generate Key Pair	54.0k	144.7k	147.8k
KEM Encapsulate	94.4k	235.0k	246.6k
KEM Decapsulate	55.8k	111.3k	146.7k
Total	204.2k	491.0k	541k
Memory usage indication			
KEM Generate Key Pair	3,417B	5,364B	6,676B
KEM Encapsulate	6,182B	9,631B	11.7KiB
KEM Decapsulate	2,813B	4,339B	5,431B

Table 9: Optimized R5N1_{1,3,5}KEM_0d $\tau = 2$ performance and memory usage

	Security Level		
	NIST1	**NIST3**	**NIST5**
API Parameters			
CRYPTO_SECRETKEYBYTES	16	24	32
CRYPTO_PUBLICKEYBYTES	5,214	8,834	14,264
CRYPTO_BYTES	16	24	32
CRYPTO_CIPHERTEXTBYTES	5,236	8,866	14,288
Performance: Elapsed time (ms)			
KEM Generate Key Pair	1.1	2.6	5.4
KEM Encapsulate	1.6	3.9	7.2
KEM Decapsulate	0.07	0.1	0.3
Total	2.7	6.6	12.9
Performance: CPU Clock Cycles			
KEM Generate Key Pair	2,766k	6,694k	14.0M
KEM Encapsulate	4,049k	10.1M	18.6M
KEM Decapsulate	188.8k	275.1k	814k
Total	7,003k	17.1M	33.4M
Memory usage indication			
KEM Generate Key Pair	26.5KiB	41.9KiB	57.4KiB
KEM Encapsulate	39.9KiB	64.6KiB	90.1KiB
KEM Decapsulate	21.5KiB	36.4KiB	51.2KiB

Table 10: Optimized r5_cpa_kem specific use case parameters, performance and memory usage

	Parameter Set	
	R5ND_0KEM_2iot	**R5ND_1KEM_4longkey**
API Parameters		
`CRYPTO_SECRETKEYBYTES`	16	24
`CRYPTO_PUBLICKEYBYTES`	342	453
`CRYPTO_BYTES`	16	24
`CRYPTO_CIPHERTEXTBYTES`	394	563
Performance: Elapsed time (ms)		
KEM Generate Key Pair	0.02	0.02
KEM Encapsulate	0.04	0.04
KEM Decapsulate	0.02	0.02
Total	0.08	0.08
Performance: CPU Clock Cycles		
KEM Generate Key Pair	56.3k	57.2k
KEM Encapsulate	97.9k	97.7k
KEM Decapsulate	59.5k	55.5k
Total	213.6k	210.3k
Memory usage indication		
KEM Generate Key Pair	2,606B	3,441B
KEM Encapsulate	4,744B	6,348B
KEM Decapsulate	2,186B	2,979B

2.9.7 r5_cca_pke: Parameters

This section contains specific parameter sets for r5_cca_pke.

Tables 11, 12, and 13 show the parameter sets for NIST security levels 1, 3, and 5. Table 14 present the parameter sets for specific use cases.

In all of the above tables, the values under *Security Levels* represent the cost of known attacks (discussed in Section 2.7) against the underlying LWR or RLWR problem.

The main conclusions from these tables are as follows:

- As for r5_cpa_kem parameters, all parameter sets strictly fulfill NIST security levels and the hybrid attack is the one that has the highest impact on Round5.
- Parameter sets using XE5 (R5ND_1,3,5PKE_5d) have around 25% lower bandwidth requirements compared with those parameters without error correction (R5ND_1,3,5KEM_0d). For instance, R5ND_5PKE_5d requires 335 Bytes less than R5ND_5PKE_0d.
- r5_cca_pke, targeting IND-CCA security, requires lower failure probability compared with r5_cpa_kem. This leads to parameters that require slightly longer public-keys and ciphertexts. When no error correction is used, this difference is larger and can be up to 422 Bytes.
- The application-specific R5N1_3PKE_0smallCT, i.e., a non-ring parameter set with short ciphertext, is a desirable alternative in use cases in which the public-key remains static for a long period of time, since the ciphertext is small and comparable in size with ring parameter sets, minimizing the overall bandwidth requirement.

Table 11: R5ND_{1,3,5}PKE_0d parameters

	Security Level		
	NIST1	**NIST3**	**NIST5**
Parameters			
$d/n/h$	586/586/182	852/852/212	1170/1170/222
$q/p/t/b$	$2^{13}/2^{9}/2^{4}/2^{1}$	$2^{12}/2^{9}/2^{5}/2^{1}$	$2^{13}/2^{9}/2^{5}/2^{1}$
$\overline{n}/\overline{m}$	1/1	1/1	1/1
$f/xe/\kappa/\mu$	0/0/128/128	0/0/192/192	0/0/256/256
Performance			
Total bandwidth (bytes)	1432	2102	2874
Public key (bytes)	676	983	1349
Encryption overhead (bytes)	756	1119	1525
Failure rate	2^{-155}	2^{-147}	2^{-143}
Classical/Quantum security (bits) – Core Sieving			
Primal attack	131/119	199/181	281/255
Dual attack	132/120	202/183	286/260
Hybrid attack	128/121	192/182	257/246
Sparse-secrets attack	130/118	199/180	281/255
Classical/Quantum optimal block size BKZ – Core Sieving			
Primal attack	448/448	683/683	963/963
Dual attack	451/451	692/692	981/981
Hybrid attack	439/457	657/688	880/929
Classical/Quantum security (bits) – Enumeration			
Primal attack	298/149	524/262	822/411
Dual attack	301/151	534/267	842/421
Hybrid attack	177/135	266/213	346/289
Sparse-secrets attack	298/149	524/262	822/411
Classical/Quantum optimal block size BKZ – Enumeration			
Primal attack	448/448	683/683	963/963
Dual attack	451/451	692/692	981/981
Hybrid attack	306/417	412/584	500/736

Table 12: R5ND_{1,3,5}PKE_5d parameters

	Security Level		
	NIST1	**NIST3**	**NIST5**
Parameters			
$d/n/h$	508/508/136	756/756/242	946/946/388
$q/p/t/b$	$2^{10}/2^{7}/2^{4}/2^{1}$	$2^{12}/2^{8}/2^{3}/2^{1}$	$2^{11}/2^{8}/2^{5}/2^{1}$
$\overline{n}/\overline{m}$	1/1	1/1	1/1
$f/xe/\kappa/\mu$	5/190/128/318	5/218/192/410	5/234/256/490
Performance			
Total bandwidth (bytes)	1097	1730	2279
Public key (bytes)	461	780	978
Encryption overhead (bytes)	636	950	1301
Failure rate	2^{-142}	2^{-256}	2^{-227}
Classical/Quantum security (bits) – Core Sieving			
Primal attack	133/121	194/176	256/233
Dual attack	135/122	197/179	259/235
Hybrid attack	128/122	193/183	263/248
Sparse-secrets attack	132/120	194/176	256/232
Classical/Quantum optimal block size BKZ – Core Sieving			
Primal attack	455/455	666/666	878/878
Dual attack	462/462	674/674	888/888
Hybrid attack	437/462	660/691	899/937
Classical/Quantum security (bits) – Enumeration			
Primal attack	305/152	507/254	729/365
Dual attack	311/156	515/258	740/370
Hybrid attack	166/134	270/215	386/306
Sparse-secrets attack	304/152	507/253	729/364
Classical/Quantum optimal block size BKZ – Enumeration			
Primal attack	455/455	666/666	878/878
Dual attack	462/462	673/674	888/888
Hybrid attack	292/413	416/588	542/768

Table 13: R5N1_{1,3,5}PKE_0d parameters

	Security Level		
	NIST1	**NIST3**	**NIST5**
Parameters			
$d/n/h$	636/1/114	876/1/446	1217/1/462
$q/p/t/b$	$2^{12}/2^{9}/2^{6}/2^{2}$	$2^{15}/2^{11}/2^{7}/2^{3}$	$2^{15}/2^{12}/2^{9}/2^{4}$
$\overline{n}/\overline{m}$	8/8	8/8	8/8
$f/xe/\kappa/\mu$	0/0/128/64	0/0/192/64	0/0/256/64
Performance			
Total bandwidth (bytes)	11544	19392	29360
Public key (bytes)	5740	9660	14636
Encryption overhead (bytes)	5804	9732	14724
Failure rate	2^{-146}	2^{-144}	2^{-144}
Classical/Quantum security (bits) – Core Sieving			
Primal attack	146/133	194/177	257/234
Dual attack	146/134	193/176	257/234
Hybrid attack	128/122	192/181	256/241
Sparse-secrets attack	145/133	192/175	257/234
Classical/Quantum optimal block size BKZ – Core Sieving			
Primal attack	469/469	632/632	848/848
Dual attack	471/471	628/628	847/847
Hybrid attack	407/428	626/646	844/874
Classical/Quantum security (bits) – Enumeration			
Primal attack	317/159	473/237	697/348
Dual attack	319/160	469/235	696/348
Hybrid attack	159/130	296/214	402/300
Sparse-secrets attack	317/158	469/234	695/347
Classical/Quantum optimal block size BKZ – Enumeration			
Primal attack	469/469	632/632	848/848
Dual attack	470/471	628/628	847/847
Hybrid attack	283/404	445/586	559/756

Table 14: Specific use case Round5 PKE parameters

	Parameter Set **R5N1_3PKE_0smallCT**
Parameters	
$d/n/h$	757/1/378
$q/p/t/b$	$2^{14}/2^{9}/2^{4}/2^{1}$
$\overline{n}/\overline{m}$	192/1
$f/xe/\kappa/\mu$	0/0/192/192
Performance	
Total bandwidth (bytes)	164524
Public key (bytes)	163536
Encryption overhead (bytes)	988
Failure rate	2^{-149}
Classical/Quantum security (bits) – Core Sieving	
Primal attack	194/177
Dual attack	193/176
Hybrid attack	191/181
Sparse-secrets attack	192/175
Classical/Quantum optimal block size BKZ – Core Sieving	
Primal attack	632/632
Dual attack	628/628
Hybrid attack	624/648
Classical/Quantum security (bits) – Enumeration	
Primal attack	473/237
Dual attack	469/235
Hybrid attack	281/211
Sparse-secrets attack	469/234
Classical/Quantum optimal block size BKZ – Enumeration	
Primal attack	632/632
Dual attack	628/628
Hybrid attack	428/580

2.9.8 r5_cca_pke: CPU and Memory Requirements

This section contains the CPU and memory requirements for r5_cca_pke described in Section 2.9.7.

Table 15 shows the performance and memory usage figures for the optimized implementation of the R5ND_{1,3,5}PKE_0d algorithm. Table 16 shows the performance and memory usage figures for the optimized implementation of the R5ND_{1,3,5}PKE_5d algorithm. Table 17 shows the performance and memory usage figures for the optimized implementation of the R5N1_{1,3,5}PKE_0d $\tau = 2$ algorithm. Table 18 shows the performance of the parameter sets for specific use cases of the optimized implementation of r5_cca_pke.

Some conclusions from these tables are as follows:

- Due to the asymmetry of $\bar{n}$ and $\bar{m}$, the key generation time of R5N1_3PKE_0smallCT is relatively high; however, this is only a one-time cost since public-keys remain static for a long period of time. On the other hand, the benefit of the asymmetry is that the encryption time decreases.
- Ring versions of Round5 perform currently around 70 times faster than its non-ring versions.

Table 15: Optimized R5ND_{1,3,5}PKE_0d performance and memory usage

	Security Level		
	NIST1	**NIST3**	**NIST5**
API Parameters			
`CRYPTO_SECRETKEYBYTES`	708	1,031	1,413
`CRYPTO_PUBLICKEYBYTES`	676	983	1,349
`CRYPTO_BYTES`	756	1,119	1,525
Performance: Elapsed time (ms)			
PKE Generate Key Pair	0.03	0.04	0.04
PKE Encrypt	0.04	0.05	0.07
PKE Decrypt	0.06	0.07	0.09
Total	0.1	0.2	0.2
Performance: CPU Clock Cycles			
PKE Generate Key Pair	65.3k	88.0k	103.7k
PKE Encrypt	100.3k	137.7k	169.3k
PKE Decrypt	140.7k	191.6k	235.3k
Total	306.3k	417.3k	508k
Memory usage indication			
PKE Generate Key Pair	4,916B	7,150B	9,814B
PKE Encrypt	7,596B	10.8KiB	14.8KiB
PKE Decrypt	4,112B	6,014B	8,226B

Table 16: Optimized R5ND_{1,3,5}PKE_5d performance and memory usage

	Security Level		
	NIST1	**NIST3**	**NIST5**
API Parameters			
CRYPTO_SECRETKEYBYTES	493	828	1,042
CRYPTO_PUBLICKEYBYTES	461	780	978
CRYPTO_BYTES	636	950	1,301
Performance: Elapsed time (ms)			
PKE Generate Key Pair	0.02	0.04	0.06
PKE Encrypt	0.03	0.06	0.09
PKE Decrypt	0.05	0.09	0.1
Total	0.1	0.2	0.3
Performance: CPU Clock Cycles			
PKE Generate Key Pair	50.0k	91.0k	142.5k
PKE Encrypt	84.5k	158.9k	240.5k
PKE Decrypt	122.0k	236.7k	365.3k
Total	256.6k	486.6k	748k
Memory usage indication			
PKE Generate Key Pair	4,018B	6,168B	7,728B
PKE Encrypt	6,481B	9,746B	12.1KiB
PKE Decrypt	3,465B	5,258B	6,735B

Table 17: Optimized R5N1_{1,3,5}PKE_0d $\tau = 2$ performance and memory usage

	Security Level		
	NIST1	**NIST3**	**NIST5**
API Parameters			
`CRYPTO_SECRETKEYBYTES`	5,772	9,708	14,700
`CRYPTO_PUBLICKEYBYTES`	5,740	9,660	14,636
`CRYPTO_BYTES`	5,804	9,732	14,724
Performance: Elapsed time (ms)			
PKE Generate Key Pair	1.4	2.6	4.9
PKE Encrypt	2.0	3.9	7.4
PKE Decrypt	2.1	4.1	7.6
Total	5.5	10.6	19.9
Performance: CPU Clock Cycles			
PKE Generate Key Pair	3,520k	6,779k	12.7M
PKE Encrypt	5,312k	10.2M	19.2M
PKE Decrypt	5,423k	10.6M	19.6M
Total	14.3M	27.6M	51.5M
Memory usage indication			
PKE Generate Key Pair	36.4KiB	52.0KiB	73.1KiB
PKE Encrypt	46.6KiB	66.0KiB	92.4KiB
PKE Decrypt	31.4KiB	46.6KiB	67.0KiB

Table 18: Optimized r5_cca_pke specific use case parameters, performance and memory usage

	Parameter Set **R5N1_3PKE_0smallCT** $\tau = 2$
API Parameters	
`CRYPTO_SECRETKEYBYTES`	163,584
`CRYPTO_PUBLICKEYBYTES`	163,536
`CRYPTO_BYTES`	988
Performance: Elapsed time (ms)	
PKE Generate Key Pair	48.5
PKE Encrypt	1.0
PKE Decrypt	3.2
Total	53
Performance: CPU Clock Cycles	
PKE Generate Key Pair	125.8M
PKE Encrypt	2,598k
PKE Decrypt	8,346k
Total	136.8M
Memory usage indication	
PKE Generate Key Pair	892.7KiB
PKE Encrypt	453.8KiB
PKE Decrypt	446.5KiB

Table 19: r5_cpa_kem AVX2 optimized performance (CPU cycles)

Parameter Set	KeyGen	Enc	Dec	Total	Speed-up
R5N1_1KEM_0d $\tau = 2$	325.8k	393.1k	172.2k	891k	7.9x
R5N1_3KEM_0d $\tau = 2$	566k	676k	227.5k	1,469k	11.6x
R5N1_5KEM_0d $\tau = 2$	1,313k	1,493k	761k	3,566k	9.4x

Table 20: r5_cca_pke AVX2 optimized performance (CPU cycles)

Parameter Set	KeyGen	Enc	Dec	Total	Speed-up
R5N1_1PKE_0d $\tau = 2$	338.2k	467.5k	544k	1,350k	10.6x
R5N1_3PKE_0d $\tau = 2$	784k	1,013k	1,375k	3,172k	8.7x
R5N1_5PKE_0d $\tau = 2$	961k	1,246k	1,544k	3,751k	13.7x
R5N1_3PKE_0smallCT $\tau = 2$	22.3M	1,729k	7,338k	31.3M	4.4x

2.9.9 Performance on AVX2

Tables 19 and 20 show the performance of the AVX2 optimized implementation of r5_cpa_kem and r5_cca_pke respectively. The performance was measured the same way as the non-AVX2 optimized implementation (see Section 2.9.4 for details).

Table 21: Optimized (AVX2) R5N1_1KEM_0d – $\boldsymbol{A}$ generation variants

	Variants			
	$\tau = 0$	$\tau = 0$	$\tau = 1$	$\tau = 2$
	(cSHAKE128)	(AES128-HW)	(cSHAKE128)	(cSHAKE128)
Performance: Elapsed time (ms)				
KEM Generate Key Pair	1.4	0.2	0.1	0.1
KEM Encapsulate	1.5	0.3	0.2	0.2
KEM Decapsulate	0.07	0.06	0.07	0.07
Total	3.0	0.6	0.4	0.3
Performance: CPU Clock Cycles				
KEM Generate Key Pair	3,703k	610k	329.6k	325.8k
KEM Encapsulate	3,786k	692k	416.2k	393.1k
KEM Decapsulate	166.7k	164.8k	170.8k	172.2k
Total	7,656k	1,466k	917k	891k
Memory usage indication				
KEM Generate Key Pair	710.5KiB		2,088.8KiB	26.5KiB
KEM Encapsulate	723.9KiB		2,102.2KiB	39.9KiB
KEM Decapsulate	21.5KiB		21.5KiB	21.5KiB

2.9.10 Performance comparison of $f_{d,n}^{(\tau)}$

Table 21 compares the performance of R5N1_1KEM_0d for different choices of $f_{d,n}^{(\tau)}$ implemented using as DRBG either SHAKE128 or AES128 with hardware acceleration. These measurements are taken using the AVX2 optimized implementation. We observe that CPU performance of $f_{d,n}^{(0)}$ is worse than $f_{d,n}^{(2)}$ that is also worse than $f_{d,n}^{(1)}$. The reason is simple: $f_{d,n}^{(1)}$ requires the smallest amount of pseudorandom data to be obtained from the DRBG while $f_{d,n}^{(0)}$ requires the highest amount of pseudorandom data. In particular, in this configuration $f_{d,n}^{(2)}$ is a factor 10 faster than $f_{d,n}^{(0)}$ and it has low memory requirements. Thus, it is a good choice for small devices. $f_{d,n}^{(1)}$ is suitable for systems in which a central device, e.g., a server, has to handle many connections. Finally, we observe that if the CPU has hardware-enabled AES instructions, then AES-based generation of $\boldsymbol{A}$ is competitive.

2.9.11 Performance on Cortex M4

The performance of Round5 on a Cortex M4 has been reported in [93]. In the following, we report updated numbers after running Round5 on the popular STM32F407 Discovery board. This board was also used by the PQM4 project [65].

Table 22 gives preliminary cycle counts for the Cortex M4 implementation at submission time. Round5 maintains roughly equivalent or better performance than any comparable NIST candidate. As the only candidate that supports both ring-based and general lattices, the performance lead over Frodo [31] is especially noteworthy.

Table 23 gives a summary of other resource utilization of the implementation. We see that in addition to having shortest messages, also the RAM requirement by the implementation itself is usually smallest. We were unable to obtain reliable Flash sizes for other proposals, but we note that implementation footprint was one of our optimization criteria.

Table 22: Speed of primitive operations on a Cortex M4 (STM32F407VGT6 @ 24 MHz) target. Numbers are in 1000s of cycles for KG = keypair generation, Enc = encryption or encapsulation, Dec = decryption or decapsulation, and Tot = full "key exchange" operation. We are also including reported performance numbers for some other proposals for comparison. Note that our CCA "PKE" numbers include timing to perform actual AES-GCM key set-up and authenticated encryption or decryption/validation. Other candidates exclude this "DEM" operation.

Algorithm	**KG**	**Enc**	**Dec**	**Tot**
R5ND_1KEM_5d	391k	573k	244k	1,209k
R5ND_3KEM_5d	784k	1,083k	398k	2,266k
R5ND_5KEM_5d	1,429k	1,951k	693k	4,073k
R5ND_0KEM_2iot	341k	465k	191k	999k
R5ND_1KEM_4longkey	419k	624k	269k	1,313k
R5ND_1PKE_5d	365k	600k	753k	1,719k
R5ND_3PKE_5d	785k	1,212k	1,514k	3,512k
R5ND_5PKE_5d	1,360k	2,017k	2,551k	5,929k
R5ND_1KEM_0d	357k	470k	153k	982k
R5ND_3KEM_0d	1,170k	1,563k	551k	3,285k
R5ND_5KEM_0d	1,587k	2,135k	716k	4,439k
R5ND_1PKE_0d	487k	758k	923k	2,168k
R5ND_3PKE_0d	797k	1,233k	1,482k	3,513k
R5ND_5PKE_0d	1,093k	1,678k	1,992k	4,763k
R5N1_1KEM_0d $\tau = 2$	6,522k	4,329k	1,100k	11,953k
R5N1_3KEM_0d $\tau = 2$	9,923k	6,579k	1,750k	18,253k
R5N1_5KEM_0d $\tau = 2$	34,756k	19,851k	4,293k	58,900k
R5N1_1PKE_0d $\tau = 2$	4,052k	3,399k	3,784k	11,237k
R5N1_3PKE_0d $\tau = 2$	17,017k	11,166k	12,844k	41,028k
R5N1_5PKE_0d $\tau = 2$	23,178k	15,647k	17,315k	56,141k

Other proposals:

Kyber-512 [65]	726k	987k	1,018k	2,731k
Kyber-768 [65]	1,200k	1,446k	1,477k	4,123k
Kyber-1024 [65]	1,771k	2,142k	2,188k	6,101k
NewHope1024CCA [65]	1,243k	1,963k	1,978k	5,184k
Saber [64]	949k	1,232k	1,260k	3,441k
Frodo-640 cSHAKE [31]	81,299k	86,255k	87,212k	254,766k
Frodo-640 AES [31]	41,681k	45,758k	46,720k	134,159k
Frodo-640 xoshiro128 [31]	14,042k	14,657k	15,456k	44,155k

Table 23: Transfer, storage, stack usage, and implementation code size requirements of various variants on an ARM7-m architecture system (such as Cortex-M4). CT, PK, SK indicate the ciphertext (extension), public key, and secret key sizes in bytes. KG, Enc, and Dec give the memory requirement (in bytes) for keypair generation, encryption / encapsulation, and decryption / decapsulation. The code size number excludes shared components such as AES (for PKE DEM) and the Keccak permutation required by SHAKE.

	Xfer	Xfer	Priv	RAM	RAM	RAM	ROM
Algorithm	**CT**	**PK**	**SK**	**KG**	**Enc**	**Dec**	**Code**
R5ND_1KEM_5d	549	445	16	4,158	4,837	2,660	5,316
R5ND_3KEM_5d	859	780	24	5,846	6,941	3,708	6,622
R5ND_5KEM_5d	1,063	972	32	7,294	8,709	4,652	4,324
R5ND_0KEM_2iot	394	342	16	3,486	3,997	2,076	3,772
R5ND_1KEM_4longkey	563	453	24	4,094	4,837	2,772	5,354
R5ND_1PKE_5d	652	461	493	4,262	5,693	5,700	5,728
R5ND_3PKE_5d	966	780	828	5,902	8,069	8,076	7,132
R5ND_5PKE_5d	1,317	978	1,042	7,342	10,349	10,356	4,920
R5ND_1KEM_0d	682	634	16	4,806	5,605	2,348	2,622
R5ND_3KEM_0d	981	909	24	6,318	7,869	4,708	2,742
R5ND_5KEM_0d	1,274	1,178	32	7,798	9,933	5,964	2,770
R5ND_1PKE_0d	772	676	708	4,814	6,725	6,732	3,140
R5ND_3PKE_0d	1,135	983	1,031	6,430	9,261	9,268	3,248
R5ND_5PKE_0d	1,541	1,349	1,413	8,366	12,317	12,324	3,238
R5N1_1KEM_0d $\tau = 2$	5,236	5,214	16	19,334	24,381	17,484	3,166
R5N1_3KEM_0d $\tau = 2$	8,866	8,834	24	26,678	35,469	27,364	3,214
R5N1_5KEM_0d $\tau = 2$	14,288	14,264	32	40,358	54,597	45,260	3,330
R5N1_1PKE_0d $\tau = 2$	5,820	5,740	5,772	19,894	31,373	31,388	3,400
R5N1_3PKE_0d $\tau = 2$	9,748	9,660	9,708	29,950	49,357	49,364	3,756
R5N1_5PKE_0d $\tau = 2$	14,740	14,636	14,700	37,046	66,493	66,500	3,614
Other proposals: [65]							
Kyber-512	800	736	1,632	6,456	9,120	9,928	?
Kyber-768	1,152	1,088	2,400	10,544	13,720	14,880	?
Kyber-1024	1,504	1,440	3,168	15,664	19,352	20,864	?
NewHope1024CCA	2,208	1,824	3,680	11,152	17,448	19,648	?
Saber	1,088	992	2,304	13,248	15,528	16,624	?
Frodo-640 cSHAKE	9,736	9,616	19,872	26,272	41,472	51,848	?
Frodo-640 AES	9,736	9,616	19,872	31,116	51,444	61,820	?

2.10 Advantages and limitations

A single scheme with multiple parameter sets Round5 defines a unified scheme that can be configured with multiple parameter sets to fit the needs of multiple applications. Although this implies additional effort to define which parameter set an application requires, a unified scheme such as Round5 has advantages. Standardization effort is limited since a single scheme needs to be specified. This also prevents potential over-provisioning of resources (memory, bandwidth) since a device supporting Round5 can handle a wide range of parameter sets without big burdens. Finally, all Round5 parameter sets can be realized by means of a single library so that review and maintenance work is reduced.

Flexibility in the underlying problem's choice Round5 can be configured to rely on the Learning with Rounding (LWR) or Ring-Learning with Rounding (RLWR) problems, by controlling the input parameters n and d. This allows users to flexibly choose the parameter set (and underlying problem instantiation) that fits best their application and security requirements, even while Round5 is already in deployment. For instance, users dealing with strictly confidential information might only trust a public-key encryption (PKE) algorithm with a construction having no additional (e.g., ring) structure, while users operating a wireless network for a less critical application would prioritize low bandwidth and/or energy requirements and thus might prefer a (more efficient) ring-based construction. The unified approach provides from day one a contingency and smooth transition path, should vulnerabilities in ring-based constructions be discovered, since the non-ring based construction is already deployed.

Small public-keys and ciphertexts Round5 relies on rounding (specifically, the GLWR problem, see Section 2.3), leading to public-keys and ciphertexts with coefficients that have only $\lceil \log p \rceil$ and $\lceil \log t \rceil$ bits. Furthermore, Round5 relies on prime cyclotomic polynomials that provide a large pool of (q, n) values to choose from, allowing the selection of parameters that satisfy security requirements while minimizing bandwidth requirements. The usage of contant-time XEf error correction allows dealing with multiple errors so that even smaller parameters ($n = d, p$), and thus, messages are feasible. Round5 thus is suited for bandwidth-constrained applications, and Internet protocols that only allow limited packet sizes. Comparing with other submissions, as reported in [53] (with "Performance" as label for the X-axis), Round5 shows a very good trade-off between security and the sizes of public keys and ciphertexts.

Common building blocks for KEM and PKE By design, r5_cpa_kem and r5_cca_pke are constructed using common building blocks. This allows for a common security and correctness analysis for both schemes. Furthermore, it reduces the amount of code in devices, and the effort required for code-review of r5_cpa_kem and r5_cca_pke.

Flexibility in achieving security levels The design choices in Round5, especially the choice for prime cyclotomic polynomials, allows the fine-tuning of parameters to each NIST level. Round5 thus enables the user to choose parameters that tightly fit the required security level.

Flexibility for bandwidth Different applications can have different security needs and operational constraints. Round5 enables the flexible choice of parameters so that it is possible to adjust bandwidth requirements and security level according to the application needs. Round5 achieves this by using prime cyclotomic polynomials and rounding. For instance, parameter set R5ND_1KEM_5d for NIST security level 1 requires the exchange of 994 Bytes so that it can be used by an application with communication constraints; parameter set R5N1_5PKE_0d offers a much higher security level and does not rely on any ring structure so that it might be a preferred option for the exchange of highly confidential information. The amount of data to be exchanged for the latter parameter set (29360 Bytes) is larger than for the former, but is smaller than in another existing non-ring proposal for an equivalent security level [28].

Flexibility for CPU Different applications can have different security needs and operational constraints. Round5 allows for flexibility in the choice of parameters so that it is possible to adjust CPU needs and security requirements according to the application needs. Furthermore, the parameters in Round5 are chosen such that a unified implementation performs well for any input value (n, d). If necessary, optimized implementations for specific parameters can be realized, leading to even faster operation than with the unified implementation. For instance, parameter set R5N1_5KEM_0d for NIST security level 5, intended for a high security level and not relying on a ring structure, requires just a few milliseconds for key generation, encapsulation and decapsulation in our testing platform. Another application, requiring faster operation and with less security needs, can use parameter set R5ND_1KEM_5d for NIST security level 1 that performs orders of magnitude faster.

Flexibility for cryptographic primitives Round5 and its building blocks can be used to create cryptographic primitives such as authenticated key-exchange schemes in a modular way, e.g., as in [30].

Flexibility for integration into real-world security protocols The Internet relies on security protocols such as TLS, IKE, SSH, IPsec, DNSSEC etc. Round5 can be easily integrated into them because many of its parameter sets lead to relatively small messages and efficient CPU performance. For instance, the public-key and ciphertext in R5ND_5KEM_5d for NIST security level 5 require a total of 2035 Bytes. This is smaller than other lattice-based proposals such as [30] or [8], and facilitates integration into protocols so that required changes, such as packet fragmentation, are minimized. An example of a protocol for which direct integration of a KEM is challenging is IKEv2 (RFC 7296). The reason is that the first message exchange in IKEv2, *IKE_SA_INIT* does not support fragmentation. If we assume Ethernet (1500 B layer 2 packets), and we use the minimal header sizes, then there is room for a 1384 B public-key/ciphertext assuming IPv4, with IPv6, this is 1364 B. Still, this misses important information that is exchanged in most real-world deployments and that further reduces the available space. Examples of such information are notification/vendor ids, a cookie that the responder could use to decide whether it might be under a DoS attack (a minimum of 12 Bytes), and an initial contact notify that tells the responder that it is the first time we are talking to it and it should clear out any stale state (8 Bytes). If NAT traversal needs to be supported, then another 56 B are required for the corresponding notify. In general, most implementations might have enough space for perhaps 1250-1300 B; smaller than that makes things easy; larger than that forces implementations to make hard decisions. All ring-based Round5 parameter sets fit in IKEv2's *IKE_SA_INIT*.

Prevention of pre-computation and back-door attacks Round5 offers different alternatives to refresh $\boldsymbol{A}$ in a way that is efficient but also secure against pre-computation and back-door attacks. In the ring setting, this is achieved by computing a new $\boldsymbol{A}$. In the non-ring setting, three options are provided: ($f^{(0)}_{d,n}$) randomly generating $\boldsymbol{A}$ in each protocol exchange, ($f^{(1)}_{d,n}$) permuting a fixed and system-wide public parameter $\boldsymbol{A_{master}}$ in each protocol exchange, or ($f^{(2)}_{d,n}$) deriving $\boldsymbol{A}$ from a large pool of values. – determined in each protocol exchange – by means of a permutation. Permutation-based approaches show a performance advantage compared with generating $\boldsymbol{A}$ randomly— for instance, in settings in which a server has to process many connections. This is because the server can keep a fixed $\boldsymbol{A}$ in memory and just permute it according to the client request. If keeping a fixed $\boldsymbol{A}$ in memory in the client is an issue, then $f^{(2)}_{d,n}$ has a clear advantage compared with $f^{(1)}_{d,n}$ due to its lower memory needs.

Post-deployment flexibility A single implementation can be used for all Round5 parameter sets, without the need to recompile the code. This approach reduces the amount of code in the devices and makes code review easier. Furthermore, it enables a unified Round5 deployment that can be customized to fit non-ring or ring-based use cases, and a smooth transition to non-ring usage, should vulnerabilities in ring-based constructions be discovered.

Efficiency in constrained platforms The implementation of Round5 uses up to 16 bit long integers to represent the Round5 data. This enables good performance even in architectures with constrained processors. Moreover, algorithms are simple, and all modular arithmetic is easy as it is done with powers of two. Finally, the fixed-weight ternary secret distribution also facilitates efficient implementations.

Round5 incorporates the special parameter set R5ND_0KEM_2iot that has a total bandwidth requirement of just 736 Bytes, while ensuring a reasonable quantum-security level (at least 2^{88} workload for an adversary utilizing lattice reduction in the core sieving model [5]. This is feasible due to the usage of prime cyclotomic rings that allows configuring Round5 with a $n = d$ parameter between 256 and 512, the only two options available with power-of-two cyclotomic rings.

Parallelization Operations in Round5, for instance matrix multiplications, can be parallelized allowing for faster performance. This type of optimization has not been applied to its full extent yet.

Resistance against side-channel attacks The design of Round5 allows for efficient constant-time implementations, since by design all secrets are ternary and have a fixed number of ones and minus ones. This facilitates constant-time implementations in certain platforms, in particular, those that do not have a cache. The XEf error correction codes avoid conditions and table look-ups altogether and are therefore resistant against timing attacks.

Low failure probability The failure probability for all proposals for r5_cpa_kem (except the one for the IoT use case) is at most 2^{-64}. The failure probability for all proposals for r5_cca_pke is at most 2^{-142}. These failure probabilities can be achieved because of the usage of sparse, ternary secrets and the large pool of parameter sets. The usage of XEf error correction codes in some Round5 parameter sets plays a fundamental role to deal with multiple errors and decrease the overall failure probability.

Known underlying mathematical problems Round5 builds on the well-known Learning with Rounding and Ring Learning with Rounding problems.

Provable security We provide security reductions from the sparse ternary LWR and RLWR problems to r5_cpa_kem and r5_cca_pke, and from the standard Learning with Errors (LWE) problem to the sparse ternary LWR problem. Even though the latter reduction is not tight, it gives confidence in the Round5 design. As the parameters using error correction require performing certain operations modulo $x^{n+1} - 1$, the above security reductions do not apply to them. We describe an RLWE-based analogon of Round5 with error correction, and present a reduction for such a scheme to RLWE. This give confidence in Round5 with error correction. We discuss why this reduction does not directly translate to the RLWR case, and argue that it does not have any impact on the concrete security estimates.

Easy adoption A device that supports Round5 can handle a wide range of parameter sets, for both the ring and the non-ring versions, without recompilation. This flexibility will ease a smooth adoption.

Perfect forward secrecy Round5's key generation algorithm is fast, which makes it suitable for scenarios in which it is necessary to refresh a public/private key pair in order to maintain forward secrecy.

Application-specific parameter sets The flexibility of Round5 allows addressing the needs of a very different range of applications. Round5 has a special ring-based KEM parameter set, addressing Internet of Things applications, with very small bandwidth requirements while still providing a moderate security level. Additionally Round5 has a ring-based, NIST level 1 KEM parameter set in which the encapsulated key is 192-bits long instead of just 128-bits, so that the difficulty of attacking the encapsulated key (e.g., brute-forcing it using Grover's quantum search algorithm [52]) is approximately equal to the difficulty of a quantum-enabled, lattice-reduction based attack against Round5. This parameter set has a total bandwidth requirement of less than 1 KB. Finally, Round5 includes a non-ring based PKE parameter set with very small encryption overhead, also under 1 KB, at the cost of a larger public key. The latter parameter set thus is especially useful in applications with static or long-term public-keys, e.g., in secure email, while still providing the security guarantees of an unstructured lattice-based assumption.

2.11 Technical Specification of Reference Implementation

This section specifies Round5 from an implementation perspective. The description is close to Round5's C reference implementation and aims at explaining it and allowing other developers to create own implementations. Algorithms 16, 17, and 18 correspond to r5_cpa_kem proposed in the key encapsulation mechanism category. Algorithms 22, 23, and 24 correspond to r5_cca_pke proposed in the public-key encryption category. In the description, we avoid special symbols to facilitate its direct implementation. For instance, Greek letters are not included and, e.g., we write *kappa* instead of κ; we write *q_bits* instead of q_{bits} or *n_bar* instead of $\bar{n}$.

2.11.1 Round5 main parameters

Each Round5 parameter set (Sections 2.9.1 and 2.9.2) is determined by the following parameters that are located on the top of each parameter table in Sections 2.9.5 and 2.9.7.

d Number of polynomial coefficients per public matrix row.

n Reduction polynomial degree: ring: $n > 1$; non-ring: $n = 1$.

h Number of non-zero values per column in secret matrices.

q, q_bits Power of two modulo size where $q = 2^{q\_bits}$.

p, p_bits Power of two modulo size where $p = 2^{p\_bits}$.

t, t_bits Power of two modulo size where $t = 2^{t\_bits}$.

b, b_bits b_bits bits are extracted per ciphertext symbol; $b = 2^{b\_bits}$.

n_bar Number of columns of the secret matrix to compute $\boldsymbol{B}$.

m_bar Number of columns of the secret matrix to compute $\boldsymbol{U}$.

$kappa$ Security parameter; number of information bits in error-correcting code.

f Number of bit errors correctable by error-correcting code.

xe Number of parity bits of error correcting code.

mu Number of ciphertext symbols: $mu \cdot b\_bits \geq kappa + xe$.

2.11.2 Round5 derived or configurable parameters

The following parameters are derived from the main parameters. Parameter tau indicates a configuration choice in function $f_{d,n}^{(\tau)}$ used to generate $\boldsymbol{A}$.

d/n Number of polynomial elements in a row of $\boldsymbol{A}$.

$kappa\_bytes$ Security parameter in bytes, i.e., kappa/8.

z $z = \max(p, \frac{tq}{p})$. Relevant in reduction proofs and definition of rounding constants.

z_bits $z\_bits = \lceil log2(z) \rceil$.

h_1 $h\_1 = \frac{q}{2p}$ is a rounding constant.

h_2 $h\_2 = \frac{q}{2z}$ is a rounding constant.

h_3 $h\_3 = \frac{p}{2t} + \frac{p}{2b} - \frac{q}{2z}$ is a constant to reduce decryption bias.

h_4 $h\_4 = \frac{q}{2p} - \frac{q}{2z}$ is a constant to reduce decryption bias.

N Identifies polynomial $N_{n+1}(x) = x^{n+1} - 1$.

Phi Identifies polynomial $\Phi_{n+1}(x) = N_{n+1}(x)/(x-1)$.

Xi Identifies if reduction polynomial is N or Phi.

tau Index in $f_{d,n}^{(\tau)}$.

For all proposed Round5 parameter sets, $z = p$, $h\_1 = h\_2 = \frac{q}{2p}$, and $h\_4 = 0$.

2.11.3 Basic data types, functions and conversions

Bit

A binary digit: 0 or 1.

Bit string

An ordered sequence of bits, e.g., $[0, 1, 1, 1]$ contains 4 bits, 0 is the left-most bit. Bit strings are denoted by a lower case letter.

Byte

An ordered sequence of 8 bits, e.g., $[0, 1, 1, 1, 0, 1, 1, 1]$. Bytes are interpreted in little-endian. The previous example represents $0xEE$ in hexadecimal or 238 in decimal representation.

Byte string

An ordered sequence of bytes e.g., $[0x01, 0x02, 0x03]$ contains 3 bytes, 0x01 is the left-most byte. Byte strings are denoted by a lower case letter.

0^c

A bit string containing c 0s.

$||$

The concatenation operator is represented by $||$ and concatenates two strings, e.g., two bit or two byte stings. For instance,

$$[0, 1, 1, 1, 0, 1] = [0, 1, 1]||[1, 0, 1]$$

$\lceil a \rceil$

For a real number a, the ceiling operator is represented by $\lceil a \rceil$ and returns the next integer value of a. For instance, $\lceil 15/8 \rceil = 2$, $\lceil -3.2 \rceil = -3$.

$\lfloor a \rfloor$

For a real number a, the flooring operator is represented by $\lfloor a \rfloor$ and returns the integer part of value a. For instance, $\lfloor 15/8 \rfloor = 1$, $\lfloor -3.2 \rfloor = -4$.

$\oplus$

For two bits x and y, $x \oplus y$ represents the modulo 2 addition of two bits. For instance, $1 \oplus 1 = 0$.

Little endianness

Round5 uses a little endianness representation for bit strings and byte strings as defined above as well as for vectors and polynomials. In a bit string this means that the leftmost bit is the least significant bit and the rightmost bit or the last bit is the most significant bit. The same applies to byte string, vectors, and polynomials.

For instance, the following byte c contains 8 bits, starting with bit 0 (c_0) and ending with bit 7 (c_7).

$$c = [c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7]$$

In another example, a 64-bit word w contains 8 bytes such as the first byte is byte 0 (c_0) and the last byte is byte 7 (c_7).

$$w = [c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7]$$

For instance, $c = [0,1,0,1,1,1,0,1]$ equals 186, or $0xba$ in hexadecimal representation.
$w = [0xef, 0xcd, 0xab, 0x89, 0x67, 0x45, 0x23, 0x01]$ equals $0x0123456789abcdef$.

Moduli

Round5 performs operations modulo r with $r = \{q, p, t\}$ being powers of two. The number of bits is denoted as $r\_bits = \{q\_bits, p\_bits, t\_bits\}$ such that $r = 2^{r\_bits}$.

Vectors

Round5 vectors $\boldsymbol{v}_r^{(k,1)}$ are defined column-wise, and contain k integers in $\mathbb{Z}_r$. Vector element with index 0 $\boldsymbol{v}[0]$ comes first (little-endian) followed by element with index 1 $\boldsymbol{v}[1]$ till last element with index $k-1$, i.e., $\boldsymbol{v}[k-1]$. A vector element $\boldsymbol{v}[i]$ is represented as a bit string containing $r\_bits = \lceil log_2 r \rceil$ bits. Vectors are denoted by a bold low case letter.

Polynomials

Round5 polynomials are in $\mathcal{R}_{n,r} = \mathbb{Z}_r[x]/\left(\xi_{n+1}(x)\right)$ where $r = \{q, p, t\}$ is a power of two, $n+1$ is chosen to be a prime number, and $\xi_{n+1}(x)$ can be either the prime cyclotomic polynomial $\Phi(x) = 1 + x + \ldots + x^n$, or the NTRU polynomial $N_{n+1}(x) = x^{n+1} - 1 = (x-1) \cdot \Phi_{n+1}(x)$ having $n+1$ coefficients. All coefficients are $r\_bits = \lceil log_2(r) \rceil$ bits long, i.e., $r\_bits = \{q\_bits, p\_bits, t\_bits\}$. When $n = 1$, polynomials have a single coefficient in $\mathbb{Z}_r$.

Polynomials are represented in little endian, i.e., coefficient of degree 0 comes first and the highest degree coefficient comes last.

A polynomial is represented as a vector $\boldsymbol{poly}$ so that $\boldsymbol{poly}[i]$ is the coefficient of degree i. In particular,

$$\boldsymbol{poly} = \boldsymbol{poly}[0] + \boldsymbol{poly}[1] \cdot x + \cdots + \boldsymbol{poly}[s] \cdot x^s$$

where $s = n$ when $Xi = N$ and $s = n-1$ when $Xi = Phi$.

Each polynomial coefficient $\boldsymbol{poly}[i]$ is represented as a bit string of length r_bits.

$$\boldsymbol{poly}[i] = [\boldsymbol{poly}[i]\_0, \boldsymbol{poly}[i]\_1, \ldots, \boldsymbol{poly}[i]\_(r\_bits - 1)]$$

Polynomial vectors

Round5 polynomial vectors $\boldsymbol{v}_r^{(k,1)}$ are defined column-wise, and contain $k = d/n$ polynomials in $\mathcal{R}_{n,r} = \mathbb{Z}_r[x]/\left(\xi_{n+1}(x)\right)$. Polynomial vectors are denoted by a bold lower case letter.

Matrices

Round5 matrices $\boldsymbol{A}_r^{(row,col)}$ contain row rows and col columns and their elements are polynomials in $\mathcal{R}_{n,r} = \mathbb{Z}_r[x]/\left(\xi_{n+1}(x)\right)$. Matrices are denoted by a bold capital letter.

2.11.4 Supporting functions

SHAKE

Specified in FIPS-202, SHAKE with security levels 128 and 256 is used as the extendable-output function to generate the coefficients in public parameter $\boldsymbol{A}$ and secrets $\boldsymbol{S}$ and $\boldsymbol{R}$.

cSHAKE

Specified in NIST SP 800-185, cSHAKE128 and cSHAKE256 offer security levels 128 and 256 and build on top of SHAKE128 and SHAKE256 such that $cSHAKE(X, L, N = "", S = "") = SHAKE(X, L)$ where X, L, N and S are as in NIST SP 800-185. *cSHAKE* allows for customization bit strings so that
$cSHAKE(X1, L1, N1, S1)$ and
$cSHAKE(X1, L1, N2, S2)$ produce unrelated outputs unless $N1 = N2$ and $S1 = S2$.

cSHAKE is used in the construction of the permutations specified in $f^{(1)}_{d,n}$ and $f^{(2)}_{d,n}$.

Round5 operations are specified in terms of *cSHAKE* although some operations can be replaced by *SHAKE* as $cSHAKE(X, L, N = "", S = "") = SHAKE(X, L)$.

AES

Specified in FIPS-197, AES with security levels 128, 192 and 256 is used as symmetric block cipher in Round5. AES is also used as an optional alternative to generate random data.

randombytes_init

Function to initialize a *true* random number generator *randombytes()*.

randombytes()

Function that returns B bytes of true random data denoted as *randomness.*

$$randomness = randombytes(B)$$

hash()

Function to obtain an *output_len* byte long hash *output* of an *input_len* byte long *input* using a *customization_string* of a given length.
output = hash(output_len, input, input_len, customization_string, customization_string_len)
Round5 uses cSHAKE with a security strength of 128- or 256-bits as the default hash function due to its fast performance in software and the requirement to obtain hashes of variable length. If the S customization_string equals "", then Round5 hash is equivalent to SHAKE with a security strength of 128- or 256-bits:

$$output = hash(output\_leninput, input\_len)$$

Usage for different key lengths is specified in Table 24.

drbg_init_customization()

Function to initialize a Deterministic Random Bit Generator (DRBG) that takes as input a seed *seed* and a *customization_string*. This function also *hides* complexity of $f_{d,n}^{(\tau)}$ and its implementation using either cSHAKE or AES.

$$drbg\_init\_customization(seed, customization\_string)$$

Round5 uses cSHAKE as the default DRBG due to its fast performance in software. When cSHAKE is used, drbg_init_customization is implemented with cSHAKE_Absorb with a security strength of 128- or 256-bits.

$$drbg\_init\_customization = cSHAKE\_absorb(seed, "", customization\_string)$$

Round5 also supports AES as an optional DRBG due to its fast performance in hardware. When AES is used, drbg_init_customization involves two steps. First, it derives $seed_{AES}$ using *cSHAKE()*

$$seed\_AES = hash(seed\_len, seed, seed\_len, customization\_string, 2).$$

The AES key, for AES with a security level of *seed_len* bits, is set to $seed\_AES$. Usage for different key lengths is specified in Table 24.

drbg_init()

Function used instead of *drbg_init_customization()* when *customization_string* equals "". This function has an identical definition, but it already assumes *customization_string* = "", and thus, the usage of cSHAKE is equivalent to the direct usage of SHAKE.

drbg()

Function to obtain a *len*-byte long byte string denoted as *randomdata* from Round5's Deterministic Random Bit Generator (DRBG).

$$randomdata = drbg(len)$$

Round5 uses cSHAKE as the default DRBG due to its fast performance in software. When cSHAKE is used, Round5's DRBG implements cSHAKE_Squeeze with a security strength of 128- or 256-bits.

$$drbg(len) = cSHAKE\_Squeeze(len)$$

Round5 also supports AES as an optional DRBG due to its fast performance in hardware. When AES is used, Round5's DRBG applies AES in counter mode (NIST SP 800-38D) for a security level of 128-, 192- or 256-bits with input 16 byte long block of all zeros $0x00000000000000000000000000000000$.

$$drbg(len) = AESCTR(len)$$

The AES key is derived using drbg_init_customization(). Usage for different key lengths is specified in Table 24.

The retrieved bits from the underlying random bit function must always be interpreted as little endian. Thus, they will need to be reversed on a big endian machine.

drbg_sampler16 Function to sample random numbers uniformly distributed in a given range.

$$randomdata = drbg\_range16(range)$$

This function requires initializing the drbg with *drbg_init* before its first invocation. It first computes the greatest positive integer $range\_divisor$ such that $range\_divisor \cdot range < 2^{16}$. Then it uses *drbg(*$2^{16}$*)* to compute random numbers x till $x < range\_divisor \cdot range$. The returned value is $\lfloor x/range\_divisor \rfloor$

drbg_sampler16_2 Function to sample random numbers uniformly distributed in a given range that is a power of two.

$$randomdata = drbg\_range16\_2(range)$$

This function requires initializing the drbg with *drbg_init* before its first invocation. It uses *drbg(*2^{16}*)* to compute a 2 byte random number x. The returned value corresponds to the first $r\_bits = log2(range)$ bits of x, i.e., $[x\_0, x\_1, \ldots, x\_(r\_bits - 1)]$ For instance, if *drbg* returns [0x12, 0x34] and $range = 2^{12}$, then the sampled element is [0x12, 0x30].

2.11.5 Cryptographic algorithm choices

Round5 includes a drbg, a function to generate pseudorandom data, a hash function, and a DEM. These functions are implemented by means of cSHAKE (or optionally, AES in counter mode), and AES in GCM mode. The main reason for choosing cSHAKE is the use the customization string in the generation of the permutation in $f_{d,n}^{(\tau)}$. In other cases, the customization string is not included and in those cases cSHAKE is equivalent to SHAKE. The reason for including – optionally – AES in counter mode is its fast performance in platforms in which AES hardware acceleration is available. SHA-3 is specified in FIPS PUB 202;

Table 24: Cryptographic algorithm choices (Def. = default; Opt. = optional

	kappa	drbg_init	drbg	hash	DEM
Def.	128	cSHAKE128_absorb	cSHAKE128_squeeze	cSHAKE128	GCM-AES128
	192	cSHAKE256_absorb	cSHAKE256_squeeze	cSHAKE256	GCM-AES192
	256	cSHAKE256_absorb	cSHAKE256_squeeze	cSHAKE256	GCM-AES256
Opt.	128	hash Init CTR-AES128	CTR-AES-128	cSHAKE128	GCM-AES128
	192	hash Init CTR-AES192	CTR-AES-192	cSHAKE256	GCM-AES192
	256	hash Init CTR-AES256	CTR-AES-256	cSHAKE256	GCM-AES256

cSHAKE is specified in SP 800-185; AES is specified in FIPS 197; counter mode is specified in SP 800-38A; and GCM mode is specified in SP 800-38D.

The security strength of the algorithms is determined as a function of the length of the secret, *kappa*. Since cSHAKE is also defined with security strengths of 128− and 256− bits, cSHAKE128 is used when *kappa* = 128; otherwise, cSHAKE256 is used.

The default configuration (Def.) means that those are the default choices in Round5 code. This choice is motivated by the faster performance of cSHAKE in software compared with AES. The optional configuration (Opt.) means that those are optional choices in Round5 code. This choice is motivated by the fact that some platforms have hardware-enabled AES instructions that increase the performance of the pseudo random data generation.

2.11.6 Core functions

mult_poly_ntru — Function that multiplies input polynomials $\boldsymbol{a}$ and $\boldsymbol{b}$ in $\mathcal{R}_{n,r} = \mathbb{Z}_r[x]/\left(N_{n+1}(x)\right)$ obtaining polynomial $\boldsymbol{c}$.

$$\boldsymbol{c} = mult\_poly\_ntru(\boldsymbol{a}, \boldsymbol{b}, n, r)$$

lift_poly — Function that lifts input polynomial $\boldsymbol{a}$ in $\mathbb{Z}_r[x]/\left(\Phi_{n+1}(x)\right)$ obtaining polynomial$\boldsymbol{a^{(L)}}$ in $\mathbb{Z}_r[x]/\left(N_{n+1}(x)\right)$.

$$\boldsymbol{a^{(L)}} = lift\_poly(\boldsymbol{a}, n, r)$$

In mathematical terms, lifting means:

$$\boldsymbol{a}^{(L)} = (x-1)\boldsymbol{a}$$

Coefficient-wise, this means that:

$$\boldsymbol{a}^{(L)} = \boldsymbol{a}_0^{(L)} + \boldsymbol{a}_1^{(L)} \cdot x + \boldsymbol{a}_2^{(L)} \cdot x^2 + \cdots + \boldsymbol{a}_n^{(L)} x^n$$

equals

$$-\boldsymbol{a}_0 + (\boldsymbol{a}_0 - \boldsymbol{a}_1)x + (\boldsymbol{a}_1 - \boldsymbol{a}_2)x^2 + \cdots + (\boldsymbol{a}_{n-2} - \boldsymbol{a}_{n-1})x^{n-1} + \boldsymbol{a}_{n-1}x^n$$

unlift_poly

Function that unlifts input polynomial $\boldsymbol{a^{(L)}}$ in $\mathbb{Z}_r[x]/\left(N_{n+1}(x)\right)$, obtaining $\boldsymbol{a}$ in $\mathbb{Z}_r[x]/\left(\Phi_{n+1}(x)\right)$.

$$\boldsymbol{a} = unlift\_poly(\boldsymbol{a^{(L)}}, n, r)$$

In mathematical terms, unlifting means:

$$\boldsymbol{a} = \boldsymbol{a}^{(L)}/(x-1)$$

As $\boldsymbol{a}^{(L)} = lift_poly(\boldsymbol{a}, n, r)$, the coefficients of $\boldsymbol{a}$ can be recursively computed as

$$\boldsymbol{a}_0 = -\boldsymbol{a}_0^L \text{ and } \boldsymbol{a}_i = \boldsymbol{a}_{i-1} - \boldsymbol{a}_i^L \text{ for } 1 \leq i \leq n-1.$$

mult_poly

Function that takes input polynomials $\boldsymbol{a}$ and $\boldsymbol{b}$ of length n and multiplies them in $\mathcal{R}_{n,r} = \mathbb{Z}_r[x]/\left(\xi_{n+1}(x)\right)$ obtaining polynomial $\boldsymbol{c}$.

$$\boldsymbol{c} = mult\_poly(\boldsymbol{a}, \boldsymbol{b}, n, r, Xi)$$

If $Xi = N$, then $\boldsymbol{c} = \mathit{mult\_poly\_ntru}(\boldsymbol{a}||0, \boldsymbol{b}, n, r)$. If $Xi = Phi$, then this function lifts input parameter $\boldsymbol{a}$ before calling *mult_poly_ntru* and unlifts the result $\boldsymbol{c}$, i.e., $\boldsymbol{c} = \mathit{unlift}(\mathit{mult\_poly\_ntru}(\mathit{lift}(\boldsymbol{a}, n, r), \boldsymbol{b}, n, r), n, r)$.

add_poly

Function that takes input polynomials $\boldsymbol{a}$ and $\boldsymbol{b}$ of length n and adds them component-wise obtaining $\boldsymbol{c}$.

$$\boldsymbol{c} = add\_poly(\boldsymbol{a}, \boldsymbol{b}, n, r, Xi)$$

mult_matrix

Function that multiplies two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of dimension $A\_row \times A\_col$ and $B\_row \times B\_col$, with elements in $\mathcal{R}_{n,r} = \mathbb{Z}_r[x]/\left(\xi_{n+1}(x)\right)$ obtaining matrix $\boldsymbol{C}$ of dimension $A\_row \times B\_col$ whose elements are also in $\mathcal{R}_{n,r}$.

$$\boldsymbol{C} = mult\_matrix(\boldsymbol{A}, A\_row, A\_col, \boldsymbol{B}, B\_row, B\_col, n, r, Xi)$$

```
$for(i = 0; i < A\_row; i++)$
    $for(j = 0; j < B\_col; j++)$
      $\boldsymbol{C[i,j] = 0}$
      $for(k = 0; k < A\_col; k++)$
        $tmp = mult\_poly(\boldsymbol{A[i,k]}, \boldsymbol{B[k,j]}, n, r, Xi)$
        $\boldsymbol{C[i,j]} = add\_poly(\boldsymbol{C[i,j]}, tmp, n, r)$
```

transpose_matrix

Function that transposes input matrix $\boldsymbol{A}$ of dimension $A\_row \times A\_col$ with elements in $\mathbb{Z}[x]/\left(\Phi_{n+1}(x)\right)$, obtaining matrix $\boldsymbol{A\_T}$ of dimension $A\_col \times A\_row$ whose elements are also in $\mathbb{Z}[x]/\left(\Phi_{n+1}(x)\right)$. Polynomial coefficients are 16-bits integers.

$$\boldsymbol{A\_T} = transpose\_matrix(\boldsymbol{A}, A\_row, A\_col, n)$$

round_element

Function that rounds a_bits bits long element x from a_bits to b_bits bits using rounding constant h where $a = 2^{a\_bits}$ and $b = 2^{b\_bits}$.

$$x = round(x, a\_bits, b\_bits, h) = \left\lfloor \frac{x+h}{2^{a\_bits-b\_bits}} \right\rfloor = \left\lfloor \frac{b}{a}(x+h) \right\rfloor$$

round_matrix

Function that performs coefficient-wise rounding applying $round\_element$ to all polynomial coefficients in matrix $\boldsymbol{A}$ of dimension $A\_row \times A\_col$ with elements in $\mathcal{R}_{n,2^{a\_bits}} = \mathbb{Z}_{2^{a\_bits}}[x]/\left(\xi_{n+1}(x)\right)$, obtaining matrix $\boldsymbol{B}$ of dimension $A\_row \times A\_col$ with elements in $\mathcal{R}_{n,2^{b\_bits}} = \mathbb{Z}_{2^{b\_bits}}[x]/\left(\xi_{n+1}(x)\right)$.

$$\boldsymbol{B} = round\_matrix(\boldsymbol{A}, A\_row \cdot A\_col, n, a\_bits, b\_bits, h)$$

decompress

Function that decompresses element x from b_bits to a_bits bits .

$$x = decompress(x, b\_bits, a\_bits) = x \cdot 2^{a\_bits-b\_bits}$$

decompress_ matrix

Function that performs coefficient-wise decompression using $decompress$ of the first mu polynomial coefficients in matrix $\boldsymbol{A}$ of dimension $A\_row \times A\_col$ with elements in $\mathcal{R}_{n,2^{b\_bits}} = \mathbb{Z}_{2^{b\_bits}}[x]/\left(\xi_{n+1}(x)\right)$, obtaining vector $\boldsymbol{b}$ of dimension mu with elements in $\mathbb{Z}_{2^{a\_bits}}$.

$$\boldsymbol{b} = decompress\_matrix(\boldsymbol{A}, mu, 1, b\_bits, a\_bits)$$

permutation_tau_1

Function that computes permutation $\boldsymbol{p1}$ associated to $f_{d,1}^{(1)}$ used to permute the row elements in a fixed master public parameter matrix $\boldsymbol{A\_master}$ and obtain the public parameter A.

$$\boldsymbol{p1} = permutation\_tau\_1(sigma)$$

$\boldsymbol{p1}$ is a vector of length d with elements in $\mathbb{Z}_d$. permutation_tau_1($sigma$) is obtained by first initializing the DRBG with *drbg_init(sigma, customization_string=0x0001)* and then sampling the elements with *drbg_sampler16()* with range d.

permutation_tau_2

Function that computes permutation $\boldsymbol{p2}$ associated to $f_{d,1}^{(2)}$ used to permute the elements in a master public parameter vector $\boldsymbol{a\_master}$ and obtain the public parameter A.

$$\boldsymbol{p2} = permutation\_tau\_2(sigma)$$

$\boldsymbol{p2}$ is a vector of length d with distinct entries from $\mathbb{Z}_q$. Permutation_tau_2(sigma) is obtained by first initializing the DRBG with *drbg_init(sigma, customization_string=0x0001)* and then running *drbg_sampler16_2* with range q. Rejection sampling is used to ensure that the entries of $\boldsymbol{p2}$ are distinct.

create_A

Function that computes master public parameter $\boldsymbol{A}$ given random *sigma*.

$$\boldsymbol{A} = create\_A(sigma)$$

Internally, depending on the choice of *tau*, create_A computes $\boldsymbol{A}$ using one out of three strategies that provide a trade-off between CPU and memory performance (see Section 2.4.3).

$\mathbf{f_{d,n}^{(0)}}$: Applies *drbg_sampler16_2* with range q. This function must be initialized with seed *sigma* and without customization string. Output of *drbg_sampler16_2* is used to fill-in $\boldsymbol{A}$ row-wise, i.e., first polynomial element 0 in row 0, then polynomial element 1 in row 0, till all polynomial elements in row 0 are assigned; then polynomial element 0 in row 1, then polynomial element 1 in row 1, till all polynomial elements in row 1 are assigned. This process goes on till all elements in $\boldsymbol{A}$ are initialized. Each element in $\boldsymbol{A}$ is a polynomial, and it is filled-in coefficient-wise, i.e., first coefficient of degree 0, then coefficient of degree 1, till the coefficient of highest degree.

When $n = 1$, $\boldsymbol{A}$ consists of d^2 polynomial elements in $\mathcal{R}_{n,q} = \mathbb{Z}_q$. Since all Round5 parameter sets are such that $2^{16} \geq q > 2^8$, a total of d^2 calls to *drbg_sampler16_2* are required. When $n = d$, $\boldsymbol{A}$ consists of a single element in $\mathcal{R}_{n,r} = \mathbb{Z}_q[x]/\left(\Phi_{n+1}(x)\right)$. Since all Round5 parameter sets are such that $2^{16} \geq q > 2^8$, thus, a total of d calls to *drbg_sampler16_2* are required. In both cases, ($n = 1$ and $n = d$) element i is assigned the output of the i^{th} call to *drbg_sampler16_2*.

$\mathbf{f}^{(\mathbf{1})}_{\mathbf{d,n}}$: only applies to $n = 1$. This requires that a matrix $\boldsymbol{A\_master}$ of size $d \times d$ has been precomputed and is a public-parameter known to all parties in the system. It requires permutation vector $\boldsymbol{p1}$ – computed with *permutation_tau_1()* – of length d with elements randomly chosen in $\mathbb{Z}_d$ to permute the elements in each row in $\boldsymbol{A\_master}$ to obtain $\boldsymbol{A}$:
$\boldsymbol{A}[i, j] = \boldsymbol{A\_master}[i, (j + \boldsymbol{p1}[i]) \text{mod } d]$.

$\mathbf{f}^{(\mathbf{2})}_{\mathbf{d,n}}$: only applies to $n = 1$. This approach first computes a vector $\boldsymbol{a\_master}$ of length len_tau_2. This is done by calling len_tau_2 times *drbg_sampler16_2* with range q. len_tau_2 has a default value 2^{11} that is the smallest power of two value greater than the d value in any of the Round5 configurations. This function must be initialized with seed *sigma* and without customization string. It then uses a permutation vector $\boldsymbol{p2}$ of length d with distinct entries from $\mathbb{Z}_q$, obtained by applying *permutation_tau_2* to *sigma*, to pick up d sets of d consecutive elements in $\boldsymbol{a\_master}$ as rows of $\boldsymbol{A}$:
$\boldsymbol{A}[i, j] = \boldsymbol{a\_master}[(\boldsymbol{p2}[i] + j) \text{mod } q]$

create_ secret_vector Function that computes a sparse ternary secret $\boldsymbol{s}$ of dimension $length \times 1$ and having fixed hamming weight h.

$$\boldsymbol{s} = create\_secret\_vector(length, h)$$

Each secret vector $\boldsymbol{s}$ contains d/n polynomials in $\mathcal{R}_{n,r} = \mathbb{Z}_r[x]/(\xi_{n+1}(x))$. Since n can only take values 1 and d, then this secret vector can represent two different data structures always consisting of d elements in $\mathbb{Z}_r$:

- ring case: a single polynomial containing n ternary coefficients, with exactly $h/2$ "+1" and $h/2$ "−1" values.
- non-ring case: d polynomials, each of them having a single ternary coefficient, and the vector with exactly $h/2$ "+1" and $h/2$ "−1" values.

Since in both data structures, there are exactly $h/2$ "+1" and $h/2$ "−1" values, then this function only computes the h positions in vector $\boldsymbol{s}$ containing the h non-zero elements. To this end, this function uses *drbg_sample16* to sample the h non-zero positions with $i = 0, ..., h - 1$. This is done by checking whether a position $\boldsymbol{s}[i]$ is occupied already, or not. The i^{th} sampled position is assigned value “ -1” if i is odd and “ $+1$” if i is even as described below.

```
$\boldsymbol{s}[length] = 0$
$for(i = 0; i < h; i++)$
  $do$
    $x = drbg\_sampler16(d)$
  $while(\boldsymbol{s}[x]! = 0);$
  $if(is\_even(i))$
    $\boldsymbol{s}[x] = 1;$
  $else$
    $\boldsymbol{s}[x] = -1;$
```

create_S_T

Function that computes secret $\boldsymbol{S}^T$ of dimension $\bar{n} \times d$ and hamming weight h per row given function *create_secret_vector(d , h)*.

$\boldsymbol{S\_T} = create\_S\_T(\bar{n}, length = d/n{\cdot}n, hamming\_weight = h)$

create_R_T

Function that computes secret $\boldsymbol{R}^T$ of dimension $\bar{m} \times d$ and hamming weight h per row given function *create_secret_vector(d , h)*.

$\boldsymbol{R\_T} = create\_R\_T(\bar{m}, length = d/n{\cdot}n, hamming\_weight = h)$

sample_mu

Function that returns the first mu polynomial coefficients of $\boldsymbol{M}$ where $\boldsymbol{M}$ contains $\bar{n} \times \bar{m}$ polynomials in $\mathcal{R}_{n,r} = \mathbb{Z}_r[x]/\left(\xi_{n+1}(x)\right)$.

When $n = d$, $\boldsymbol{M}$ contains a single polynomial $\boldsymbol{m}$ and the values returned by *sample_mu* corresponds to the first mu polynomial coefficients:

$$m0 + m1x + \cdots + m_{mu-1}x^{mu-1}$$

When $n = 1$, $\boldsymbol{M}$ contains $\bar{n} \times \bar{m}$ polynomials, each of them having a single element in $\mathbb{Z}_r$ denoted as $M_{i,j}$ where i and j refer to the row and column indexes. The returned coefficients are the first mu coefficients in the matrix iterated row-wise.

$$\overbrace{\underbrace{M_{0,0}, M_{0,1}, \ldots, M_{0,\bar{n}-1}}_{\text{Row } 0}, \ldots \underbrace{M_{ii-1,0}, M_{ii-1,1}, \ldots, M_{ii-1,jj-1}}_{\text{Row } ii-1},}^{mu \text{ coefficients}}$$

where $mu = ii \cdot \bar{n} + jj$

add_msg

Function that converts a binary string message m of length $mu \cdot b\_bits$ into mu elements in $\mathbb{Z}_t$ and then adds them to a vector $\boldsymbol{x}$ of length mu whose elements are each t_bits long.

$$\boldsymbol{result} = add\_msg(len, \boldsymbol{x}, m, b\_bits, t\_bits)$$

Addition of message bits $m[i \cdot b\_bits, \ldots, (i+1) \cdot b\_bits - 1]$ is done by first multiplying this bit string interpreted as an element in $\mathbb{Z}_b$ by t/b so that this is represented in $\mathbb{Z}_t$. For instance, if $t = 2^5$ and $b = 2^2$, and $m[0,1] = [1,1]$, then its bit representation in $\mathbb{Z}_t$ is [0,0,0,1,1] where the left most bit is bit 0 and the right most bit is bit 4. This value is added to the t_bits-bit value $\boldsymbol{x}[0]$.

diff_msg

Function that computes difference of mu elements in $vecv$ and $\boldsymbol{x}$ modulo p.

$$\boldsymbol{result} = diff\_msg(len, \boldsymbol{v}, \boldsymbol{x}, p)$$

xef_compute

Round5's error correction operates on bit strings of size mu bits, of which $kappa$ bits are used to transport a shared secret message (with $kappa \in \{128, 192, 256\}$), and the remaining $xe = mu - kappa$ bits are used to correct errors. Note that $b\_bits = 1$ for all parameter sets using error correction. The XEf design is easily scalable, so that it is possible to use the parameter f to control the number of errors guaranteed to be corrected.

$$m1 = xef\_compute(m1, kappa\_bytes, f)$$

is the function that given a string $m1$ of length mu, computes the xe parity bits corresponding to the $kappa$ leftmost bits of $m1$, and XORs these parity bits with the xe rightmost bits of $m1$. The xe parity bits consist of the values of $2f$ registers. The lengths $\{l_0, l_1, \ldots, l_{2f-1}\}$ of registers $\{r_0, r_1, \ldots, r_{2f-1}\}$ for $XE(kappa, f)$, and the number of parity bits $xe = \sum l_i$ are as follows:

$kappa$	f	register lengths	xe
128	2	$\{11, 13, 14, 15\}$	53
128	4	$\{11, 13, 16, 17, 19, 21, 23, 29\}$	149
128	5	$\{16^{(*)}, 11, 13, 16, 17, 19, 21, 23, 25, 29\}$	190
192	5	$\{24^{(*)}, 13, 16, 17, 19, 21, 23, 25, 29, 31\}$	218
256	5	$\{16^{(*)}, 16, 17, 19, 21, 23, 25, 29, 31, 37\}$	234

Bit j in register r_i of length l_i is computed as:

$$r_i[j] = m1[j] \oplus m1[j+l_i] \oplus \cdots \oplus m1[j + \lfloor \frac{kappa - 1 - j}{l_i} \rfloor \cdot l_i]$$

As in HILA5, register r_0 in the XE5 codes is special – marked with $(*)$ – and computes the register bits as the block-wise XOR of $kappa/l_0$ consecutive string bits.

$$r_0[j] = m1[j \cdot \frac{kappa}{l_0}] \oplus \cdots \oplus m1[(j+1) \cdot \frac{kappa}{l_0} - 1]$$

The output bit string of *xef_compute* is:

$$m1 \oplus [0^{kappa}||r_0||r_1||\ldots||r_{2f-1}]$$

xef_fixerr

Function used for correcting up to f errors in a string of $mu = kappa + xe$ bits. For correcting a bit string $m1 = (m'||r')$, where m' has length $kappa$ and r' has length xe, first $m1 = xef\_compute(m1, kappa\_bytes, f)$ is computed. Then $m1 = (m'||r''' = r' \oplus r'')$, where the registers r''_i correspond to the parity bits for the message m'. Next,

$$m = xef\_fixerr(m1, kappa\_bytes, f),$$

is computed, where $m = (m[0], \ldots, m[kappa-1])$ is a bit string of length $kappa$ with

$$m[k] = \begin{cases} m'[k] \oplus 1 & \text{if } \sum_{i=0}^{2f-1} r'''_i[f_{r_i}(k)] > f, \\ m'[k] & \text{otherwise.} \end{cases}$$

Here $f_{r_i}(k)$ is the bit in register r_i that depends on message bit k as specified in xef_compute. See Section 2.4.1 for more background.

pack

Function that serializes an input vector ***input*** containing *input_length* components, each of size *input_size* bits and places it into an output message *output* of size $\lceil input\_length \cdot input\_size/8 \rceil$ bytes.

$$output = pack(\boldsymbol{input}, input\_length, input\_size)$$

For instance, input vector $[0x8, 0x1, 0xf, 0x2]$ contains 4 elements, each of them 4 bits long is serialized in output vector $[0x81, 0xf2]$. If the size is not a multiple of 8 bits, then the most significant bits of the last byte are padded with 0s.

pack_pk

Function that serializes the components *sigma* and $\boldsymbol{B}$ of Round5 public key. The function is defined as follows and uses *pack* internally.

$$pk = pack\_pk(sigma, ss\_size, \boldsymbol{B}, d/n \cdot n\_bar \cdot n, p\_bits)$$

By definition, *sigma* is packed in the first ss_size bytes of pk. Next, the $d/n \times \bar{n}$ n-coefficient polynomials in $\boldsymbol{B}$ are packed. Packing is done row-wise (first element 0 in row 0, then element 1 in row 0,..., till the last element in row 0; next the second row till the last row). For each polynomial element, packing is done starting with coefficient of degree 0 and ending with the coefficient of degree $n-1$. For each polynomial coefficient having p_bits bits, bit 0 comes first and bit p_bits-1 comes last.

pack_ct

Function that serializes the components $\boldsymbol{U}$ and $\boldsymbol{v}$ of Round5 ciphertext. The function is defined as follows and uses *pack* internally.

$$ct = pack\_ct(\boldsymbol{U}, d/n \cdot m\_bar \cdot n, p\_bits, \boldsymbol{v}, mu, t\_bits)$$

By definition, $\boldsymbol{U}$ is packed in the first $\lceil d/n \cdot m\_bar \cdot n \cdot p\_bits/8 \rceil$ bytes of ct. Packing is done row-wise (first element 0 in row 0, then element 1 in row 0,..., till the last element in row 0; next the second row till the last row). For each polynomial element, packing is done starting with coefficient of degree 0 and ending with the coefficient of degree $n-1$. For each polynomial coefficient having p_bits bits, bit 0 comes first and bit $p\_bits - 1$ comes last. Next, v is packed packing first $v[0]$, then $v[1]$, till $v[mu-1]$. Each coefficient $v[i]$ of v has t_bits bits and those are also packed following a little endian approach, i.e., first bit 0, then bit 1, till bit $t\_bits - 1$.

unpack

Function that deserializes a bit string *input* of length $\lceil number\_elements \cdot element\_bits/8 \rceil$ bytes into a vector of size $number\_elements$ in which each vector element is in $\mathbb{Z}_{2^{element\_bits}}$.

$$\boldsymbol{output} = unpack(input, number\_elements, element\_bits)$$

Bits $\{i * element\_bits, \ldots, (i+1) * element\_bits - 1\}$ in the input bit string correspond to element $\boldsymbol{output}[i]$ of the deserialized output vector.

unpack_pk

Function that de-serializes the components *sigma* and $\boldsymbol{B}$ of Round5's public-key *pk*. The function is defined as follows and uses *unpack* internally.

$$(\mathit{sigma}, \boldsymbol{B}) = \mathit{unpack\_pk}(\mathit{pk}, \mathit{sigma\_size}, \mathit{B\_elements}, \mathit{B\_element\_bits})$$

where the input parameters are the serialized public-key, the size of *sigma*, the number of polynomial coefficients in $\boldsymbol{B}$ and the size in bits of each polynomial coefficient.

unpack_ct — Function that de-serializes the components $\boldsymbol{U}$ and $\boldsymbol{v}$ of Round5's ciphertext *ct*. The function is defined as follows and uses *unpack* internally.

$$(\boldsymbol{U}, \boldsymbol{v}) = \mathit{unpack\_ct}(\mathit{ct}, \mathit{U\_elem}, \mathit{U\_elem\_size}, \mathit{v\_elem}, \mathit{v\_elem\_size})$$

where the input parameters are the serialized ciphertext, the number of polynomial coefficients in $\boldsymbol{U}$, the size in bits of each polynomial coefficient, the number of polynomial coefficients in $\boldsymbol{v}$, and the size in bits of each vector coefficient.

verify — Compares two byte strings $s1$ and $s2$ of equal length l and outputs bit 0 if they are equal.

$$c = \text{verify}(s1, s2, l)$$

conditional _constant _time_memcpy
Function that copies byte string a of length *a_bytes* starting at memory address *mem* if condition *cond* is true.

$$\mathit{conditional\_constant\_time\_memcpy}(\mathit{mem}, a, \mathit{a\_bytes}, \mathit{cond})$$

2.11.7 Implementation of r5_cpa_pke

Round5's INDCPA public-key encryption is specified in Algorithms 1, 2, and 3. Their implementation is described in Algorithms 13, 14, and 15.

Algorithm 13: r5_cpa_pke_keygen

```
output: $pk : kappa\_bytes + \lceil n\_bar \cdot d/n \cdot n \cdot p\_bits/8 \rceil$ byte string.
      : $sk : kappa\_bytes$ byte string.
$sigma = randombytes(kappa\_bytes)$
$\boldsymbol{A} = create\_A(sigma)$
$sk = randombytes(kappa\_bytes)$
$\boldsymbol{S\_T} = create\_S\_T(sk)$
$\boldsymbol{S} = transpose\_matrix(S\_T, n\_bar, d/n, n)$
$\boldsymbol{B} = mult\_matrix(\boldsymbol{A}, d/n, d/n, \boldsymbol{S}, d/n, n\_bar, n, q, Phi)$
$\boldsymbol{B} = round\_matrix(\boldsymbol{B}, d/n \cdot n\_bar, n, q\_bits, p\_bits, h1)$
$pk = pack\_pk(sigma, kappa\_bytes, b\_bits, d/n \cdot n\_bar \cdot n, p\_bits)$
return $pk, sk$
```

Algorithm 14: r5_cpa_pke_encrypt

```
input: $pk : kappa\_bytes + \lceil n\_bar \cdot d/n \cdot n \cdot p\_bits/8 \rceil$ byte string.
input: $m : kappa\_bytes$ byte string that is encrypted.
input: $rho : kappa\_bytes$ byte string.
output: $ct : (\lceil m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8 \rceil$ encrypted byte string.
$sigma, \boldsymbol{B} = unpack\_pk(pk, kappa\_bytes, d/n \cdot n\_bar \cdot n, p\_bits)$
$\boldsymbol{A} = create\_A(sigma)$
$\boldsymbol{R\_T} = create\_R\_T(rho)$
$\boldsymbol{A\_T} = transpose\_matrix(\boldsymbol{A}, d/n, d/n, n)$
$\boldsymbol{R} = transpose\_matrix(\boldsymbol{R\_T}, m\_bar, d/n, n)$
$\boldsymbol{U} = mult\_matrix(\boldsymbol{A\_T}, d/n, d/n, \boldsymbol{R}, d/n, m\_bar, n, q, Phi)$
$\boldsymbol{U} = round\_matrix(\boldsymbol{U}, d/n \cdot m\_bar, n, q\_bits, p\_bits, h2)$
$\boldsymbol{U\_T} = transpose\_matrix(\boldsymbol{U}, d/n, m\_bar, n)$
$\boldsymbol{B\_T} = transpose\_matrix(\boldsymbol{B}, d/n, n\_bar, n)$
$\boldsymbol{X} = mult\_matrix(\boldsymbol{B\_T}, n\_bar, d/n, \boldsymbol{R}, d/n, m\_bar, n, p, Xi)$
$\boldsymbol{x} = round\_matrix(\boldsymbol{sample\_mu(X)}, mu, 1, p\_bits, t\_bits, h2)$
$m1 = (m||0^{xe})$
$m1 = xef\_compute(m1, kappa\_bytes, f)$
$\boldsymbol{v} = add\_msg(mu, \boldsymbol{x}, m1, b\_bits, t\_bits)$
$ct = pack\_ct(\boldsymbol{U\_T}, d/n \cdot m\_bar \cdot n, p\_bits, \boldsymbol{v}, mu, t\_bits)$
return $ct$
```

Algorithm 15: r5_cpa_pke_decrypt

```
input: $sk : kappa\_bytes$ byte string.
       : $ct : (\lceil m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8\rceil$ byte string.
output: $m : kappa\_bytes$ byte string that has been decrypted.
$\boldsymbol{S\_T} = create\_S\_T(sk)$
$\boldsymbol{U\_T}, \boldsymbol{v} = unpack\_ct(ct, d/n \cdot m\_bar \cdot n, p\_bits, mu, t\_bits)$
$\boldsymbol{U} = transpose\_matrix(\boldsymbol{U\_T}, m\_bar, d/n, n)$
$\boldsymbol{v} = decompress\_matrix(\boldsymbol{v}, mu, 1, p\_bits, t\_bits)$
$\boldsymbol{X\_prime} = mult\_matrix(\boldsymbol{S\_T}, n\_bar, d/n, \boldsymbol{U}, d/n, m\_bar, n, p, Xi)$
$\boldsymbol{m2} = diff\_msg(mu, \boldsymbol{v}, \boldsymbol{sample\_mu(X\_prime)}, p)$
$\boldsymbol{m2} = round\_matrix(\boldsymbol{m2}, mu, 1, p\_bits, b\_bits, h3)$
$m1 = pack(\boldsymbol{m2}, mu, b\_bits)$
$m1 = xef\_compute(m1, kappa\_bytes, f)$
return $m = xef\_fixerr(m1, kappa\_bytes, f)$
```

2.11.8 Implementation of r5_cpa_kem

Round5's main building block is an INDCPA KEM are specified in Algorithms 4, 5, and 6. They are described from an implementation perspective in Algorithms 16, 17, and 18.

Algorithm 16: r5_cpa_kem_keygen

```
output: $pk : \lceil kappa\_bytes + n\_bar \cdot d/n \cdot n \cdot p\_bits/8\rceil$ byte string.
        : $sk : kappa\_bytes$ byte string.
$(pk, sk) = r5\_cpa\_pke\_keygen$
return $pk, sk$
```

Algorithm 17: r5_cpa_kem_encapsulate

```
input: $pk : \lceil kappa\_bytes + n\_bar \cdot d/n \cdot n \cdot p\_bits/8\rceil$ byte string.
output: $ct : (\lceil m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8\rceil$ byte string.
output: $k : kappa\_bytes$ byte string.
$m = randombytes(kappa\_bytes)$
$rho = randombytes(kappa\_bytes)$
$ct = r5\_cpa\_pke\_encrypt(pk, m, rho)$
$k = hash(kappa\_bytes, m||ct, "", kappa\_bytes + \lceil m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot$
  $t\_bits)/8\rceil, "", 0)$
return $(ct, k)$
```

Algorithm 18: r5_cpa_kem_decapsulate

```
input: $ct : (\lceil m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8\rceil$ byte string.
input: $sk : kappa\_bytes$ byte string.
output: $k : kappa\_bytes$ byte string.
$m = r5\_cpa\_pke\_decrypt(sk, ct)$
$k = hash(kappa\_bytes, m||ct, kappa\_bytes + \lceil m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot$
  $t\_bits)/8\rceil, "", 0)$
return $k$
```

2.11.9 Implementation of r5_cca_kem

Round5 relies on an INDCCA KEM specified in Algorithms 7, 8, and 9. They are described from an implementation perspective in Algorithms 19, 20, and 21.

Algorithm 19: r5_cca_kem_keygen

```
output: $pk : \lceil kappa\_bytes + n\_bar \cdot d/n \cdot n \cdot p\_bits/8 \rceil$ byte string.
      : $sk = sk\_cpa\_pke || y || pk : \lceil 3 \cdot kappa\_bytes + n\_bar \cdot d/n \cdot n \cdot p\_bits/8 \rceil$
        byte string.
$(pk, sk\_cpa\_pke) = r5\_cpa\_pke\_keygen$
$y = randombytes(kappa\_bytes)$
$sk = sk\_cpa\_pke || y || pk$
return $(pk, sk)$
```

Algorithm 20: r5_cca_kem_encapsulate

```
input: $pk : \lceil 3 \cdot kappa\_bytes + (n\_bar \cdot d/n \cdot n \cdot p\_bits)/8 \rceil$ byte string.
output: $ct : \lceil kappa\_bytes + (m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8 \rceil$ byte string.
output: $k : kappa\_bytes$ byte string.
$m = randombytes(kappa\_bytes)$
$L || g || rho =$
  $hash(3 \cdot kappa\_bytes, m || pk, \lceil 2 \cdot kappa\_bytes + (n\_bar \cdot d/n \cdot n \cdot p\_bits)/8 \rceil, "", 0)$
$(\boldsymbol{U\_T}, \boldsymbol{v}) = r5\_cpa\_pke\_encrypt(pk, m, rho)$
$ct = \boldsymbol{U\_T} || \boldsymbol{v} || g$
$k = hash(kappa\_bytes, L || ct, kappa\_bytes + \lceil (m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot$
  $t\_bits)/8 \rceil, "", 0)$
return $ct, k$
```

Algorithm 21: r5_cca_kem_decapsulate

```
input: $ct : \lceil kappa\_bytes + (m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8 \rceil$ byte string.
input: $sk = sk\_cpa\_pke || y || pk : \lceil 3 \cdot kappa\_bytes + n\_bar \cdot d/n \cdot n \cdot p\_bits/8 \rceil$ byte
       string.
output: $k : kappa\_bytes$ byte string.
$m' = r5\_cpa\_pke\_decrypt(sk\_cpa\_pke, ct)$
$L\_prime || g\_prime || rho\_prime =$
  $hash(3 \cdot kappa\_bytes, m' || pk, \lceil 2 \cdot kappa\_bytes + (n\_bar \cdot d/n \cdot n \cdot p\_bits)/8 \rceil, "", 0)$
$(\boldsymbol{U\_T\_prime}, \boldsymbol{v\_prime}) = r5\_cpa\_pke\_encrypt(pk, m\_prime, rho\_prime)$
$ct\_prime = \boldsymbol{U\_T\_prime} || \boldsymbol{v\_prime} || g\_prime$
$fail = verify(ct, ct\_prime, \lceil kappa\_bytes + (m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8 \rceil)$
$conditional\_constant\_time\_memcpy(hash\_input, y, kappa\_bytes, fail)$
$k = hash(kappa\_bytes, hash\_input, 2 \cdot kappa\_bytes + \lceil m\_bar \cdot d/n \cdot n \cdot p\_bits +$
  $mu \cdot t\_bits)/8 \rceil, "", 0)$
return $k$
```

2.11.10 Implementation of r5_cca_pke

Round5 relies on an INDCCA PKE specified in Algorithms 10, 11, and 12. They are described from an implementation perspective in Algorithms 22, 23, and 24. These algorithms rely on *r5_dem()* and *r5_dem_inverse()* algorithms that in Round5 are implemented by means of AES in GCM mode according to FIPS 197 and SP 800-38D.

```
Algorithm 22: r5_cca_pke_keygen
  output: $pk : \lceil kappa\_bytes + n\_bar \cdot d/n \cdot n \cdot p\_bits/8 \rceil$ byte string.
        : $sk : \lceil 3 \cdot kappa\_bytes + n\_bar \cdot d/n \cdot n \cdot p\_bits/8 \rceil$ byte string.
  $(pk, sk) = r5\_cca\_kem\_keygen()$
  return $(pk, sk)$
```

```
Algorithm 23: r5_cca_pke_encrypt
  input: $m : m\_len$ byte string containing the message to be encrypted.
  input: $m\_len$: message length, integer smaller than $2^{64}$
  input: $pk : \lceil 3 \cdot kappa\_bytes + (n\_bar \cdot d/n \cdot n \cdot p\_bits)/8 \rceil$ byte string.
  output: $ct = c1||c2 : \lceil kappa\_bytes + (m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8 \rceil + c2\_len$
          byte string.
  output: $clen$: ciphtertext length, integer smaller than $2^{64}$
  $(c1, k) = r5\_cca\_kem\_encapsulate(pk)$
  $(c2, c2\_len) = r5\_dem(k, kappa\_bytes, m, m\_len)$
  $ct = c1||c2$
  $clen = \lceil kappa\_bytes + (m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8 \rceil + c2\_len$
  return $(ct, clen)$
```

```
Algorithm 24: r5_cca_pke_decrypt
  input: $ct = c1||c2 : \lceil kappa\_bytes + (m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8 \rceil + c2\_len$
         byte string encoding the ciphertext.
  input: $clen$ : ciphertext length, integer smaller than $2^{64}$
  input: $sk : \lceil 3 \cdot kappa\_bytes + n\_bar \cdot d/n \cdot n \cdot p\_bits/8 \rceil$ byte string.
  output: $m : m\_len$ byte string containing the decrypted message.
  output: $m\_len$: message length, integer smaller than $2^{64}$
  $c2\_len = clen - \lceil kappa\_bytes + (m\_bar \cdot d/n \cdot n \cdot p\_bits + mu \cdot t\_bits)/8 \rceil$
  $k = r5\_cca\_kem\_decapsulate(c1, sk)$
  $(m, m\_len) = r5\_dem\_inverse(k, kappa\_bytes, c2, c2\_len)$
  return $(m, m\_len)$
```

3 Description of contents in digital media

The contents of the digital media is structured as follows. In the root directory the following files and directories can be found:

`README` A file listing the content of the media.

`Supporting_Documentation` Directory with the documentation of our submission.

`Reference_Implementation` Directory with the source code for the reference implementation of our submission.

`Optimized_Implementation` Directory with the source code for the optimized (fixed parameters) implementation of our submission.

`Additional_Implementations` Directory with the source code for additional implementations.

`KAT` Directory with the KAT files.

The contents of each directory is detailed below.

3.1 Supporting documentation

The `Supporting_Documentation` directory contains the documentation of our submission in PDF form.

`Round5_Submission.pdf` The main submission document.

`Speedtest` Directory with the source files for the application with which we performed our performance measurements (see `speedTest/README` for information).

`Correctness` Directory with source code for analyzing the correctness of parameter sets (see the `README` in this directory for information).

`Parameters` Directory with a script for summarizing the parameter sets (see the `README` in this directory for information).

3.2 Reference, and Optimized implementations

For the reference (found inside directory `Reference_Implementation`), configurable (found inside directory `Configurable_Implementation`), and optimized (found inside directory `Optimized_Implementation`) implementations, the subdirectories `kem` and `encrypt` contain the implementations of our KEM and PKE algorithms, respectively.

Inside those subdirectories a further directory level can be found for each of the algorithm variants and NIST levels. The naming convention for these subdirectories is as shown in the following table:

`R5`	Fixed text
`ND`, or `N1`	To indicate a ring (`ND`), or non-ring (`N1`) variant
_	Fixed text
level	To indicate the NIST level (1, 3, or 5) or 0 for no NIST level.
`KEM` or `PKE`	The type of the algorithm
_	Fixed text
xef	The number of bit-errors to be corrected.
version	Version identifier, `d` for the standard parameters, extttiot for iot parameter sets, etc.

Inside the directories with the variant, you can find the complete source code of that variant:

`Makefile` A makefile with which the KAT generation executable can be created.

`PQCgetKAT_*.c` The NIST source code for the generation of KATs for the variant.

`api.h` The API functions and settings in use for the variant.

`*.h` and `*.c` The header and source code files of our submission.

In tables 25 we have listed all submitted KEM and PKE variants.

3.3 Additional implementations

The directory `Additional_Implementations` contains additional implementations of our submission.

`AVX2` Directory with the source code for an AVX2 optimized implementation. This version also has cache attack countermeasures enabled.

`Configurable` Directory with the source code for an optimized, but configurable at runtime, implementation.

3.4 KAT files

The files with the Known Answer Test values can be found in the `KAT` directory. Underneath that directory, the KAT files can be found in subdirectories using the same structure as the implementations.

For instance the KAT files for the "R5N1_3PKE_0d" variant can be found in directory `KAT/encrypt/R5N1_3PKE_0d`.

Inside the KAT file directory, the KAT files can be found named according to the NIST specification, including the number of bytes of the secret key as part of the name and the use of the suffixes `.req`, `.int`, and `.rsp` to denote the request, intermediate output, and response files respectively.

We have provided KAT files for all submitted algorithm variants.

Table 25: Submitted algorithm variants

KEM	**PKE**
R5ND_1KEM_0d	R5ND_1PKE_0d
R5ND_1KEM_5d	R5ND_1PKE_5d
R5ND_3KEM_0d	R5ND_3PKE_0d
R5ND_3KEM_5d	R5ND_3PKE_5d
R5ND_5KEM_0d	R5ND_5PKE_0d
R5ND_5KEM_5d	R5ND_5PKE_5d
R5ND_0KEM_2iot	R5N1_1PKE_0d
R5ND_1KEM_4longkey	R5N1_3PKE_0d
R5N1_1KEM_0d	R5N1_5PKE_0d
R5N1_3KEM_0d	R5N1_3PKE_0smallCT
R5N1_5KEM_0d	

4 IPR Statements

2.D.1 Statement by Each Submitter

I, Hayo Baan, High Tech Campus 5, 5656 AE Eindhoven, The Netherlands, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

□ *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5;* ***OR*** *(check one or both of the following):*

□ *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*

■ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of th~~e~~ public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from

consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Hayo Baan
Title: Software Developer
Date: 5 March 2019
Place: Eindhoven, The Netherlands

2.D.1 Statement by Each Submitter

I, Sauvik Bhattacharya, of High Tech Campus 34, 5656AE Eindhoven, the Netherlands, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

> *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5;* ***OR*** *(check one or both of the following):*
>
> > *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*
> >
> > ■ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from

consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Sauvik Bhattacharya

Signed: Sauvik Bhattacharya
Title: Scientist
Date: 4 March 2019
Place: Eindhoven, the Netherlands

2.D.1 Statement by Each Submitter

I, Scott Fluhrer, 31 Massand Rd, North Attleborough, MA, USA, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

□ *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5;* ***OR*** *(check one or both of the following):*

> □ *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5, may be covered by the following U.S. and/or foreign patents: ____ (describe and enumerate or state "none" if applicable)____ ;*
>
> ☒ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from

consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Scott Fluhrer
Title: Principal Engineer
Date: 3-11/19 3/11/19
Place: North Attleboro, MA, 02760

2.D.1 Statement by Each Submitter

I, Oscar Garcia-Morchon, High Tech Campus 5, 5656 AE Eindhoven, The Netherlands, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

□ *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as ____ (print name of cryptosystem)____;* ***OR*** *(check one or both of the following):*

□ *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as ____ (print name of cryptosystem)____, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*

■ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of th~~e~~ public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived

cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Oscar Garcia-Morchon
Title: Senior Cryptography Architect
Date: March 12, 2019
Place: Eindhoven, The Netherlands

2.D.1 Statement by Each Submitter

I, Thijs Laarhoven, Eindhoven University of Technology, The Netherlands, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

- □ *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5;* ***OR*** *(check one or both of the following):*

  - □ *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*

  - ☒ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from

consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Thijs Laarhoven
Title: dr.ir.
Date: March 4th, 2019
Place: Eindhoven

TLaarhoven

2.D.1 Statement by Each Submitter

I, Rachel Player, Information Security Group, McCrea Buidling, Royal Holloway, University of London, Egham Hill, Egham, Surrey, TW20 0EX, UK, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

□ *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5;* ***OR*** *(check one or both of the following):*

□ *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*

■ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances *made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.*

Signed: Dr. Rachel Player
Title: Postdoctoral Researcher
Date: 4 March 2019
Place: Egham, UK

R. Player

2.D.1 Statement by Each Submitter

I, Ronald Rietman, Philips Research, High Tech Campus 34, 5656AE Eindhoven, The Netherlands, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

> I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5; **OR** (check one or both of the following):
>
> > to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;
> >
> > ☒ I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from

consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Ronald Rietman
Title: Senior scientist
Date: 2019-3-4
Place: Eindhoven

2.D.1 Statement by Each Submitter

I, Markku-Juhani Olavi Saarinen, PQShield Ltd., Prama House, 267 Banbury Road, Oxford OX2 7HQ,United Kingdom, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

- □ *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5;* ***OR*** *(check one or both of the following):*

  - □ *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*

  - ■ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived

1/2

cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Markku-Juhani O. Saarinen

Title: Senior Cryptography Engineer

Date: March 5, 2019

Place: OXFORD, UK

2.D.1 Statement by Each Submitter

I, Ludo Tolhuizen, High Tech Campus 34, 5656 AE Eindhoven, The Netherlands, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

> *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5;* ***OR*** *(check one or both of the following):*
>
> > *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*
> >
> > ■ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from

consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

L. Tolhuizen

Signed: Ludo Tolhuizen
Title: Senior Scientist
Date: March 4, 2019
Place: Eindhoven, The Netherlands

2.D.1 Statement by Each Submitter

I, Mr. JOSÉ LUIS TORRE ARCE, address Avenida BILBAO, nº 9, C.P. 39600 Muriedas (Spain), do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

□ *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as ____ (print name of cryptosystem)____;* ***OR*** *(check one or both of the following):*

- □ *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as ____ (print name of cryptosystem)____, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*

- ■ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived

cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: NAME...JOSÉ LUIS TORRE ARCE
Title: TITLE......Mr.
Date: DATE......2019.03.08
Place: PLACE...Santander

2.D.1 Statement by Each Submitter

I, Zhenfei Zhang, 888 Boylston St. Boston, MA, U.S., do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5 (formerly Round2 and Hila5), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check one):

- *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5;* ***OR*** *(check one or both of the following):*

  - *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Round5, may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*

  ■ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such *implementations for the purposes of the public review and evaluation process.*

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

3/11/2019

Signed: Zhenfei Zhang
Title: Cryptography engineer
Date: March 11, 2019
Place: 888 Boylston st. Boston, MA

2.D.2 Statement by Patent (and Patent Application) Owner(s)

If there are any patents (or patent applications) identified by the submitter, including those held by the submitter, the following statement must be signed by each and every owner, or each owner's authorized representative, of each patent and patent application identified.

I, Jako Eleveld , of High Tech Campus 5, 5656 AE Eindhoven, The Netherlands, am the owner or authorized representative of the owner Koninklijke Philips N.V. of the following patent(s) and/or patent application(s): EP17156214, EP17170508, EP17159296, EP17196812, EP17196926, EP18194118 and do hereby commit and agree to grant to any interested party on a worldwide basis, if the cryptosystem known as Round5 (formerly Round2 and Hila5) is selected for standardization, in consideration of its evaluation and selection by NIST, a non-exclusive license for the purpose of implementing the standard (check one):

~~without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination,~~ **OR**

☒ *under reasonable terms and conditions that are demonstrably free of any unfair discrimination.*

I further do hereby commit and agree to license such party on the same basis with respect to any other patent application or patent hereafter granted to me, or owned or controlled by me, that is or may be necessary for the purpose of implementing the standard.

I further do hereby commit and agree that I will include, in any documents transferring ownership of each patent and patent application, provisions to ensure that the commitments and assurances made by me are binding on the transferee and any future transferee.

I further do hereby commit and agree that these commitments and assurances are intended by me to be binding on successors-in-interest of each patent and patent application, regardless of whether such provisions are included in the relevant transfer documents.

I further do hereby grant to the U.S. Government, during the public review and the evaluation process, and during the lifetime of the standard, a nonexclusive, nontransferrable, irrevocable, paid-up worldwide license solely for the purpose of modifying my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability) for incorporation into the standard.

Signed: Jako Eleveld
Title: Head of IP Licensing
Date: 27 FEB 2019
Place: Eindhoven, The Netherlands

2.D.3 Statement by Reference/Optimized Implementations' Owner(s)

The following must also be included:

I, Ali El Kaafarani, of Prama House, 267 Banbury Rd, Oxford OX2 7HQ, UK, am the owner or authorized representative of the owner PQShield Ltd. of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the post-quantum algorithm public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed: Ali El Kaafarani
Title: CEO of PQShield.
Date: 04/03/2019
Place: OXFORD, UK.

2.D.3 Statement by Reference/Optimized Implementations' Owner(s)

The following must also be included:

I, Jako Eleveld , High Tech Campus 5, 5656 AE Eindhoven, The Netherlands, am the owner or authorized representative of the owner Koninklijke Philips N.V. of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the post-quantum algorithm public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed: Jako Eleveld
Title: Head of IP Licensing
Date: 27 FEB 2019
Place: Eindhoven, The Netherlands

A Formal security of Round5

This section contains details on the formal security of Round5, its algorithms and building blocks. It is organized as follows: Section A.1 gives an overview of the security reduction for Round5 when replacing the GLWR public parameter $\boldsymbol{A}$ sampled from a truly uniform distribution with one generated by the instantiation $f_{d,n}^{(0)}$ of the Round5 function $f_{d,n}^{(\tau)}$. Section A.2 introduces notions of security based on indistinguishablity of ciphertexts or encapsulated keys. Section A.3 gives results (existing and new) on the hardness of our underlying problem, or rather its two specific instances we use – the Learning with Rounding problem with sparse ternary secrets ($\texttt{LWR}_{\text{spt}}$) and the Ring Learning with Rounding problem with sparse ternary secrets ($\texttt{RLWR}_{\text{spt}}$). Sections A.6 and A.7 contain the first main result of this section – a proof of IND-CPA security for r5_cpa_kem (Theorem A.6.1), and a proof of IND-CCA security of r5_cca_pke (Theorems A.7.1 and A.7.2), assuming the hardness of the above problems. Finally, Section A.8 and Theorem A.8.1 contain the second main result of this reduction: a proof of the hardness for $\texttt{LWR}_{\text{spt}}$, the underlying problem of our schemes for the non-ring case (i.e., for $n = 1$) in the form of a polynomial-time reduction to it from the Learning with Errors (LWE) problem with secrets uniformly chosen from $\mathbb{Z}_q^d$ and errors drawn according to a Gaussian distribution.

A.1 Deterministic generation of $\boldsymbol{A}$

The General Learning with Rounding (GLWR) public parameter $\boldsymbol{A}$ in Round5 is generated using the function $f_{d,n}^{(\tau)}$ from a short random seed (see Section 2.4.3). The core component in $f_{d,n}^{(\tau)}$ responsible for deterministically expanding this short random seed into a longer random sequence is either AES(128 or 256) [49] or CSHAKE(128 or 256) [48]. In order to relate Round5's security to the hardness of the GLWR problem, we reuse Naehrig *et al.*'s argument in [79] to argue that we can replace a uniformly sampled matrix $\boldsymbol{A} \in \mathcal{R}_{n,q}^{d/n \times d/n}$ with matrices sampled according to Round5's key-generation algorithm when it uses $f_{d,n}^{(0)}$, for both AES and CSHAKE, while considering a realistic adversary with access to the seed. The proof for both AES and CSHAKE proceeds by using the notion of indifferentiability [74, 39, Def. 3], in exactly the same manner as in [79, Sec. 5.1.4].

We explain the intuition behind the proof in case of AES, considering $f_{d,n}^{(0)}$. Let $\mathcal{F}$ denote an "ideal domain expansion" primitive that expands a short random seed, block-wise, into a larger sequence, such that each block is unique and also sampled uniformly at random. In our security reductions, the GLWR public parameter $\boldsymbol{A}$ is generated by querying the GLWR oracle. It can be shown that marginally increasing the number of calls to the GLWR oracle makes it possible to construct a GLWR matrix $\boldsymbol{A}$ that fits (with high probability) the output distribution of $\mathcal{F}$, without deteriorating the problem's hardness [79, Sec. 5.1.4]. Next, we consider a construction $\mathcal{C}^{\mathcal{G}}$ in the Ideal Cipher model implementing $\mathcal{F}$

as AES (as in Round5). It can be shown that $\mathcal{C}^{\mathcal{G}}$ is indifferentiable from $\mathcal{F}$ [79, Sec. 5.1.4]. This therefore allows us to replace the uniformly random sampling of $\boldsymbol{A}$ in the GLWR problem with one generated as in Round5's $f_{d,n}^{(0)}$ without affecting security.

Next, we explain the intuition behind the proof when CSHAKE is used in $f_{d,n}^{(0)}$. In the random oracle model, CSHAKE is an ideal XOF [48]. It can be shown that [79, Sec. 5.1.4] CSHAKE can be modeled as an ideal hash function used to expand a seed into each row of the matrix $\boldsymbol{A}$, each step being independent, thereby expanding the uniformly random seed into a larger uniformly random matrix. This construction implements the ideal functionality $\mathcal{F}$ perfectly, completing the proof. We refer to [79, Sec. 5.1.4] for details.

A.2 Security Definitions

Security requirements for public-key encryption and key-encapsulation schemes are based on whether static or ephemeral keys are used. (Static) public-key encryption (PKE) schemes, and nominally ephemeral key-encapsulation mechanisms that allow key caching that provide security against adaptive chosen ciphertext attack (corresponding to IND-CCA2 security) are considered to be sufficiently secure [80, Section 4.A.2]. On the other hand, a purely ephemeral key encapsulation mechanism (KEM) that provides semantic security against chosen plaintext attack, i.e., IND-CPA security, is considered to be sufficiently secure [80, Section 4.A.3].

Definition A.2.1 (Distinguishing advantage)**.** *Let $\mathcal{A}$ be a randomized algorithm that takes as input elements from a set X with output in $\{0,1\}$. Let D_1 and D_2 two probability distributions on X. The advantage of $\mathcal{A}$ for distinguishing between D_1 and D_2, denoted by $Adv_{D_1,D_2}(\mathcal{A})$, is defined as*

$$Adv_{D_1,D_2}(\mathcal{A}) = |Pr[\mathcal{A}(x) = 1 \mid x \leftarrow D_1] - Pr[\mathcal{A}(x) = 1 \mid x \leftarrow D_2] \mid$$

Table 26: IND-CPA game for PKE

```
1. $(pk, sk) = \texttt{KeyGen}(\lambda)$.
2. $b \xleftarrow{\$} \{0,1\}$
3. $(m_0, m_1, st) = \mathcal{A}(pk)$ such that $|m_0| = |m_1|$.
4. $c = \texttt{Enc}(pk, m_b)$
5. $b' = \mathcal{A}(pk, c, st)$
6. return $[b' = b]$
```

Definition A.2.2 (IND-CPA Secure PKE)**.** *Let PKE=(`Keygen`, `Enc`, `Dec`) be a public key encryption scheme with message space $\mathcal{M}$. Let λ be a security parameter. The IND-CPA game is defined in Table 26, and the IND-CPA advantage of* *an adversary $\mathcal{A}$ against PKE is defined as*

$$Adv_{PKE}^{IND\text{-}CPA}(\mathcal{A}) \ = | \ Pr[\textsf{IND-CPA}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2} \ | \ .$$

Definition A.2.3 (IND-CCA Secure PKE)**.** *Let PKE=(*`Keygen`*,* `Enc`*,* `Dec`*) be a public key encryption scheme with message space $\mathcal{M}$. Let λ be a security parameter. Let $\mathcal{A}$ be an adversary against PKE. The IND-CCA game is defined as Table 26, with the addition that $\mathcal{A}$ has access to a decryption oracle* $\texttt{Dec}(\cdot) = \texttt{Dec}(sk, \cdot)$ *that returns* $m' = \texttt{Dec}(sk, \texttt{Enc}(pk, m'))$*, and the restriction that $\mathcal{A}$ cannot query* $\texttt{Dec}(\cdot)$ *with the challenge c. The IND-CCA advantage of $\mathcal{A}$ against PKE is defined as*

$$Adv_{PKE}^{IND\text{-}CCA}(\mathcal{A}^{Dec(\cdot)}) \ = | \ Pr[\textsf{IND-CCA}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2} \ | \ .$$

Table 27: IND-CPA game for KEM

```
1. $(pk, sk) = \texttt{KeyGen}(\lambda)$.
2. $b \xleftarrow{\$} \{0, 1\}$
3. $(c, K_0) = \texttt{Encaps}(pk)$
4. $K_1 \xleftarrow{\$} \mathcal{K}$
5. $b' = \mathcal{A}(pk, c, K_b)$
6. return $[b' = b]$
```

Definition A.2.4 (IND-CPA Secure KEM)**.** *Let KEM=(*`Keygen`*,* `Encaps`*,* `Decaps`*) be a key encapsulation mechanism with key space $\mathcal{K}$. Let λ be a security parameter. The IND-CPA game is defined in Table 27, and the IND-CPA advantage of an adversary $\mathcal{A}$ against KEM is defined as*

$$Adv_{KEM}^{IND\text{-}CPA}(\mathcal{A}) \ = | \ Pr[\textsf{IND-CPA}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2} \ | \ .$$

Definition A.2.5 (IND-CCA Secure KEM)**.** *Let KEM=(*`Keygen`*,* `Encaps`*,* `Decaps`*) be a key encapsulation mechanism with key space $\mathcal{K}$. Let λ be a security parameter. Let $\mathcal{A}$ be an adversary against KEM. The IND-CCA game is defined in Table 27, with the addition that $\mathcal{A}$ has access to a decapsulation oracle* $\texttt{Decaps}(\cdot) = \texttt{Decaps}(sk, \cdot)$ *that returns* $K' = \texttt{Decaps}(sk, c')$ *where* $(c', K') = \texttt{Encaps}(pk)$*, and the restriction that $\mathcal{A}$ cannot query* $\texttt{Decaps}(\cdot)$ *with the challenge c. The IND-CCA advantage of $\mathcal{A}$ against KEM is defined as*

$$Adv_{KEM}^{IND\text{-}CCA}(\mathcal{A}^{Decaps(\cdot)}) \ = | \ Pr[\textsf{IND-CCA}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2} \ | \ .$$

In the random oracle model [22] (both for IND-CPA and IND-CCA games), the adversary is given access to a random oracle H that it can query up to a polynomial number q_H of times. In a post-quantum setting, it can be assumed that the adversary has access to a quantum accessible random oracle H_Q [26] that can be queried up to q_{H_Q} times on arbitrary superpositions of input strings.

A.3 Hardness Assumption (Underlying Problem)

In this section, we detail the underlying problem on whose hardness the security of our schemes are established. Depending on whether the system parameter n is chosen to be 1 or d (see Section 2.4.3), the proposed public-key encryption and key-encapsulation mechanism are instantiated either as non-ring (LWR) based or ring (RLWR) based schemes. The security of the proposals are therefore based on the hardness of the Decision-General Learning with Rounding problem with sparse-ternary secrets, i.e., $\texttt{dGLWR}_{\text{spt}}$ (see Section 2.3). We first recall hardness results for LWR before introducing our own results on the hardness of $\texttt{dLWR}_{\text{spt}}$. We then recall hardness results of RLWR.

Provable Security in the non-ring case: The hardness of the LWR problem has been studied in [19, 9, 25, 18] and established based on the hardness of the Learning with Errors (LWE) problem [88]. The most recent work are two independent reductions to LWR from LWE: the first, due to Bai et al. [18, Theorem 6.4] preserves the dimension n between the two problems but decreases the number of LWR samples that may be queried by an attacker by a factor (p/q); the second due to Bogdanov et al. [25, Theorem 3] preserves the number of samples between the two problems but increases the LWR dimension by a factor $\log q$. We follow the approach of Bai since it results in a smaller LWR dimension, leading to smaller bandwidth requirements and better performance. We note that the requirements on the number of samples is met, as $\boldsymbol{A}$ is a $d/n \times d/n$ matrix.

Bai et al.'s reduction [18, Theorem 6.4] is an essential component that we use in Section A.8 to prove a reduction from $\texttt{dLWE}_{n,m',q,D_\alpha}(\mathcal{U}(\mathbb{Z}_q))$ to $\texttt{dLWR}_{n,m,q,p}(\mathcal{U}(\mathcal{H}_n(h)))$, i.e., to $\texttt{dLWR}_{\text{spt}}$.

Provable Security in the ring case: Next, we recall hardness results for the ring case, i.e., for Decision-RLWR [19]. To the best of our knowledge, the only existing result on the hardness of Decision-RLWR is due to [19, Theorem 3.2], who show that Decision-RLWR is at least as hard as Decision-RLWE as long as the underlying ring and secret distribution remain the *same* for the two problems, the RLWE noise is sampled from *any* (balanced) distribution in $\{-B, \dots, B\}$, and q is super-polynomial in n, i.e., $q \geq pBn^{\omega(1)}$. The last condition may be too restrictive for practical schemes. Hence, although [19, Theorem 3.2] is relevant for the provable (IND) security of our schemes' ring-based instantiations, it remains to be seen whether the above reduction can be improved to be made practical.

A.4 IND-CPA Security of r5_cpa_pke

In this section it is shown that the public-key encryption scheme r5_cpa_pke is IND-CPA secure, based on the hardness of the decision GLWR problem with sparse-ternary secrets. In Section A.7 this result will be used to show that r5_cca_pke is IND-CCA Secure. The proof is restricted to the case that $\xi(x) = \Phi_{n+1}(x)$. Referring to Table 26, we take for `KeyGen` the function r5_cpa_pke_keygen, and for `Enc` the function r5_cpa_pke_encrypt(pk, m, ρ) with uniform choice for ρ. That is,

$$\rho \xleftarrow{\$} \{0,1\}^{\kappa}; \mathtt{Enc}(pk, m) = \text{r5\_cpa\_pke\_encrypt}(pk, m, \rho).$$

Let n, m, p, q, d, h be positive integers with $n \in \{1, d\}$. The hard problem underlying the security of our schemes is decision-GLWR with sparse-ternary secrets (see Section 2.3). For the above parameters, we define the GLWR oracle $O_{m,\chi_S,\boldsymbol{s}}$ for a secret distribution χ_S that returns m GLWR samples as follows:

$$O_{m,\chi_S,\boldsymbol{s}} : \boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{m\times d/n},\ \boldsymbol{s} \leftarrow \chi_S;\ \text{return}\ (\boldsymbol{A}, \mathrm{R}_{q\to p}(\boldsymbol{A}\boldsymbol{s})) \tag{37}$$

The decision-GLWR problem with sparse-ternary secrets is to distinguish between the distributions $\mathcal{U}(\mathcal{R}_{n,q}^{m\times d/n}) \times \mathcal{U}(\mathcal{R}_{n,p})$ and $O_{m,\chi_S,\boldsymbol{s}}$, with $\boldsymbol{s}$ common to all samples and $\chi_S := \mathcal{U}(\mathcal{H}_{n,d/n}(h))$. For an adversary $\mathcal{A}$, we define

$$\mathrm{Adv}_{d,n,m,q,p}^{\mathtt{dGLWR}_{\mathrm{spt}}}(\mathcal{A}) =$$

$$|\Pr\left[\mathcal{A}(\boldsymbol{A},\boldsymbol{b}) = 1 \mid (\boldsymbol{A},\boldsymbol{b}) \xleftarrow{\$} O_{m,\chi_S,\boldsymbol{s}}\right] - \Pr\left[\mathcal{A}(\boldsymbol{A},\boldsymbol{b}) = 1 \mid \boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{m\times d/n}, \boldsymbol{b} \xleftarrow{\$} \mathcal{R}_{n,p}^{m}\right]|$$

For an extended form of the decision-GLWR problem with the secret in form of a matrix consisting of $\overline{n}$ independent secret vectors, we define a similar oracle $O_{m,\chi_S,\overline{n},\boldsymbol{S}}$ as follows:

$$O_{m,\chi_S,\overline{n},\boldsymbol{S}} : \boldsymbol{A} \xleftarrow{\$} \mathcal{U}\left(\mathcal{R}_{n,q}^{m\times d/n}\right),\ \boldsymbol{S} \leftarrow (\chi_S)^{\overline{n}};\ \text{return}\ (\boldsymbol{A}, \mathrm{R}_{q\to p}(\boldsymbol{A}\boldsymbol{S})) \tag{38}$$

The advantage of an adversary for this extended form of the decision-GLWR problem is defined in a similar manner as above.

The following theorem shows that r5_cpa_pke is IND-CPA secure assuming the hardness of decision-GLWR with sparse-ternary secrets.

Theorem A.4.1. *Let p, q, t be integers such that $t|p|q$, and let $z = \max(p, tq/p)$. Furthermore, assume that $\xi(x) = \Phi_{n+1}(x)$. If $f_{d,n}^{(\tau)}$, f_S and f_R induce distributions indistinguishable from uniform, then r5_cpa_pke is IND-CPA secure under the hardness assumption of the Decision-GLWR problem with sparse-ternary secrets. More precisely, for every IND-CPA adversary $\mathcal{A}$, if $Adv_{CPA\text{-}PKE}^{IND\text{-}CPA}(\mathcal{A})$ is the advantage in winning the IND-CPA game, then there exist distinguishers $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}$ such that*

$$\begin{aligned} Adv_{CPA\text{-}PKE}^{IND\text{-}CPA}(\mathcal{A}) \leq {} & Adv_{\mathcal{U}(\mathcal{R}_{n,q}^{d/n\times d/n}),F_\tau}(\mathcal{B}) + Adv_{G_S,\chi_S^{\overline{n}}}(\mathcal{C}) + Adv_{G_R,\chi_S^{\overline{m}}}(\mathcal{D}) \\ & + \overline{n}\cdot Adv_{d,n,d/n,q,p}^{\mathtt{dGLWR}_{spt}}(\mathcal{E})\ + Adv_{d,n,d/n+\overline{n},q,z}^{\mathtt{dGLWR}_{spt}}(\mathcal{F}) \end{aligned} \tag{39}$$

Table 28: IND-CPA games for r5_cpa_pke: games G_0 and G_1

Game $\boldsymbol{G_0}$	**Game $\boldsymbol{G_1}$**
1. $\sigma \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{A} = f^{(\tau)}_{d,n}(\sigma)$	1. $\sigma \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{A} = f^{(\tau)}_{d,n}(\sigma)$
2. $s \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{S} = f_S(s)$	2. $s \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{S} = f_S(s)$
3. $\boldsymbol{B} = \mathrm{R}_{q\to p}(\langle \boldsymbol{AS}\rangle_{\Phi_{n+1}})$	3. $\boldsymbol{B} = \mathrm{R}_{q\to p}(\langle \boldsymbol{AS}\rangle_{\Phi_{n+1}})$
4. Choose $\beta \xleftarrow{\$} \{0,1\}$.	4. Choose $\beta \xleftarrow{\$} \{0,1\}$.
5. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$	5. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$
6. $\rho \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{R} = f_R(\rho)$	6. $\rho \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{R} = f_R(\rho)$
7. $\boldsymbol{U} = \mathrm{R}_{q\to p,h_2}(\langle \boldsymbol{A}^T\boldsymbol{R}\rangle_{\Phi_{n+1}})$	7. $\boldsymbol{U} = \mathrm{R}_{q\to p,h_2}(\langle \boldsymbol{A}^T\boldsymbol{R}\rangle_{\Phi_{n+1}})$
8. $\boldsymbol{v} = \langle \mathrm{Sample}_\mu\left(\mathrm{R}_{p\to t,h_2}(\langle \boldsymbol{B}^T\boldsymbol{R}\rangle_\xi)\right) + \frac{t}{b} ECC\_Enc_{\kappa,f}(m_\beta)\rangle_t$	8. $\boldsymbol{V} = \langle\left(\mathrm{R}_{p\to t,h_2}(\langle \boldsymbol{B}^T\boldsymbol{R}\rangle_\xi)\right) + \frac{t}{b}\boldsymbol{M_\beta}\rangle_t$
9. $\beta' = \mathcal{A}\left((\boldsymbol{A},\boldsymbol{B}), (\boldsymbol{U}, \boldsymbol{v}), st\right)$	9. $\beta' = \mathcal{A}\left((\boldsymbol{A},\boldsymbol{B}), (\boldsymbol{U}, \mathrm{Sample}_\mu(\boldsymbol{V}), st\right)$
10. **return**$[(\beta' = \beta)]$.	10. **return**$[(\beta' = \beta)]$.

In this equation, F_τ is the distribution of $f^{(\tau)}_{d,n}$ with $\sigma \xleftarrow{\$} \{0,1\}^\kappa$, G_S is the distribution of $f_S(s)$ with $s \xleftarrow{\$} \{0,1\}^\kappa$ and G_R is the distribution of $f_R(\rho)$ with $\rho \xleftarrow{\$} \{0,1\}^\kappa$. Moreover, $Adv^{dGLWR_{spt}}_{d,n,m,q_1,q_2}(\mathcal{Z})$ is the advantage of adversary $\mathcal{Z}$ in distinguishing m GLWR samples (with sparse-ternary secrets) from uniform, with the GLWR problem defined for the parameters d, n, q_1, q_2.

Theorem A.4.1 via a sequence of IND-CPA games shown in Tables 28 to 31, following the methodology of Peikert et al. in [81, Lemma 4.1] and that of [30, Theorem 3.3]. Steps for combining samples using rounding from q to p and samples using rounding from p to t are due to [44]. Game G_0 is the actual CPA-PKE game. For convenience, the steps of r5_cpa_pke_encrypt(pk, m, ρ) are written out explicitly. Moreover, we write χ_S for the uniform distribution on $\mathcal{H}_{n,d/n}(h)$.
In Game G_1, $\boldsymbol{M}_\beta$ is the matrix that has zero coefficients in all positions not picked up by Sample_μ and satisfies $ECC\_Enc_{\kappa,f}(m_\beta) = \mathrm{Sample}_\mu(\boldsymbol{M}_\beta)$. Clearly,

$$\Pr(S_0) = \Pr(S_1) \tag{40}$$

Games G_1 and G_2 only differ in the generation of $\boldsymbol{A}$. A distinguisher between $\mathcal{U}(\mathcal{R}^{d/n\times d/n}_{n,q})$ and F_τ, defined as the distribution of $f^\tau_{d,n}(\sigma)$ with $\sigma \xleftarrow{\$} \{0,1\}^\kappa$, can be constructed as follows. On input $\boldsymbol{A} \in \mathcal{R}^{d/n\times d/n}_{n,q}$, perform steps 2-10 of game G_1. The output b' is distributed as in game G_1 if $\boldsymbol{A}$ is distributed according to F_τ, and is distributed as in game G_2 if $\boldsymbol{A}$ is distributed uniformly. We conclude that there is a distinghuisher $\mathcal{B}$ such that

$$\mathrm{Adv}_{\mathcal{U}(\mathcal{R}^{d/n\times d/n}_{n,q}),F_\tau}(\mathcal{B}) = \mid \Pr(S_1) - \Pr(S_2) \mid . \tag{41}$$

Table 29: IND-CPA games for r5_cpa_pke: games G_2 and G_3

Game $\boldsymbol{G_2}$	**Game $\boldsymbol{G_3}$**
1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$	1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$
2. $s \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{S} = f_S(s)$	2. $\boldsymbol{S} \xleftarrow{\$} \chi_S^{\overline{n}}$
3. $\boldsymbol{B} = \mathrm{R}_{q \to p}(\langle \boldsymbol{AS} \rangle_{\Phi_{n+1}})$	3. $\boldsymbol{B} = \mathrm{R}_{q \to p}(\langle \boldsymbol{AS} \rangle_{\Phi_{n+1}})$
4. Choose $\beta \xleftarrow{\$} \{0,1\}$.	4. Choose $\beta \xleftarrow{\$} \{0,1\}$.
5. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$.	5. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$.
6. $\rho \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{R} = f_R(\rho)$	6. $\rho \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{R} = f_R(\rho)$
7. $\boldsymbol{U} = \mathrm{R}_{q \to p, h_2}(\langle \boldsymbol{A}^T \boldsymbol{R} \rangle_{\phi_{n+1}})$	7. $\boldsymbol{U} = \mathrm{R}_{q \to p, h_2}(\langle \boldsymbol{A}^T \boldsymbol{R} \rangle_{\phi_{n+1}})$
8. $\boldsymbol{V} = \langle \big(\mathrm{R}_{p \to t, h_2}(\langle \boldsymbol{B}^T \boldsymbol{R} \rangle_\xi)\big) + \frac{t}{b} \boldsymbol{M}_\beta \rangle_t$	8. $\boldsymbol{V} = \langle \big(\mathrm{R}_{p \to t, h_2}(\langle \boldsymbol{B}^T \boldsymbol{R} \rangle_\xi)\big) + \frac{t}{b} \boldsymbol{M}_\beta \rangle_t$
9. $\beta' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \mathrm{Sample}_\mu(\boldsymbol{V})), st\right)$	9. $\beta' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \mathrm{Sample}_\mu(\boldsymbol{V})), st\right)$
10. **return**$[(\beta' = \beta)]$.	10. **return**$[(\beta' = \beta)]$.

Games G_2 and G_3 only differ in the generation of the secret matrix $\boldsymbol{S}$. We denote the distribution of $f_S(s)$ with $s \xleftarrow{\$} \{0,1\}^\kappa$ by G_S. Similarly to the reasoning for Games G_1 and G_2, there is a distinguisher $\mathcal{C}$ such that

$$\mathrm{Adv}_{G_S, \chi_S^{\overline{n}}}(\mathcal{C}) = \mid \Pr(S_2) - \Pr(S_3) \mid . \tag{42}$$

Games G_3 and G_4 only differ in the generation of $\boldsymbol{B}$. Consider the following algorithm. On input $(\boldsymbol{A}, \boldsymbol{B}) \in \mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$, run steps 3-10 from game G_3. As $(\mathbf{A}, \mathbf{B})$ is distributed like $O_{m, \chi_S, \overline{n}, \boldsymbol{S}}$ in game G_3 and uniformly in game G_4, we conclude that there is a distinguisher $\mathcal{X}$ such that

$$\mathrm{Adv}_{O_{m, \chi_S, \overline{n}, \boldsymbol{S}, \mathcal{U}(\mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}})}}(\mathcal{X}) = \mid \Pr(S_3) - \Pr(S_4)) \mid \tag{43}$$

By using a standard hybrid argument, we infer that there is a distinguisher $\mathcal{E}$ such that

$$\mathrm{Adv}_{O_{m, \chi_S, \overline{n}, \boldsymbol{S}, \mathcal{U}(\mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}})}}(\mathcal{X}) \leq \overline{n} \cdot \mathrm{Adv}_{O_{m, \chi_S, \overline{1}, \boldsymbol{s}, \mathcal{U}(\mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n \times 1})}}(\mathcal{E}) \tag{44}$$

Games G_4 and G_5 only differ in the generation of the secret matrix $\boldsymbol{R}$. We denote the distribution of $f_R(\rho)$ with $\rho \xleftarrow{\$} \{0,1\}^\kappa$ by G_R. Similarly to the reasoning for games G_1 and G_2, there is a distinguisher $\mathcal{D}$ such that

$$\mathrm{Adv}_{G_R, \chi_S^{\overline{m}}}(\mathcal{D}) = \mid \Pr(S_4) - \Pr(S_5) \mid . \tag{45}$$

Now consider Games G_5 and G_6. As p divides q, $\langle \boldsymbol{B}_q \rangle_p$ is uniformly distributed. By definition of the rounding function,

$$\boldsymbol{V'} \equiv \lfloor \frac{t}{p}(\langle {\boldsymbol{B_q}}^T \boldsymbol{R} \rangle_\xi + h_2 \boldsymbol{J}) \rfloor + \frac{t}{b} \boldsymbol{M}_\beta \pmod{\frac{tq}{p}}.$$

Table 30: IND-CPA games for r5_cpa_pke: games G_4 and G_5

Game $\boldsymbol{G_4}$	**Game** $\boldsymbol{G_5}$
1. $\boldsymbol{A} \overset{\$}{\leftarrow} \mathcal{R}_{n,q}^{d/n \times d/n}$	1. $\boldsymbol{A} \overset{\$}{\leftarrow} \mathcal{R}_{n,q}^{d/n \times d/n}$
2. -	2. -
3. $\boldsymbol{B} \overset{\$}{\leftarrow} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$	3. $\boldsymbol{B} \overset{\$}{\leftarrow} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$
4. Choose $\beta \overset{\$}{\leftarrow} \{0,1\}$.	4. Choose $\beta \overset{\$}{\leftarrow} \{0,1\}$.
5. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$.	5. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \langle(\boldsymbol{B_q}\rangle_p)$.
6. $\rho \overset{\$}{\leftarrow} \{0,1\}^{\kappa}$; $\boldsymbol{R} = f_R(\rho)$	6. $\boldsymbol{R} \overset{\$}{\leftarrow} \chi_S^{\overline{m}}$
7. $\boldsymbol{U} = \mathrm{R}_{q \to p, h_2}(\langle \boldsymbol{A}^T \boldsymbol{R} \rangle_{\Phi_{n+1}})$	7. $\boldsymbol{U} = \mathrm{R}_{q \to p, h_2}(\langle \boldsymbol{A}^T \boldsymbol{R} \rangle_{\Phi_{n+1}})$
8. $\boldsymbol{V} = \langle \big(\mathrm{R}_{p \to t, h_2}(\langle \boldsymbol{B}^T \boldsymbol{R} \rangle_{\xi})\big) + \frac{t}{b} \boldsymbol{M}_{\beta} \rangle_t$	8. $\boldsymbol{V} = \langle \big(\mathrm{R}_{p \to t, h_2}(\langle \boldsymbol{B}^T \boldsymbol{R} \rangle_{\xi})\big) + \frac{t}{b} \boldsymbol{M}_{\beta} \rangle_t$
9. $\beta' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \mathrm{Sample}_{\mu}(\boldsymbol{V})), st\right)$	9. $\beta' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \mathrm{Sample}_{\mu}(\boldsymbol{V})), st\right)$
10. **return**$[(\beta' = \beta)]$.	10. **return**$[(\beta' = \beta)]$.

As p divides q and $\boldsymbol{B}_q \equiv \langle \boldsymbol{B_q} \rangle_p \pmod{p}$,

$$\boldsymbol{V''} \equiv \lfloor \frac{t}{p}(\langle {\boldsymbol{B_q}}^T \rangle_p \boldsymbol{R} \rangle_{\xi} + h_2 \boldsymbol{J}) \rfloor + \frac{t}{b} \boldsymbol{M}_{\beta} \pmod{t}.$$

As a result, the pairs $(\boldsymbol{B}, \boldsymbol{v})$ in Game G_5 and $(\langle \boldsymbol{B}_q \rangle_p, \langle \boldsymbol{v'} \rangle_t)$ in Game G_6 have the same distribution, and so

$$\Pr(S_5) = \Pr(S_6). \tag{46}$$

We now consider games G_6 and G_7. We will use the following lemma.

Lemma A.4.1. *Let a, b, c be positive integers such that $a|b|c$. Then for any $x \in \mathbb{R}$*

$$\lfloor \frac{a}{c} x \rfloor = \lfloor \frac{a}{b} \lfloor \frac{b}{c} x \rfloor \rfloor$$

Proof Write $m = \frac{c}{b}$ and $n = \frac{b}{a}$. It is sufficient that to show that

$$\lfloor \frac{x}{mn} \rfloor = \lfloor \frac{1}{n} \lfloor \frac{x}{m} \rfloor \rfloor.$$

We write $x = mn\lfloor \frac{x}{mn} \rfloor + y$ with $0 \leq y < mn$. Obviously, $\frac{1}{n} \lfloor \frac{x}{m} \rfloor = \lfloor \frac{x}{mn} \rfloor + \frac{1}{n} \lfloor \frac{y}{m} \rfloor$. As $0 \leq y < mn$, it holds that $0 \leq \frac{y}{m} < n$ and so $0 \leq \lfloor \frac{y}{m} \rfloor \leq n-1$. □

Applying the above lemma, we infer that $(\boldsymbol{U}, \boldsymbol{V'})$ in Game G_6 and $(\boldsymbol{U'}, \boldsymbol{V''})$ in Game G_7 are related as $\boldsymbol{U} = \lfloor \frac{p}{z} \boldsymbol{U'} \rfloor$ and $\boldsymbol{V'} \equiv \lfloor \frac{tq}{pz} \boldsymbol{V''} \rfloor \pmod{\frac{tq}{p}}$. As a result,

$$\Pr(S_6) = \Pr(S_7). \tag{47}$$

Table 31: IND-CPA games for r5_cpa_pke: games G_6, G_7 and G_8

Game G_6	**Game G_7**	**Game G_8**
1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$	1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$	1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$
2. -	2. -	2.-
3. $\boldsymbol{B_q} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$	3. $\boldsymbol{B_q} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times \overline{n}}$	3. $\boldsymbol{B_q} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$
4. Choose $\beta \xleftarrow{\$} \{0,1\}$.	4. Choose $\beta \xleftarrow{\$} \{0,1\}$.	4. Choose $\beta \xleftarrow{\$} \{0,1\}$.
5. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \langle \boldsymbol{B_q} \rangle_p)$.	5. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \langle (\boldsymbol{B_q} \rangle_p)$.	5. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \langle \boldsymbol{B_q} \rangle_p)$.
6. $\boldsymbol{R} \xleftarrow{\$} \chi_S^{\overline{m}}$	6. $\boldsymbol{R} \xleftarrow{\$} \chi_S^{\overline{m}}$	6. -
7. $\boldsymbol{U} = \mathrm{R}_{q \to p, h_2}(\langle \boldsymbol{A}^T \boldsymbol{R} \rangle_{\Phi_{n+1}})$	7. $\boldsymbol{U'} = \mathrm{R}_{q \to z, h_2}(\langle \boldsymbol{A}^T \boldsymbol{R} \rangle_{\Phi_{n+1}})$ with $z = \max(p, tq/p)$	7. $\boldsymbol{U"} \xleftarrow{\$} \mathcal{R}_{n,z}^{d/n \times \overline{m}}$
8. $\boldsymbol{V'} = \langle \big(\mathrm{R}_{q \to tq/p, h_2}(\langle {\boldsymbol{B_q}}^T \boldsymbol{R} \rangle_\xi)\big) + \frac{t}{b} \boldsymbol{M}_\beta \rangle_{tq/p}$	8. $\boldsymbol{V''} = \langle \big(\mathrm{R}_{q \to z, h_2}(\langle {\boldsymbol{B_q}}^T \boldsymbol{R} \rangle_\xi)\big) + \frac{pz}{qb} \boldsymbol{M}_\beta \rangle_z$	8. $\boldsymbol{W} \xleftarrow{\$} \mathcal{R}_{n,z}^{\overline{n} \times \overline{m}}$; $\boldsymbol{V'''} = \langle \boldsymbol{W} + \frac{pz}{qb} \boldsymbol{M}_\beta \rangle_z$
9. $\beta' = \mathcal{A}\left((\boldsymbol{A}, \langle \boldsymbol{B_q} \rangle_p), (\boldsymbol{U}, \mathrm{Sample}_\mu(\langle \boldsymbol{V'} \rangle_t), st\right)$	9. $\beta' = \mathcal{A}((\boldsymbol{A}, \langle \boldsymbol{B_q} \rangle_p), (\langle \lfloor \frac{p}{z} \boldsymbol{U'} \rceil \rangle_p,$ $\mathrm{Sample}_\mu(\langle \lfloor \frac{tq}{pz} \boldsymbol{V''} \rceil \rangle_t), st)$	9. $\beta' = \mathcal{A}((\boldsymbol{A}, \langle \boldsymbol{B_q} \rangle_p), (\langle \lfloor \frac{p}{z} \boldsymbol{U"} \rceil \rangle_p,$ $\mathrm{Sample}_\mu(\langle \lfloor \frac{tq}{pz} \boldsymbol{V'''} \rceil \rangle_t), st)$
10. **return**$[(\beta' = \beta)]$.	10. **return**$[(\beta' = \beta)]$.	10. **return**$[(\beta' = \beta)]$.

In Game G_8, the variable $\begin{bmatrix} \mathrm{R}_{q \to z, h_2}(\langle \boldsymbol{A^T R} \rangle_{\phi_{n+1}}) \\ \mathrm{R}_{q \to z, h_2}(\langle \boldsymbol{B^T R} \rangle_\xi) \end{bmatrix}$ from Game G_7 is replaced by the $(d/n + \overline{n}) \times \overline{m}$ matrix $\begin{bmatrix} \boldsymbol{U"} \\ \boldsymbol{W'} \end{bmatrix}$ with entries uniformly drawn from $\mathcal{R}_{n,z}$. As $h_2 = \frac{q}{2z}$, $\mathrm{R}_{q \to z, h_2} = \mathrm{R}_{q \to z}$. Moreover, if $\xi = \Phi_{n+1}$, the first variable in fact equals $\mathrm{R}_{q \to z}(\langle \begin{bmatrix} \boldsymbol{A}^T \\ \boldsymbol{B}^T \end{bmatrix} \boldsymbol{R} \rangle_\phi)$. We infer that there exists a distinghuisher $\mathcal{E}$ such that

$$\mathrm{Adv}_{d,n,d/n+\overline{n},q,z}^{\mathtt{dGLWR}_{\mathrm{spt}}}(\mathcal{A} \circ \mathcal{Z}) = \mid \Pr(S_7) - \Pr(S_8) \mid \tag{48}$$

As all inputs to $\mathcal{A}$ in Game G_8 are uniform, $\Pr(S_8) = \frac{1}{2}$, and so

$$\mathrm{Adv}_{\text{CPA-PKE}}^{\text{IND-CPA}}(\mathcal{A}) = \mid \Pr(S_0) - \Pr(S_8) \mid = \mid \sum_{i=0}^{7} \Pr(S_i) - \Pr(S_{i+1}) \mid \leq \sum_{i=0}^{7} |\Pr(S_i) - \Pr(S_{i+1})|. \tag{49}$$

By combining (40)-(49), the theorem follows.

A.5 IND-CPA Security for RLWE-based Round5 variant with different reduction polynomials

As described in Section 2.4.1, the merged parameters of Round5 require that the reduction polynomial $\xi(x)$ used in the computation of the ciphertext component $\boldsymbol{v}$ and in decryption equals $x^{n+1} - 1$. The proof of the IND-CPA security of r5_cpa_pke presented in Section A.4, however, only applies if $\xi(x) = \Phi_{n+1}(x)$.

The reason is that otherwise the replacement of $(\boldsymbol{U}', \boldsymbol{V}'')$ by random matrices in the transition from game G_7 to game G_8 cannot be directly related to the difficulty of GLWR with one single reduction polynomial.

Next we argue why this construction is secure: First, Round5 uses function $Sample_\mu$ that selects μ coefficients out of $n+1$. We show how it prevents known distinguishing attacks such as the "*Evaluate at 1*" attack [57]. Second, we discuss an extension of the IND-CPA security proof in Section A.4 for a RLWE-variant of Round5. The existence of this proof for the RLWE case gives strong confidence in the Round5 parameters using error correction. Finally, we discuss why this proof does not directly translate to an RLWR based design and a simple design change in Round5 that would make it work, but that we have not introduced since it does not bring major benefits from a concrete security viewpoint.

Distinguishing attack at $x = 1$. In this section, the well-known "Evaluate at $x = 1$" distinguishing attack [57] is described that can be applied if $\xi(x) = x^{n+1} - 1$ and $\mu = n+1$. Next, it is argued that this attack cannot be applied in Round5 if $\mu \leq n$. As a shorthand, $N(x)$ is written instead of $x^{n+1} - 1$.

Consider a pair of polynomials $(b(x), v(x))$ with $b(x)$ uniformly distributed on $\mathbb{Z}_p[x]/(x^{n+1}-1)$ and $v(x) = \langle \text{Sample}_\mu(\lfloor \frac{t}{p}(\langle b(x)r(x)\rangle_{N(x)} + h_2)\rfloor) + \frac{t}{b}m(x)\rangle_t$ with $r(x)$ drawn independently and uniformly from the ternary polynomials of degree at most $n-1$ satisfying $r(1) = 0$, and $m(x)$ drawn according to some distribution on $\mathbb{Z}_b[x]/(x^\mu - 1)$. We then have that $v(x) \equiv \lfloor \text{Sample}_\mu(\frac{t}{p}(\langle b(x)r(x)\rangle_{N(x)} + h_2)\rfloor) + \frac{t}{b}m(x) \pmod t$, and so $w(x) = \frac{p}{t}v(x)$ satisfies

$$w(x) \equiv \text{Sample}_\mu(\langle b(x)r(x)\rangle_{N(x)}) + \frac{p}{t}\cdot h_2 \sum_{i=0}^{\mu-1} x^i - \frac{p}{t}\epsilon(x) + \frac{p}{b}m(x) \pmod p.$$

where $\epsilon(x)$ is the result of rounding downwards, so all components of $\frac{p}{t}\epsilon(x)$ are in $[0, \frac{p}{t}) \cap \mathbb{Z}$. As $(x-1)$ divides both $r(x)$ and $N(x)$, it follows that $x-1$ divides $\langle b(x)r(x)\rangle_{N(x)}$, and so if $\mu = n+1$, then

$$w(1) \equiv \frac{p}{t}\cdot h_2 \cdot (n+1) - \frac{p}{t}\sum_{i=0}^{n} \epsilon_i + \frac{p}{b}m(1) \pmod p.$$

For large n, the value of $\frac{p}{t}\sum_{i=0}^{n}\epsilon_i$ is close to its average, i.e., close to $n\frac{p}{2t}$. As a result, has maxima at values $\frac{p}{t}h_2(n+1) - n\frac{p}{2t} + \frac{p}{b}k$ for $0 \leq k \leq b-1$. So $w(1)$ can serve as a distinguisher between the above distribution and the uniform one. Now assume that $\mu < n+1$. We take $\mu = n$, which is the case giving most information to the attacker. Writing $f(x) = \langle b(x)r(x)\rangle_{N(x)} = \sum_{i=0}^{n} f_i x^i$, it holds that

$$w(1) \equiv \sum_{i=0}^{n-1} f_i + \frac{p}{t}\cdot h_2 \cdot n - \frac{p}{t}\epsilon(1) + \frac{p}{b}m(1) \pmod p.$$

Algorithm 25: round_to_root(a, q, p)

```
$b \leftarrow \left\lfloor \frac{p}{q} a \right\rceil$
for $i \leftarrow 0$ to $n-1$ do
    $e_i \leftarrow \left(\texttt{idx} = i \in \mathbb{Z},\ \texttt{val} = \frac{p}{q} a - \left\lfloor \frac{p}{q} a \right\rceil \in \mathbb{Q}\right)$
end
Sort $e$ in descending order of $e.\texttt{val}$.
$k \leftarrow p \left\lceil \frac{b(1)}{p} \right\rceil - b(1)$
for $i \leftarrow 0$ to $k-1$ do
    $b_{e_i.\texttt{idx}} \leftarrow b_{e_i.\texttt{idx}} + 1$
end
return $b$
```

As shown above, $f(1) = 0$, and so $\sum_{i=0}^{n-1} f_i = -f_n$. Hence, under the assumption that f_n is distributed uniformly modulo p, also $w(1)$ is distributed uniformly modulo p. The latter assumption is supported by [82].

Requirements for IND-CPA Security – Design rationale. An IND-CPA security proof is feasible for a RLWE variant of r5_cpa_pke, i.e., a Round5 variant where the noise is independently generated. This proof is presented below and gives confidence in Round5's design and choices made.

The proof requires the secrets to be a multiple of $(x-1)$ and also the noise polynomial for the ciphertext component v to be a multiple of $(x-1)$ (this is used in an essential step of the proof, specifically in the map Ψ in (58)) This last requirement is the reason why this proof does not apply to Round5 with $\xi(x) = x^{n+1} - 1$ using RLWR defined as *component-wise rounding*. This deterministic component-wise rounding does not allow enforcing that the noisy "rounding" polynomials are multiples of $(x-1)$.

Round5's design can be adapted to use a slightly different type of rounding informally named as "rounding to the root lattice" [46, 47, 75] - that allows the IND-CPA proof to work. This alternate rounding technique is described in Algorithm 25, that takes as input an $a \in \mathbb{Z}_q[x]$, integer moduli q, p where $p < q$ and returns a $b \in \mathbb{Z}_p[x]$ satisfying $b(1) \equiv 0 \pmod{p}$.

Rounded noise introduced in b using Algorithm 25 is a polynomial whose coefficients sum to zero, so that the IND-CPA proof given below can be directly translated to the RLWR case. However, this modification – going from component-wise rounding to rounding to the root lattice – would introduce additional complexity with no clear concrete security benefits. First, Sample$_\mu$ gets rid of $n + 1 - \mu$ coefficients so that knowing k is irrelevant. Second, concrete security attacks use the norm of the noise that hardly changes here. Because of these two reasons, we argue that the current Round5 design (and the rounding used in it) is sound and secure, and further modifications are not required.

IND-CPA Proof for RLWE-based variant of Round5. We now present the proof of IND-CPA security for the RLWE variant of r5_cpa_pke. We restrict ourselves to the case $B = 1$, that is, one single bit is extracted from each symbol. Our proof uses elements from [27, Sec E1].

The following notation will be used. We write $\phi(x) = 1 + x + \ldots + x^n$, and $N(x) = x^{n+1} - 1$, where $n + 1$ is prime. Moreover, $R_\phi = \mathbb{Z}_q[x]/\phi(x)$, and

$$R_0 = \{f(x) = \sum_{i=0}^{n} f_i x^i \in \mathbb{Z}_q[x] \mid \sum_{i=0}^{n} f_i \equiv 0 \pmod q\} \tag{50}$$

As $N(x) = (x-1)\phi(x)$, it holds that $\langle (x-1)f(x)\rangle_{N(x)} = (x-1)\langle f(x)\rangle_{\phi(x)}$ for any $f \in \mathbb{Z}[x]$. As a result, $f(x) \mapsto (x-1)f(x)$ is a bijection from R_ϕ to R_0.

In the proof, the following lemma will be used.

Lemma A.5.1. *Let q and $n+1$ be relatively prime, and let $(n+1)^{-1}$ be the multiplicative inverse of $n+1$ in $\mathbb{Z}_q$. The mapping $\mathcal{F}$ defined as*

$$\mathcal{F} : (\sum_{i=0}^{n-1} f_i x^i) \mapsto \sum_{i=0}^{n-1} f_i x^i - (n+1)^{-1} \cdot \left(\sum_{i=0}^{n-1} f_i\right) \cdot \phi(x)$$

is a bijection from R_ϕ to R_0.

Proof. It is easy to see that $\mathcal{F}$ maps R_ϕ to R_0. To show that $\mathcal{F}$ is a bijection, let $g(x) = \sum_{i=0}^{n} g_i x^i \in R_0$, and let $f(x) = \sum_{i=0}^{n} \langle g_i - g_n\rangle_q x^i$. Clearly, $f \in \mathbb{Z}_q[x]$ has degree at most $n-1$, and by direct computation, $\mathcal{F}(f(x)) = g(x)$. □

In the description below, $\mathcal{S}$ denotes a set of secrets such that

$$\mathcal{S} \subset \{f(x) = \sum_{i=0}^{n-1} f_i x^i \in \mathbb{Z}_q[x] \mid \sum_{i=0}^{n-1} f_i \equiv 0 \pmod q\}, \tag{51}$$

Moreover, $\mathcal{M}$ denotes a message space, and *ECC_Enc* and *ECC_Dec* are error correcting encoding and decoding algorithms such that

$$\{ECC\_Enc(m) \mid m \in \mathcal{M}\} \subset \{f(x) = \sum_{i=0}^{n} f_i x^i \in \mathbb{Z}_2[x] \mid \sum_{i=0}^{n} f_i \equiv 0 \pmod 2\}. \tag{52}$$

Moreover, χ denotes a probability distribution on R_ϕ.

For understanding Algorithm 28 below, note that as $(x-1)|s(x)$, we have that $su' \equiv sa'r + se_1 \pmod N$, and, as $(x-1)|r(x)$, that $rb' \equiv ra's + re_0 \pmod N$. As a consequence,

$$\zeta \equiv v - su' \equiv \frac{q}{2}ECC\_Enc(m) + (x-1)e_2 + re_0 - se_1 \pmod N, \text{ whence}$$

$$\lfloor \frac{2}{q}\zeta \rceil \equiv ECC\_Enc(m) + \lfloor \frac{2}{q}((x-1)e_2 + re_0 - se_1)\rceil \pmod N.$$

We are now in a position to prove the following result.

Algorithm 26: CPA-PKE.Keygen()

```
$a' \xleftarrow{\$} R_\phi, s \xleftarrow{\$} \mathcal{S}, e_0 \leftarrow \chi$
$b' = \langle a's + e_0 \rangle_\phi$
$pk = (a', b')$
$sk = s$
return $(pk, sk)$
```

Algorithm 27: CPA-PKE.Enc($pk = (a', b'), m \in \mathcal{M}$)

```
$r \xleftarrow{\$} \mathcal{S}, e_1, e_2 \xleftarrow{\$} \chi$
$u' = \langle a'r + e_1 \rangle_\phi$
$v = \langle \frac{q}{2} ECC\_Enc(m) + b'r + (x-1)e_2 \rangle_N$
$ct = (u', v)$
return $ct$
```

Theorem A.5.1. *For every IND-CPA adversary $\mathcal{A}$ with advantage A, there exist algorithms C and E such that*

$$A \leq Adv_1(C) + Adv_3(E). \tag{53}$$

Here Adv_1 refers to the advantage of distinguishing between the uniform distribution on $(\mathbb{Z}_q[x]/\phi(x))^2$ and the R-LWE distribution

$$(a', b' = \langle a's + e_0 \rangle_\phi) \textit{ with } a' \xleftarrow{\$} R_\phi, s \xleftarrow{\$} \mathcal{S}, e_0 \leftarrow \chi \tag{54}$$

Similarly, Adv_3 refers to the advantage of distinguishing between the uniform distribution on $(\mathbb{Z}_q[x]/\phi(x))^4$ and the distribution of two R-LWE samples with a common secret, given by

$$(a', b'', u', v') \textit{ with } a', b'' \xleftarrow{\$} \mathbb{Z}_q[x]/\phi(x), u = \langle a'r + e_1 \rangle_\phi, \tag{55}$$

$$v = \langle b"r + e_2 \rangle_\phi \textit{ with } r \xleftarrow{\$} \mathcal{S}, e_1, e_2 \leftarrow \chi \tag{56}$$

Proof. We prove the theorem using a sequence of IND-CPA games. We denote by S_i the event that the output of game i equals 1.
Game G_0 is the original IND-CPA game. In Game G_1, the public key (a', b') is replaced by a pair (a', b') uniformly drawn from R_ϕ^2. It can be shown that there exists an algorithm $\mathcal{C}$ for distinguishing between the uniform distribution on R_ϕ^2 and the R-LWE distribution of pairs (a', b') with $a' \xleftarrow{\$} R_\phi$, $b' = \langle as' + e_0 \rangle_\phi$ with $s \xleftarrow{\$} \mathcal{S}$ and $e_0 \leftarrow \chi$ such that

$$\mathrm{Adv}_1(\mathcal{C}) = |\Pr(S_0) - \Pr(S_1)|.$$

In Game G_2, the values $u' = \langle a'r + e_1 \rangle_\phi$ and $\hat{v} = \langle b'r + (x-1)e_2 \rangle_N$ used in the generation of v are simultaneously substituted with uniform random variables

Algorithm 28: CPA-PKE.Dec(sk, ct)

```
$\zeta = \langle v - su' \rangle_N$
$\hat{m} = ECC\_Dec \langle \lfloor \frac{2\zeta}{q} \rceil \rangle_2)$
return $\hat{m}$
```

from R_ϕ and R_0, respectively. it can be shown that there exists an adversary $\mathcal{D}$ with the same running time as that of $\mathcal{A}$ such that

$$\mathrm{Adv}_2(\mathcal{D}) = |\Pr(S_1) - \Pr(S_2)|.$$

Here Adv_2 refers to the advantage of distinguishing between the uniform distribution on $R_\phi^3 \times R_0$ and the distribution

$$(a', b', u', v) = (a', b', \langle a'r + e_1 \rangle_\phi, \langle b'r + (x-1)e_2 \rangle_N) \text{ with } a', b' \xleftarrow{\$} R_\phi, r \xleftarrow{\$} \mathcal{S}, e_1, e_2 \xleftarrow{\$} \chi. \tag{57}$$

Because of (52), the value of the ciphertext v in Game G_2 is independent of bit b, and therefore $\Pr(S_2) = 1/2$. As a final step, we define $\Psi : R_\phi^3 \times R_0 \rightarrow R_\phi^4$ as

$$\Psi(a'(x), b'(x), u'(x), v(x)) = (a'(x), b''(x), u'(x), v'(x)) \text{ with} \tag{58}$$

$$b''(x) = \frac{\mathcal{F}(b'(x))}{x-1}, v'(x) = \frac{v(x)}{x-1} \tag{59}$$

As $\mathcal{F}$ is a bijection from R_ϕ to R_0 (see Lemma A.5.1) and $f(x) \mapsto \frac{f(x)}{x-1}$ is a bijection from R_0 to R_ϕ, it follows that Ψ is a bijection. Writing $b(x) = \mathcal{F}(b'(x))$, we infer that

$$b(x)r(x) = b'(x)r(x) - (n+1)^{-1} b'(1)\phi(x)r(x) \equiv b'(x)r(x) \pmod{N(x)},$$

where the latter equivalence holds as $r(x)$ is a multiple of $(x-1)$, and so

$$v(x) = \langle b'(x)r(x) + (x-1)e_2(x) \rangle_N = \langle b(x)r(x) + (x-1)e_2(x) \rangle_N.$$

As $r(x)$ is a multiple of $x-1$, it follows that $v(x) \in R_0$ and that

$$v'(x) = \frac{v(x)}{x-1} \equiv \langle b''(x)r(x) + e_2(x) \rangle_\phi \text{ where } b''(x) = \frac{b(x)}{x-1}.$$

As a result, the advantage of $\mathcal{E} = \Psi \circ \mathcal{D}$ in distinguishing between the uniform distribution on R_Φ^4 and the distribution

$$(a', b'', u', v') \text{ with } a, b'' \xleftarrow{\$} R_\phi, u'(x) = \langle a'r + e_1 \rangle_\phi \text{ and } v' = \langle b''r + e_2 \rangle_\phi$$

is equal to $\mathrm{Adv}_2(D)$. Note that (a, u') and $(b", v')$ are two R-LWE samples with common secret $r(x) \in \mathcal{S}$, with $a', b"$ chosen uniformly in $\mathcal{R}_\phi$ and independent noise polynomials $e_1(x)$ and $e_2(x)$.
As $Pr(S_2) = \frac{1}{2}$, we conclude that

$$Adv(\mathcal{A}) = |\Pr(S_0) - \Pr(S_2)| \leq \sum_{i=0}^{1} |\Pr(S_i) - \Pr(S_{i+1})| = \mathrm{Adv}_1(\mathcal{C}) + \mathrm{Adv}_2(\mathcal{E}).$$

□

A.6 IND-CPA Security of r5_cpa_kem

This section presents a proof that r5_cpa_kem is IND-CPA secure, based on the hardness of the decision GLWR problem with sparse-ternary secrets. In the proof, instead of using $pk = (\sigma, \boldsymbol{B})$, we use $pk = (\boldsymbol{A} = f_{d,n}^{(\tau)}(\sigma), \boldsymbol{B})$. The distribution of $f_{d,n}^{(\tau)}(\sigma)$ with $\sigma \xleftarrow{\$} \{0,1\}^\kappa$ is denoted by F_τ. Moreover, the uniform distribution $\mathcal{H}_{n,d/n}(h)$ is denoted by χ_S. The proof is restricted to the case $\xi(x) = \Phi_{n+1}(x)$.

Theorem A.6.1. *Let b, p, q, t be powers of two such that $b|t|p|q$, and let $z = \max(p, tq/p)$. Furthermore, assume that $\xi(x) = \Phi_{n+1}(x)$. If $f_{d,n}^{(\tau)}$, f_S and f_R induce distributions indistinguishable from uniform, and H is a secure pseudorandom function, then r5_cpa_kem is IND-CPA secure under the hardness assumption of the Decision-GLWR problem with sparse-ternary secrets. More precisely, if $Adv_{r5\_cpa\_kem}^{IND\text{-}CPA}(\mathcal{A})$ is the advantage of adversary $\mathcal{A}$ in distinguishing a key encapsulated using r5_cpa_kem from random, then there exist distinghuishers $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}, \mathcal{G}$ such that*

$$\begin{aligned} Adv_{r5\_cpa\_kem}^{IND\text{-}CPA}(\mathcal{A}) \leq\ & Adv_{\mathcal{U}(\mathcal{R}_{n,q}^{d/n\times d/n}),F_\tau}(\mathcal{B}) + Adv_{G_S,\chi_S^{\overline{n}}}(\mathcal{C}) + Adv_{G_R,\chi_S^{\overline{m}}}(\mathcal{D}) \\ & + \overline{n}\cdot Adv_{d,n,d/n,q,p}^{dGLWR_{spt}}(\mathcal{E}) + Adv_{d,n,d/n+\overline{n},q,z}^{dGLWR_{spt}}(\mathcal{F}) + Adv_{G_H,\mathcal{U}(\{0,1\}^\kappa)}(\mathcal{G}) \end{aligned} \tag{60}$$

In this equation, F_τ is the distribution of $f_{d,n}^{(\tau)}$ with $\sigma \xleftarrow{\$} \{0,1\}^\kappa$, G_S is the distribution of $f_S(s)$ with $s \xleftarrow{\$} \{0,1\}^\kappa$ and G_R is the distribution of $f_R(\rho)$ with $\rho \xleftarrow{\$} \{0,1\}^\kappa$. Moreover, $Adv_{d,n,m,q_1,q_2}^{dGLWR_{spt}}(\mathcal{Z})$ is the advantage of adversary $\mathcal{Z}$ in distinguishing m GLWR samples (with sparse-ternary secrets) from uniform, with the GLWR problem defined for the parameters d, n, q_1, q_2. Finally, G_H is the distribution of $H(x)$ with $x \xleftarrow{\$} \{0,1\}^{\kappa+\lambda_1}$ with $\lambda_1 = d\cdot\overline{n}\log_2(p) + \mu\log_2(t)$.

Proof. The proof for Theorem A.6.1, proceeds via a similar sequence of games as in the proof of Theorem A.4.1, shown in Tables 32 to 35. Again, S_i denotes the event that the output in game G_i equals 1. As the sequence of games is essentially the same as for the proof of Theorem A.4.1, we focus on the essential difference, which is game G_8. Game G_8 differs from game G_7 only in the generation of k_0. As p, q, t all are powers of two and $z = \max(\frac{tq}{p}, p)$, p divides z and tq divides pz. As a result, in games G_7 and G_8, $\lfloor \frac{p}{z}\boldsymbol{U}'' \rceil$ is uniformly distributed on $\mathcal{R}_{n,p}^{d/n\times\overline{m}}$. Also, as $\boldsymbol{W}'''$ in games G_7 and G_8 is uniformly distributed on $\mathcal{R}_{n,z}^{\overline{n}\times\overline{m}}$, the vector $\lfloor \frac{tq}{pz}\boldsymbol{v}''' \rceil$ is uniformly distributed on $\mathbb{Z}_{\frac{tq}{p}}$. As t divides $\frac{tq}{p}$, the vector $\langle \lfloor \frac{tq}{pz}\boldsymbol{v}''' \rceil \rangle_t$ is uniformly distributed on $\mathbb{Z}_t$. We conclude that the input to H in game G_7 is uniform. Therefore, there exists a distinguisher between the uniform distribution on $\{0,1\}^\kappa$ and the distribution of $H(x)$ with $x \xleftarrow{\$} \{0,1\}^{\kappa+\lambda_1}$ (where $\lambda_1 = d\cdot\overline{n}\log_2(p) + \mu\log_2(t)$) with advantage equal to $|\Pr(S_7) - \Pr(S_8)|$. As the input to $\mathcal{A}$ in game G_8 is uniform, $\Pr(S_8) = \frac{1}{2}$. □

Table 32: IND-CPA games for r5_cpa_kem: Games G_0 and G_1

Game $\boldsymbol{G_0}$	**Game $\boldsymbol{G_1}$**
1. $\boldsymbol{A} \leftarrow F_\tau$	1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$
2. $s \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{S} = f_S(s)$	2. $s \xleftarrow{\$} \{0,1\}^\kappa$; $\boldsymbol{S} = f_S(s)$;
3. $\boldsymbol{B} = \mathrm{R}_{q\to p}(\langle \boldsymbol{AS} \rangle_{\Phi_{n+1}})$	3. $\boldsymbol{B} = \mathrm{R}_{q\to p}(\langle \boldsymbol{AS} \rangle_{\Phi_{n+1}})$
4. Choose $b \xleftarrow{\$} \{0,1\}$.	4. Choose $b \xleftarrow{\$} \{0,1\}$.
5. $(ct = (\boldsymbol{U}, \boldsymbol{v}), k_0) =$ r5_cpa_kem_encapsulate $(\boldsymbol{A}, \boldsymbol{B})$.	5. $(ct = (\boldsymbol{U}, \boldsymbol{v}), k_0) =$ r5_cpa_kem_encapsulate $(\boldsymbol{A}, \boldsymbol{B})$.
6. $k_1 \xleftarrow{\$} \{0,1\}^\kappa$.	6. $k_1 \xleftarrow{\$} \{0,1\}^\kappa$.
7. $b' = \mathcal{A}((\boldsymbol{A}, \boldsymbol{B}), ct, k_b)$.	7. $b' = \mathcal{A}((\boldsymbol{A}, \boldsymbol{B}), ct, k_b)$.
8. Output $[(b' = b)]$.	8. Output $[(b' = b)]$.

Table 33: IND-CPA games for r5_cpa_kem: Games G_2 and G_3

Game $\boldsymbol{G_2}$	**Game $\boldsymbol{G_3}$**
1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$	1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$
2. $\boldsymbol{S} \leftarrow \chi_S^{\overline{n}}$	2. -
3. $\boldsymbol{B} = \mathrm{R}_{q\to p}(\langle \boldsymbol{AS} \rangle_{\Phi_{n+1}})$	3. $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$
4. Choose $b \xleftarrow{\$} \{0,1\}$.	4. Choose $b \xleftarrow{\$} \{0,1\}$.
5. $(ct = (\boldsymbol{U}, \boldsymbol{v}), k_0) =$ r5_cpa_kem_encapsulate $(\boldsymbol{A}, \boldsymbol{B})$.	5. $(ct = (\boldsymbol{U}, \boldsymbol{v}), k_0) =$ r5_cpa_kem_encapsulate $(\boldsymbol{A}, \boldsymbol{B})$.
6. $k_1 \xleftarrow{\$} \{0,1\}^\kappa$.	6. $k_1 \xleftarrow{\$} \{0,1\}^\kappa$.
7. $b' = \mathcal{A}((\boldsymbol{A}, \boldsymbol{B}), ct, k_b)$.	7. $b' = \mathcal{A}((\boldsymbol{A}, \boldsymbol{B}), ct, k_b)$.
8. Output $[(b' = b)]$.	8. Output $[(b' = b)]$.

A.7 IND-CCA security of r5_cca_pke

In this section, it is shown that r5_cca_pke is IND-CCA secure. As r5_cca_pke is constructed from r5_cca_kem and a secure data-encapsulation mechanism as proposed by Cramer and Shoup [42], it is sufficient to show the IND-CCA security of r5_cca_kem. Indeed, as stated in Theorem A.7.1, when the hash functions G and H in Algorithms 8 and 9 are modeled as random oracles, the key-encapsulation mechanism r5_cca_kem defined in Section 2.4.5 is IND-CCA secure, assuming the hardness of the decision GLWR problem with sparse-ternary secrets.

Theorem A.7.1. *For any adversary $\mathcal{A}$ that makes at most q_H queries to the random oracle H, at most q_G queries to the random oracle G, and at most q_D queries to the decryption oracle, there exists an adversary $\mathcal{B}$ such that*

$$Adv_{\textit{CCA-KEM}}^{\textit{IND-CCA}}(\mathcal{A}) \leq 3 \cdot Adv_{\textit{CPA-PKE}}^{\textit{IND-CPA}}(\mathcal{B}) + q_G \cdot \delta + \frac{2q_G + q_H + 1}{2^{\mu B}} \tag{61}$$

Table 34: IND-CPA games for r5_cpa_kem: Games G_4 and G_5

Game G_4	**Game G_5**
1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n\times d/n}$, $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n\times\overline{n}}$.	1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n\times d/n}$, $\boldsymbol{B}_q \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n\times\overline{n}}$.
2. Choose $b \xleftarrow{\$} \{0,1\}$.	2. Choose $b \xleftarrow{\$} \{0,1\}$.
3. $m \xleftarrow{\$} \{0,1\}^{\kappa}, \boldsymbol{\zeta} = ECC\_Enc_{\kappa,f}(m)$	3. $m \xleftarrow{\$} \{0,1\}^{\kappa}, \boldsymbol{\zeta} = ECC\_Enc_{\kappa,f}(m)$
4. $\boldsymbol{R} \xleftarrow{\$} \left(\mathcal{H}_{n,d/n}(h)\right)^{1\times\overline{m}}$.	4. $\boldsymbol{R} \xleftarrow{\$} \left(\mathcal{H}_{n,d/n}(h)\right)^{1\times\overline{m}}$
5. $\boldsymbol{U} = \mathrm{R}_{q\to p,h_2}(\langle \boldsymbol{A}^T\boldsymbol{R}\rangle_{\Phi_{n+1}})$	5. $\boldsymbol{U} = \mathrm{R}_{q\to p,h_2}(\langle \boldsymbol{A}^T\boldsymbol{R}\rangle_{\Phi_{n+1}})$
6. $\boldsymbol{W} = \mathrm{R}_{p\to t,h_2}(\langle \boldsymbol{B}^T\boldsymbol{R}\rangle_{\xi})$	6. $\boldsymbol{W'} = \mathrm{R}_{q\to tq/p,h_2}(\langle \boldsymbol{B}_q^T\boldsymbol{R}\rangle_{\xi})$
7. $\boldsymbol{v} = \langle \mathrm{Sample}_{\mu}(\boldsymbol{W}) + \frac{t}{b}\boldsymbol{\zeta}\rangle_t$	7. $\boldsymbol{v'} = \langle \mathrm{Sample}_{\mu}(\boldsymbol{W'}) + \frac{t}{b}\boldsymbol{\zeta}\rangle_{tq/p}$
8. $k_0 = H\left(m\|\texttt{bin}_1\left(ct = (\boldsymbol{U}, \boldsymbol{v})\right)\right)$.	8. $k_0 = H\left(m\|\texttt{bin}_1\left(ct = (\boldsymbol{U}, \langle\boldsymbol{v'}\rangle_t)\right)\right)$.
9. $k_1 \xleftarrow{\$} \{0,1\}^{\kappa}$.	9. $k_1 \xleftarrow{\$} \{0,1\}^{\kappa}$.
10. $b' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), ct, k_b\right)$.	10. $b' = \mathcal{A}\left((\boldsymbol{A}, \langle\boldsymbol{B}_q\rangle_p), ct, k_b\right)$.
11. Output $[(b' = b)]$.	11. Output $[(b' = b)]$.

when r5_cpa_pke and r5_cca_kem both have a probability of decryption/decapsulation failure that is at most δ.

Proof. The proof of Theorem A.7.1 proceeds via two transformation reductions due to [58]. First, Lemma A.7.1 establishes that the OW-PCA [1] security of the deterministic public-key encryption scheme PKE$_1$ obtained from the public-key encryption scheme PKE via transformation T [58], tightly reduces to IND-CPA security of PKE$_1$. This lemma is a special case of [58, Theorem 3.2] with $q_v = 0$, since by definition OW-PCA security is OW-PCVA[2] security where the attacker is not allowed to query the ciphertext validity checking oracle.

Lemma A.7.1 (Adapted from [58, Theorem 3.2])**.** *Assume PKE to be δ correct. Then, for any OW-PCA adversary $\mathcal{B}$ that issues at most q_G queries to the random oracle G, q_P queries to a plaintext checking oracle P_{CO}, there exists an IND-CPA adversary $\mathcal{C}$ such that*

$$Adv_{PKE_1}^{OW\text{-}PCA}(\mathcal{B}) \leq q_G \cdot \delta + \frac{2q_G + 1}{|\mathcal{M}|} + 3 \cdot Adv_{PKE}^{IND\text{-}CPA}(\mathcal{C}) \qquad (62)$$

where $\mathcal{M}$ is the message/plaintext space of the public-key encryption schemes PKE and PKE$_1$.

Next, combination of Lemma A.7.1 and the reduction in [58, Theorem 3.4] shows that the IND-CCA security of a KEM with implicit rejection that is constructed using a non-deterministic PKE (like r5_cca_kem), tightly reduces to the IND-CPA security of said PKE. □

[1] The security notion of One-Way against Plaintext Checking Attacks.

[2] The security notion of OW-PCA, with access to a ciphertext Validity checking oracle.

Table 35: IND-CPA games for r5_cpa_kem: Games G_6, G_7 and G_8

Game $\boldsymbol{G_6}$	Game $\boldsymbol{G_7}$	Game $\boldsymbol{G_8}$
1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $\boldsymbol{B}_q \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times \overline{n}}$.	1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $\boldsymbol{B_q} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times \overline{n}}$.	1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $\boldsymbol{B}_q \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times \overline{n}}$.
2. Choose $b \xleftarrow{\$} \{0,1\}$.	2. Choose $b \xleftarrow{\$} \{0,1\}$.	2. Choose $b \xleftarrow{\$} \{0,1\}$.
3. $m \xleftarrow{\$} \{0,1\}^{\kappa}, \boldsymbol{\zeta} = ECC\_Enc_{\kappa,f}(m)$	3. $m \xleftarrow{\$} \{0,1\}^{\kappa}, \boldsymbol{\zeta} = ECC\_Enc_{\kappa,f}(m)$	3. $m \xleftarrow{\$} \{0,1\}^{\kappa}, \boldsymbol{\zeta} = ECC\_Enc_{\kappa,f}(m)$
4. $\boldsymbol{R} \xleftarrow{\$} \left(\mathcal{H}_{n,d/n}(h)\right)^{1 \times \overline{m}}$.	4. -	4. -
5. $\boldsymbol{U'} = \mathrm{R}_{q \to z,h_2}(\langle \boldsymbol{A}^T \boldsymbol{R} \rangle_{\Phi_{n+1}})$, $z = \max(p, tq/p)$	5. $\boldsymbol{U"} \xleftarrow{\$} \mathcal{R}_{n,z}^{d/n \times \overline{m}}$	5. $\boldsymbol{U"} \xleftarrow{\$} \mathcal{R}_{n,z}^{d/n \times \overline{m}}$
6. $\boldsymbol{W}" = \mathrm{R}_{q \to z,h_2}(\langle \boldsymbol{B}_q^T \boldsymbol{R} \rangle_{\xi})$	6. $\boldsymbol{W}''' \xleftarrow{\$} R_{n,z}^{\overline{n} \times \overline{m}}$	6. $\boldsymbol{W}''' \xleftarrow{\$} R_{n,z}^{\overline{n} \times \overline{m}}$
7. $\boldsymbol{v}" = \langle \mathrm{Sample}_{\mu}(\boldsymbol{W}") + \frac{pz}{qb}\boldsymbol{\zeta} \rangle_z$	7. $\boldsymbol{v}''' = \langle \mathrm{Sample}_{\mu}(\boldsymbol{W}''') + \frac{pz}{qb}\boldsymbol{\zeta} \rangle_z$	7. $\boldsymbol{v}''' = \langle \mathrm{Sample}_{\mu}(\boldsymbol{W}''') + \frac{pz}{qb}\boldsymbol{\zeta} \rangle_z$
8. $k_0 = H\left(m \Vert \mathtt{bin}_1\left(ct = \left(\lfloor \frac{p}{z}\boldsymbol{U}' \rceil, \langle \lfloor \frac{tq}{pz}\boldsymbol{v}" \rceil \rangle_t\right)\right)\right)$.	8. $k_0 = H\left(m \Vert \mathtt{bin}_1\left(ct = \left(\lfloor \frac{p}{z}\boldsymbol{U}" \rceil, \langle \lfloor \frac{tq}{pz}\boldsymbol{v}''' \rceil \rangle_t\right)\right)\right)$.	8. $k_0 \xleftarrow{\$} \{0,1\}^{\kappa}$
9. $k_1 \xleftarrow{\$} \{0,1\}^{\kappa}$.	9. $k_1 \xleftarrow{\$} \{0,1\}^{\kappa}$.	9. $k_1 \xleftarrow{\$} \{0,1\}^{\kappa}$.
10. $b' = \mathcal{A}(\left((\boldsymbol{A}, \langle \boldsymbol{B}_q \rangle_p), (\lfloor \frac{p}{z}\boldsymbol{U'} \rceil, \langle \lfloor \frac{tq}{pz}\boldsymbol{v}" \rceil \rangle_t), k_b\right)$.	10. $b' = \mathcal{A}(\left((\boldsymbol{A}, \langle \boldsymbol{B}_q \rangle_p), (\lfloor \frac{p}{z}\boldsymbol{U"} \rceil, \langle \lfloor \frac{tq}{pz}\boldsymbol{v}''' \rceil \rangle_t), k_b\right)$.	10. $b' = \mathcal{A}(\left((\boldsymbol{A}, \langle \boldsymbol{B}_q \rangle_p), (\lfloor \frac{p}{z}\boldsymbol{U"} \rceil, \langle \lfloor \frac{tq}{pz}\boldsymbol{v}''' \rceil \rangle_t), k_b\right)$.
11. Output $[b' = b)]$.	11. Output $[(b' = b)]$.	10. Output $[(b' = b)]$.

Direct application of [58, Theorem 4.6], similarly as in [30, Theorem 4.2], shows that r5_cca_kem is IND-CCA secure in the quantum random oracle model. The resulting security bound however is not tight.

Theorem A.7.2. *For any quantum adversary $\mathcal{A}$ that makes at most q_H queries to the quantum random oracle H, at most q_G queries to the quantum random oracle G, and at most q_D (classical) queries to the decapsulation oracle, there exists a quantum adversary $\mathcal{B}$ such that*

$$Adv_{CCA\text{-}KEM}^{IND\text{-}CCA}(\mathcal{A}) \leq 4q_H \sqrt{q_D \cdot q_H \cdot \delta + q_G \cdot \sqrt{Adv_{CPA\text{-}PKE}^{IND\text{-}CPA}(\mathcal{B})}} \tag{63}$$

A.8 Hardness of Sparse-Ternary LWR

In this section, we prove the hardness of the Decision-LWR problem with sparse-ternary secrets assuming that the small modulus p divides the large modulus q. Figure 4 provides an overview of the reductions involved in the proof of the main result, Theorem A.8.1.

Theorem A.8.1. *Let $k, p, q \geq 1$ and $m \geq n \geq h \geq 1$ be integers such that p divides q, and $k \geq m' = \frac{q}{p} \cdot m$. Let $\epsilon \in (0, \frac{1}{2})$, and $\alpha, \delta > 0$ such that*

$$\alpha \geq q^{-1}\sqrt{(2/\pi)\ln(2n(1+\epsilon^{-1}))},\ \binom{n}{h}2^h \geq q^{k+1} \cdot \delta^{-2},\ \textit{and } m = O(\frac{\log n}{\alpha\sqrt{10h}})$$

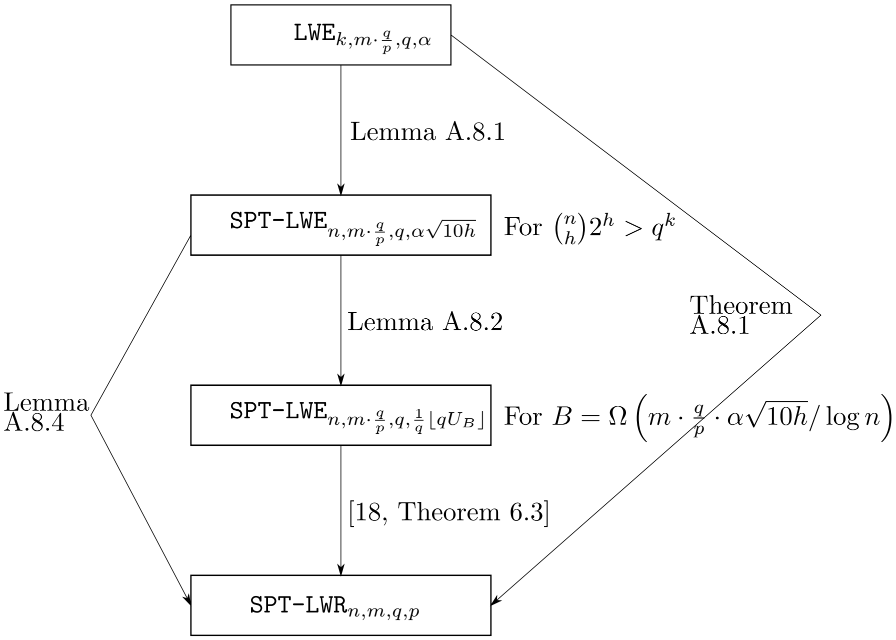
$\texttt{LWE}_{k,m\cdot\frac{q}{p},q,\alpha}$
Lemma A.8.1
$\texttt{SPT-LWE}_{n,m\cdot\frac{q}{p},q,\alpha\sqrt{10h}}$
For $\binom{n}{h}2^h > q^k$
Theorem A.8.1
Lemma A.8.2
Lemma A.8.4
$\texttt{SPT-LWE}_{n,m\cdot\frac{q}{p},q,\frac{1}{q}\lfloor qU_B\rfloor}$
For $B = \Omega\left(m\cdot\frac{q}{p}\cdot\alpha\sqrt{10h}/\log n\right)$
[18, Theorem 6.3]
$\texttt{SPT-LWR}_{n,m,q,p}$

Figure 4: Summary of reductions used in Theorem A.8.1. "SPT" refers to a variant of the problem in question where the secret is sparse-ternary, instead of uniform in $\mathbb{Z}_q^d$.

There exist three (transformation) reductions from $\texttt{dLWE}_{k,m',q,D_\alpha}$ *to* $\texttt{dLWE}_{n,m',q,D_{\alpha\sqrt{10h}}}(\mathcal{U}(\mathcal{H}_n(h)))$ *such that for any algorithm for the latter problem with advantage* ζ*, at least one of the reductions produces an algorithm for the former with advantage at least*

$$(\zeta-\delta)/(3m') - 41\epsilon/2 - \sum_{s|q,s \text{ prime}} s^{-k-1}.$$

Moreover, there is a reduction from $\texttt{dLWE}_{n,m',q,D_{\alpha\sqrt{10h}}}(\mathcal{U}(\mathcal{H}_n(h)))$ *to* $\texttt{dLWR}_{n,m,q,p}(\mathcal{U}(\mathcal{H}_n(h)))$*.*

Proof. Combination of Lemma A.8.1 and Lemma A.8.4 with $\alpha' = \alpha\sqrt{10h}$. □

Theorem A.8.1 implies the hardness of the sparse-ternary LWR problem $\texttt{LWR}_{\text{spt}}$ based on the hardness of the $\texttt{LWE}$ problem with uniformly random secrets in $\mathbb{Z}_q$ and Gaussian errors.

Step 1: Reduction from LWE with secrets in $\mathbb{Z}_q$ and Gaussian errors to Sparse-ternary LWE: In [37, Theorem 1], specializing [36, Theorem 4], it is shown that if $\binom{n}{h}2^h > q^{k+1}$ and $\omega > \alpha\sqrt{10h}$, then the $\texttt{dLWE}_{n,m,q,D_\omega}(\mathcal{U}(\mathcal{H}_n(h)))$ problem is at least as hard as the $\texttt{dLWE}_{k,m,q,D_\alpha}$ problem. More formally, generalizing [34, Theorem 4.1], the following holds.

Lemma A.8.1. *Let $k, q \geq 1$ and $m \geq n \geq h \geq 1$ be integers, and let $\epsilon \in (0, \frac{1}{2})$, and $\alpha, \delta > 0$ such that*

$$\alpha \geq q^{-1}\sqrt{(2/\pi)\ln(2n(1+\epsilon^{-1}))}, \text{ and } \binom{n}{h}2^h \geq q^{k+1} \cdot \delta^{-2}$$

There exist three (transformation) reductions from $\mathtt{dLWE}_{k,m,q,D_\alpha}$ to $\mathtt{dLWE}_{n,m,q,D_{\alpha\sqrt{10h}}}(\mathcal{U}(\mathcal{H}_n(h)))$ such that for any algorithm for the latter problem with advantage ζ, at least one of the reductions produces an algorithm for the former with advantage at least

$$(\zeta - \delta)/(3m) - 41\epsilon/2 - \sum_{s|q, s \text{ prime}} s^{-k-1}.$$

Step 2: Reduction from Sparse-ternary LWE to Sparse-ternary LWR: Bai et al. provide in [18, Theorem 6.4] a reduction from LWE with Gaussian noise to LWR, that is based on two independent reductions It can readily be seen that one of these reductions [18, Theorem 6.3] holds for any secret distribution with support on $\mathbb{Z}_q^{n*} = \{(x_1, \ldots, x_n) \in \mathbb{Z}_q^n \mid \gcd(x_1, x_2, \ldots, x_n, q) = 1\}$, and therefore can be applied to the case when the secret is chosen from $\{-1, 0, 1\}^n$. The other reduction [18, Theorem 5.1] however, implicitly assumes the secret to be chosen uniformly at random from $\mathbb{Z}_q^n$. Below, we describe an extension of [18, Theorem 5.1] that describes a reduction from LWE with Gaussian noise and sparse ternary secrets reduces to LWR with sparse-ternary secrets. Below, we will describe such an extension. U_B denotes the continuous uniform distribution in $[-B, \ldots, B]$.

Lemma A.8.2 (Adapted from [18, Theorem 5.1])**.** *Let n, m, q be positive integers. Let $\alpha, B > 0$ be real numbers with $B = \Omega(m\alpha/\log n)$ and $Bq \in \mathbb{Z}$. Let $m > \log\left(\binom{n}{h}2^h\right)/\log(\alpha + B)^{-1} \geq 1$. Then there is a polynomial time reduction from $\mathtt{LWE}_{n,m,q,D_\alpha}(\mathcal{U}(\mathcal{H}_n(h)))$ to $\mathtt{LWE}_{n,m,q,\phi}(\mathcal{U}(\mathcal{H}_n(h)))$ with $\phi = \frac{1}{q}\lfloor qU_B \rceil$.*

Proof. The reduction proceeds similar to that of [18, Theorem 5.1], relying on five steps. Steps 1, 3, 4 below proceed exactly as in [18, Theorem 5.1]. For steps 2 and 5, we mention our adaptations in order to prove the reduction for the case of sparse-ternary secrets, and the resulting conditions. We omit details for brevity.

1. A reduction from $\mathtt{dLWE}_{n,m,q,D_\alpha}$ to $\mathtt{dLWE}_{n,m,q,\psi}$, with $\psi = D_\alpha + U_B$.

2. A reduction from $\mathtt{dLWE}_{n,m,q,\psi}$ to $\mathtt{sLWE}_{n,m,q,\psi}$. We adapt the corresponding step in [18, Theorem 5.1] to work for the uniform distribution on $\mathcal{H}_n(h)$ instead of the uniform distribution on $\mathbb{Z}_q^n$. This results in the bound on m as stated in the lemma.

3. A reduction from $\mathtt{sLWE}_{n,m,q,\psi}$ to $\mathtt{sLWE}_{n,m,q,U_B}$.

4. A reduction from $\mathtt{sLWE}_{n,m,q,U_B}$ to $\mathtt{sLWE}_{n,m,q,\phi}$, with $\phi = \frac{1}{q}\lfloor qU_B \rceil$.

5. A reduction from $\texttt{sLWE}_{n,m,q,\phi}$ to $\texttt{dLWE}_{n,m,q,\phi}$. Since the modulus q is not a prime, the argument from [18, Theorem 5.1] cannot be applied. Instead, we extend an argument due to Regev (see, e.g, [88]) to prove the search-to-decision reduction, which requires that Bq is an integer. We first state an easy lemma.

   Lemma A.8.3. *Let $a > 1$, and let ϕ be the discrete probability distribution obtained by rounding the continuous uniform probability on $[-a, a]$ to the closest integer. If a is an integer, then $\sum_{k even} \phi(k) = \sum_{k odd} \phi(k) = \frac{1}{2}$.*

   Proof. For $|k| \leq \lfloor a \rfloor - 1$, the interval $[k - \frac{1}{2}, k + \frac{1}{2}]$ is a subset of $-[a, a]$, so that $\sum_{k \equiv 1-\lfloor a \rfloor \pmod 2} \phi(k) = \sum_{j=0}^{\lfloor a \rfloor -1} \phi(2j - \lfloor a \rfloor + 1) = \frac{\lfloor a \rfloor}{2a}$. □

   We are now in a position to extend Regev's reduction. Let ϕ be a probability distribution on $\mathbb{Z}_q$ such that $\sum_k \phi(2k) = \sum_k \phi(2k+1) = \frac{1}{2}$. For each $\mathbf{s} \in \mathbb{Z}_q^n$, the probability distribution $A_{\mathbf{s},\phi}$ on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is obtained by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly, choosing e according to ϕ, and outputting $(\mathbf{a}, (\mathbf{a}, \mathbf{s}) + e)$ where the additions are modulo q. If qB is integer, then a distinguisher for $\texttt{dLWE}_{n,m,q,\phi}(D_s)$ can be used to construct a solver for $\texttt{sLWE}_{n,m,q,\phi}(D_s)$ for any secret distribution D_s supported on $\{-1, 0, 1\}^n$, where ϕ is the discrete noise $\frac{1}{q}\lfloor qU_B \rceil$. Note that if Bq is integer, the noise ϕ is distributed as $\phi(k) = \frac{1}{2B}$ for $|k| \leq B - 1$, and $\phi(B) = \phi(-B) = \frac{1}{4B}$. We now show that if Bq is integer, then a distinguisher for deciding between uniform samples $(\mathbf{a}, u) \in U(\mathbb{Z}_q^n) \times U(Z_q)$ and samples $(\mathbf{a}, b)$ from $A_{\mathbf{s},\phi}$ for some unknown $s \in \mathcal{S} \subset \{-1, 0, 1\}^n$ can be used for solving. We show how to find s_1, the first coordinate of a secret. For each $k \in \mathbb{Z}_q$, we consider the following transformation. For each pair $(\mathbf{a}, b)$, we choose a random $r \in \mathbb{Z}_q$ and output $(\mathbf{a}', b') = (\mathbf{a} + (r, 0, \ldots, 0), b + rk)$. Clearly, this transformation takes the uniform distribution to itself. So let us now assume that $b = (\mathbf{a}, \mathbf{s}) + e$ for some $s \in \mathcal{S}$ and some error e. Then $b' = (\mathbf{a}', \mathbf{s}) + r(k - s_1) + e$. If $k = s_1$, then $(\mathbf{a}', b')$ is from $A_{\mathbf{s},\phi}$. If $|k - s_1| = 1$, then $r(k - s_1)$ is uniform over $\mathbb{Z}_q$, and so $(\mathbf{a}', \mathbf{b})$ follows the uniform distribution. Finally, we can have that $|k - s_1| = 2$. We consider $k - s_1 = 2$, the other case being similar. We then have that $b' = (\mathbf{a}, \mathbf{s}) + 2r + e \pmod q$. If q is odd, then $2r$ is uniformly distributed on $\mathbb{Z}_q$, so that $(\mathbf{a}', \mathbf{b})$ is uniformly distributed. If q is even, then $2r$ is distributed uniformly on the even elements of $\mathbb{Z}_q$. With our specific error distribution, e is even with probability $\frac{1}{2}$, so that $2r + e$ is distributed uniformly on $\mathbb{Z}_q$. So also in this case, $(\mathbf{a}', b)$ is distributed uniformly.

   □

Finally, we state the reduction from $\texttt{dLWE}_{n,m,q,D_\alpha}$ to $\texttt{dLWR}_{n,m,q,p}$, for the sparse-ternary secret distribution.

Lemma A.8.4. *Let p, q be positive integers such that p divides q. Let $\alpha' > 0$. Let $m' = m \cdot (q/p)$ with $m = O(\log n/\alpha')$ for $m' \geq m \geq n \geq 1$. There is a* *polynomial time reduction from* $\texttt{dLWE}_{n,m',q,D_{\alpha'}}$ *to* $\texttt{dLWR}_{n,m,q,p}$*, both defined for the sparse-ternary secret distribution.*

Proof. Let $B = q/2p$. The reduction has two steps:

1. A reduction from $\texttt{dLWE}_{n,m',q,D_{\alpha'}}$ to $\texttt{dLWE}_{n,m',q,\phi}$, where $B = \Omega(m'\alpha'/\log n)$. due to Lemma A.8.2.

2. A reduction from $\texttt{dLWE}_{n,m',q,\phi}$ to $\texttt{dLWR}_{n,m,q,p}$, due to [18, Theorem 6.3].

As $m' = m \cdot (q/p) = (q/p)O(\frac{\log n}{\alpha'})$, it follows that $B = q/2p = \Omega(m'\alpha'/\log n)$, so that Lemma A.8.2 indeed is applicable. □

Note that the conditions imposed by Lemma]A.8.2 imply that $1/\alpha$ must at least grow linearly in n. This is a common bottleneck in all known LWE to LWR reductions [18, 25, 19].

References

[1] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time via discrete Gaussian sampling. In *STOC*, pages 733–742, 2015.

[2] Miklós Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions. In *STOC*, pages 10–19, 1998.

[3] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

[4] Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In *EUROCRYPT*, pages 103–129, 2017.

[5] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wünderer. Estimate all the LWE, NTRU schemes! Cryptology ePrint Archive, Report 2018/331, 2018.

[6] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *EUROCRYPT*, 2019.

[7] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. NewHope without reconciliation. Cryptology ePrint Archive, Report 2016/1157, 2016.

[8] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. In *USENIX Security Symposium*, pages 327–343, 2016.

[9] Joel Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited: New reduction, properties and applications. In *CRYPTO*, pages 57–74, 2013.

[10] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *SODA*, pages 47–66, 2017.

[11] Yoshinori Aono and Phong Q. Nguyên. Random sampling revisited: lattice enumeration with discrete pruning. In *EUROCRYPT*, pages 65–102, 2017.

[12] Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In *ASIACRYPT*, pages 405–434, 2018.

[13] Roberto Avanzi, Joppe W. Bos, Léo Ducas, Eike Kiltz, Trancrède Lepoint, Vadim Lyubashevsky, John M. Shanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber. Technical report, National Institute of Standards and Technology, 2017. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[14] Hayo Baan, Sauvik Bhattacharya, Oscar Garcia-Morchon, Ronald Rietman, Ludo Tolhuizen, Jose-Luis Torre-Arce, and Zhenfei Zhang. Round2: KEM and PKE based on GLWR. Technical report, National Institute of Standards and Technology, November 2017. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[15] Hayo Baan, Sauvik Bhattacharya, Oscar Garcia-Morchon, Ronald Rietman, Ludo Tolhuizen, Jose-Luis Torre-Arce, and Zhenfei Zhang. Round2: KEM and PKE based on GLWR. Cryptology ePrint Archive, Report 2017/1183, 2017.

[16] L. Babai. On Lovász lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[17] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In *ACISP*, pages 322–337, 2014.

[18] Shi Bai, Adeline Langlois, Tancrède Lepoint, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: using the Rényi divergence rather than the statistical distance. *Journal of Cryptology*, 31(2):610–640, 2015.

[19] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, pages 719–737, 2011.

[20] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, pages 10–24, 2016. `https://eprint.iacr.org/2015/1128`.

[21] Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In *AFRICACRYPT*, pages 3–23, 2016.

[22] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73, 1993.

[23] Sauvik Bhattacharya, Oscar Garcia-Morchon, Ronald Rietman, and Ludo Tolhuizen. spKEX: An optimized lattice-based key exchange. Cryptology ePrint Archive, Report 2017/709, 2017.

[24] Jean-François Biasse and Fang Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In *SODA*, pages 893–902, 2016.

[25] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In *TCC*, pages 209–224, 2016.

[26] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *ASIACRYPT*, pages 41–69, 2011.

[27] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE Gates from Tensored Homomorphic Accumulator. In *AFRICACRYPT*, pages 217–251, 2018.

[28] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *CCS*, pages 1006–1018, 2016.

[29] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *IEEE S&P*, pages 553–570, 2015.

[30] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. In *Euro S&P*, pages 353–367, 2018.

[31] Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. Fly, you fool! Faster Frodo for the ARM Cortex-M4. IACR Cryptology ePrint Archive 2008/1116, November 2018.

[32] Joppe W. Bos, Michael Naehrig, and Joop van de Pol. Sieving for shortest vectors in ideal lattices: a practical perspective. *International Journal of Applied Cryptography*, 3(4):313–329, 2017.

[33] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computation Theory*, 6(3), 2014. Article No. 13.

[34] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.

[35] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT*, pages 1–20, 2011.

[36] Jung Hee Cheon, Kyoo Hyung Han, Jinsu Kim, Changmin Lee, and Yongha Son. A practical post-quantum public-key cryptosystem based on spLWE. In *ICISC*, pages 51–74, 2016.

[37] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yongsoo Song. Lizard: Cut off the tail! practical post-quantum public-key encryption from LWE and LWR. In *SCN*, pages 160–177, 2018.

[38] John H. Conway and Neil J.A. Sloane. *Sphere packings, lattices and groups*. Springer, 1999.

[39] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In *CRYPTO*, pages 430–448, 2005.

[40] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In *EUROCRYPT (II)*, pages 559–585, 2016.

[41] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short Stickelberger class relations and application to ideal-SVP. In *EUROCRYPT*, pages 324–348, 2016.

[42] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[43] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER. Technical report, National Institute of Standards and Technology, 2017. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[44] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In *AFRICACRYPT*, pages 282–305, 2018.

[45] TU Darmstadt. SVP challenge, 2018. Available at http://latticechallenge.org/svp-challenge/.

[46] Léo Ducas. Shortest vector from lattice sieving: a few dimensions for free. In *EUROCRYPT*, pages 125–145, 2018.

[47] Léo Ducas and Wessel P. J. van Woerden. The closest vector problem in tensored root lattices of type A and in their duals. *Designs, Codes and Cryptography*, 86(1):137–150, Jan 2018. https://eprint.iacr.org/2016/910.

[48] Morris J Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. Technical report, National Institute for Standards and Technology, August 2015. Federal Information Processing Standards (NIST FIPS)-202.

[49] FIPS. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, November 2001.

[50] Tim Fritzmann, Thomas Pöppelmann, and Johanna Sepulveda. Analysis of error-correcting codes for lattice-based key exchange. In *SAC*, pages 369–390, 2018.

[51] Nicolas Gama, Phong Q. Nguyên, and Oded Regev. Lattice enumeration using extreme pruning. In *EUROCRYPT*, pages 257–278, 2010.

[52] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219, 1996.

[53] Mike Hamburg. Graphs of "estimate all the LWE, NTRU schemes!" indexed to the "pqc lounge" data., 2017. Available at `https://bitwiseshiftleft.github.io/estimate-all-the-lwe-ntru-schemes.github.io/graphs`.

[54] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *CRYPTO*, pages 447–464, 2011.

[55] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time-memory trade-offs for tuple lattice sieving. In *PKC*, pages 407–436, 2018.

[56] Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing Parameters for NTRUEncrypt. In *CT-RSA*, pages 3–18, 2017.

[57] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

[58] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *TCC*, pages 341–371, 2017.

[59] James Howe, Ayesha Khalid, Marco Martinoli, Francesco Regazzoni, and Elisabeth Oswald. Fault Attack Countermeasures for Error Samplers in Lattice-Based Cryptography. Cryptology e-Print Archive, Report 2019/206, 2019. Accepted for publication at IEEE ISCAS 2019.

[60] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *CRYPTO*, pages 150–169, 2007.

[61] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In *CHES*, pages 232–252, 2017.

[62] Tsukasa Ishiguro, Shinsaku Kiyomoto, Yutaka Miyake, and Tsuyoshi Takagi. Parallel Gauss Sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In *PKC*, pages 411–428, 2014.

[63] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *STOC*, pages 193–206, 1983.

[64] Matthias J. Kannwischer, Joost Rijneveld, and Peter Schwabe. Faster multiplication in $\mathbf{Z}_{2^m}[x]$ on Cortex-M4 to speed up NIST PQC candidates. IACR Cryptology ePrint Archive 2008/1018, October 2018.

[65] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. (Accessed March 12, 2019.), 2018.

[66] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. In *FOCS*, pages 126–135, 2004.

[67] Aleksandr Nikolayevich Korkin and Yegor Ivanovich Zolotarev. Sur les formes quadratiques. *Mathematische Annalen*, 6(3):366–389, 1873.

[68] Thijs Laarhoven. *Search problems in cryptography. From fingerprinting to lattice sieving.* PhD thesis, Eindhoven University of Technology, 2016.

[69] Thijs Laarhoven and Artur Mariano. Progressive lattice sieving. In *PQCrypto*, pages 292–311, 2018.

[70] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2–3):375–400, 2015.

[71] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Code and Cryptography*, 77(3):565–599, 2015.

[72] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[73] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23, 2010.

[74] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC*, pages 21–39, 2004.

[75] Robby G. McKilliam, I. Vaughan L. Clarkson, and Barry G. Quinn. An Algorithm to Compute the Nearest Point in the Lattice A_n^*. *IEEE Transactions on Information Theory*, 54(9):4378–4381, 2008.

[76] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC*, pages 351–358, 2010.

[77] Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In *SODA*, pages 276–294, 2015.

[78] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *EUROCRYPT*, pages 820–849, 2016.

[79] Michael Naehrig, Erdem Alkim, Joppe W. Bos, Leo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2017. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[80] NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. POST-QUANTUM CRYPTO STANDARDIZATION. Call For Proposals Announcement, 2016.

[81] Chris Peikert. Lattice cryptography for the internet. In *PQCrypto*, pages 197–219, 2014.

[82] Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of ring-LWE for any ring and modulus. In *STOC*, pages 461–473, 2017.

[83] Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-SVP in ideal lattices with pre-processing. In *EUROCRYPT*, 2019.

[84] Michael E. Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *ACM SIGSAM Bulletin*, 15(1):37–44, 1981.

[85] Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *Cryptology ePrint Archive, Report 2009/605*, pages 1–7, 2009.

[86] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*, pages 433–444, 1991.

[87] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Number "Not Used" Once – Practical faut attack on pqm4 implementations of NIST Candidates. Cryptology ePrint Archive, Report 2018/211, 2018. Accepted for publication in COSADE 2019.

[88] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[89] Oded Regev. The learning with errors problem (invited survey). In *CCC*, pages 191–204, 2010.

[90] Markku-Juhani O. Saarinen. HILA5: Key Encapsulation Mechanism (KEM) and Public Key Encryption Algorithm. Technical report, National Institute of Standards and Technology, November 2017. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[91] Markku-Juhani O. Saarinen. HILA5: On reliability, reconciliation, and error correction for Ring-LWE encryption. In *SAC*, pages 192–212, 2017.

[92] Markku-Juhani O. Saarinen. Ring-LWE ciphertext compression and error correction: Tools for lightweight post-quantum cryptography. In *IoTPTS*, pages 15–22, April 2017.

[93] Markku-Juhani O. Saarinen, Sauvik Bhattacharya, Oscar Garcia-Morchon, Ronald Rietman, Ludo Tolhuizen, and Zhenfei Zhang. Shorter messages and faster post-quantum encryption with Round5 on Cortex M. In *CARDIS*, pages 95–110, 2019.

[94] Michael Schneider. Sieving for shortest vectors in ideal lattices. In *AFRICACRYPT*, pages 375–391, 2013.

[95] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2–3):201–224, 1987.

[96] Claus-Peter Schnorr and Matthias Euchner. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. *Math. Program.*, 66(2):181–199, September 1994.

[97] Claus-Peter Schnorr and Horst Helmut Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *EUROCRYPT*, pages 1–12, 1995.

[98] Naftali Sommer, Meir Feder, and Ofir Shalvi. Finding the closest lattice point by iterative slicing. *SIAM Journal of Discrete Mathematics*, 23(2):715–731, 2009.

[99] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, pages 617–635, 2009.

[100] Thomas Wunderer. Revisiting the hybrid attack: Improved analysis and refined security estimates. Cryptology ePrint Archive, Report 2016/733, 2016.

[101] Shang-Yi Yang, Po-Chun Kuo, Bo-Yin Yang, and Chen-Mou Cheng. Gauss sieve algorithm on GPUs. In *CT-RSA*, pages 39–57, 2017.