# Round2:
# KEM and PKE based on GLWR

Hayo Baan[1], Sauvik Bhattacharya[1], Oscar Garcia-Morchon[2], Ronald Rietman[1], Ludo Tolhuizen[1], Jose-Luis Torre-Arce[1], and Zhenfei Zhang[3]

[1]Philips Research, Eindhoven, The Netherlands
[2]Philips IP&S, Eindhoven, The Netherlands
[3]Onboard Security Inc., Wilmington, USA
Contacting authors: {sauvik.bhattacharya, ludo.tolhuizen}@philips.com

**Abstract**

Cryptographic primitives that are secure against quantum computing are receiving growing attention with recent, steady advances in quantum computing and standardization initiatives in post-quantum cryptography by NIST and ETSI. Lattice-based cryptography is one of the families in post-quantum cryptography, demonstrating desirable features such as well-understood security, efficient performance, and versatility.

In this work, we present *Round2* that consists of a key-encapsulation mechanism and a public-key encryption scheme. Round2 is based on the General Learning with Rounding problem, that unifies the Learning with Rounding and Ring Learning with Rounding problems. Round2's construction using the above problem allows for a unified description and implementation. The key-encapsulation mechanism and public-key encryption scheme furthermore share common building blocks, simplifying (security and operational) analysis and code review. Round2's reliance on prime cyclotomic rings offers a large design space that allows fine-tuning of parameters to required security levels. The use of rounding reduces bandwidth requirements and the use of sparse-trinary secrets improves CPU performance and decryption success rates. Finally, Round2 includes various approaches of refreshing the system public parameter $\boldsymbol{A}$, allowing efficient ways of preventing precomputation and back-door attacks.

# Contents

# 1 Introduction

Existing public-key cryptographic primitives will be broken once a quantum-computer is available. Thus, it is important to have suitable quantum-secure alternatives. Lattice-based primitives are believed to be quantum-secure, they are relatively well studied and offer good performance. In this work, we present *Round2*, consisting of a key-encapsulation mechanism Round2.KEM and public-key encryption scheme Round2.PKE, based on the General Learning with Rounding problem. Round2 is characterized by the following features.

*Unified*: depending on the input configuration parameters, the same KEM and PKE algorithm description and implementation instantiate different underlying problems, namely Learning with Rounding and Ring Learning with Rounding. The reason for such a unified construction is to provide a simple way for the user to choose between a scheme that either prioritizes security and has no additional structure (as in non-ring based schemes), or prioritizes performance (as is the case for ring-based schemes). This approach also simplifies code review and ensures a migration strategy from day one if attacks on ring-based constructions were discovered. The unified configuration of Round2 allows the modulus $q$ to be a power of two, and is implemented to support ring and non-ring configurations seamlessly, without relying on NTT. Instead, polynomial operations are implemented by matrix-vector multiplications – similar to the non-ring case – by lifting operations from the prime cyclotomic ring to the NTRU [27, 29] ring. It is also possible to configure Round2 with a prime $q$ such that $q \equiv 1 \pmod{n+1}$– this enables NTT friendly optimizations in the ring setting.

*Designed for performance*: We choose to construct Round2 on the General Learning with rounding problem in order to achieve low bandwidth requirements. Round2.KEM and Round2.PKE utilize secrets that are trinary (i.e., the entries are 0,1 or -1) and sparse, i.e., contain relatively few non-zeroes. Having sparse trinary secrets plays an important role in the operational correctness of the scheme; furthermore, it improves CPU performance. Finally, Round2 presents a number of methods for refreshing the system public parameter $\boldsymbol{A}$ using permutations, that provides a tradeoff between security (to prevent pre-computation and back-door attacks) and performance (e.g., regenerating a fully random $\boldsymbol{A}$ in every session is expensive).

*Prime cyclotomic ring*: Round2 relies on the ring $\mathbb{Z}_q[x]/\left(\Phi_{n+1}(x)\right)$ with $n+1$ prime so that $(n+1)$-th cyclotomic polynomial $\Phi_{n+1}(x)$ satisfies $\Phi_{n+1}(x) = 1 + x + \ldots + x^n$. In case that $q$ is a power of two, we impose the additional requirement that $\Phi_{n+1}$ is irreducible modulo two, ensuring that $\mathbb{Z}_q[x]/\Phi_{n+1}(x)$ does not have proper subrings. This ring choice provides a large pool of potential $(n, q)$ values so that bandwidth requirements and security levels can be finely tuned.

The rest of this document is organized as follows: Section 2.1 details the rationale of our submission in more detail. In Section 2.2, we present preliminaries and notation. Section 2.3 states the problem underlying the security of the schemes constituting Round2. The specification of these schemes,

Round2.KEM and Round2.PKE, along with their internal building blocks are presented in Section 2.4. Section 2.5 includes security definitions, a proof of IND-CPA and IND-CCA security for Round2.KEM and Round2.PKE respectively, as well as a proof of hardness for the underlying problem for the non-ring configuration, i.e., sparse-trinary learning with rounding. Section 2.6 presents a security analysis with respect to known attacks against Round2 and its underlying hard problem (lattice reduction-based, hybrid, and sparse-secret specific attacks). Section 2.7 presents an analysis of the decryption/decapsulation failure probability. Section 2.8 presents implementation considerations and optimization details. Section 2.9 presents Round2 configurations, parameters and performance details. Finally, Section 2.10 presents advantages and limitations of Round2.

# 2 Algorithm Specifications

## 2.1 Design Rationale

This submission proposes Round2 that consists of algorithms for the key encapsulation mechanism Round2.KEM and the public-key encryption scheme Round2.PKE. The proposed algorithms fall under the category of lattice-based cryptography, in particular, they rely on the General Learning with Rounding (GLWR) problem. This problem has been chosen for the design of Round2 for the reasons explained in the following subsections.

### 2.1.1 A Unified Design

A key feature of Round2 is that it has been designed to instantiate the LWR problem and the Ring LWR (RLWR) problem in a seamless and unified way. This is done by defining the General LWR problem, on which Round2 is based, that can instantiate LWR or RLWR depending on the input parameters. The reasons behind this choice are as follows:

Round2 is adaptive and can be applied to multiple environments. On the one hand, LWR-based algorithms are required by 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 [31, 27]). 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.

Round2 reduces code analysis and maintenance since the unified scheme definitions of Round2.KEM and Round2.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 found in future, then an alternative secure solution would already be available and could be deployed directly.

### 2.1.2 Parameter choices for optimized performance

Parameters in GLWR and Round2 are chosen to allow for optimized performance.

- The usage of GLWR, i.e., LWR and RLWR, rather than their LWE counterparts, leads to lower bandwidth requirements in Round2 since fewer bits need to be transmitted per coefficient.
- GLWR avoids sampling of non-uniform noise, and thus requires the generation of less random data.
- Round 2 relies on a definition of GLWR with sparse trinary secrets. This simplifies implementation and reduces the probability of decryption/decapsulation errors.
- The ring configuration of Round2 relies on the RLWR problem over a cyclotomic ring $\mathbb{Z}_q[x]/\Phi_{n+1}(x)$ with $n+1$ prime. This problem is well studied: there exist reductions [8, 13] from the RLWE problem [31] to RLWR, and the former is well-studied for the ring in question. Operations over this ring can be mapped to an NTRU [27] ring $\mathbb{Z}_q[x]/(x^{n+1}-1)$ to improve performance.
- Round2 can be instantiated with a unified set of parameters, uRound2. This set of parameters allows for a single implementation for a wide range of security levels, relying on either LWR or RLWR. The moduli $q$ and $p$ are powers of two for uRound2, since this simplifies the implementation of the rounding function. Similarly, the modular computations can be realized by ignoring the most significant bits.
- Round2 can be instantiated with an NTT-friendly set of parameters, nRound2. This set of parameters is intended for cases where computational performance is the priority.
- 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. Round2 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 problem is $\mathbb{Z}_q[x]/\Phi_{2n}(x)$ where $n$ is a power of 2, 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 [17] and [2]. However, requiring that $n$ be a power of 2 narrows the choice of $n$: 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 [16], NTRUEncrypt [27], NTRU-KEM [29] and more.

The following observations can be made:

- Kyber [16] 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, and as suggested by [38], although the NTRU problem remains hard for this ring, the decisional RLWE problem over this ring seems to be easy.

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 [29]. With the resulting ring, there is a wide range of $n$ to choose from, for various security levels. In addition, as shown in [36], decisional RLWE over this ring remains hard for any modulus; this gives us confidence in the underlying design and security of Round2. Note that although operating over the same ring as the NTRU-KEM scheme, Round2 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 [36], we can use any modulus that optimizes our performances. Common choices of the modulus are

- A number theoretical transform (NTT) friendly prime number, such as 12289 in [2];
- A composite number that fits in a data type for modern computers, such as $2^{32} - 1$ in [17];
- A power of 2 that makes modulo operations and integer multiplications efficient, such as $2^{11}$ in NTRUEncrypt [27].

In our proposal, we consider two sets of parameters as will be explained in Section 2.9. The first set of parameters considers a prime cyclotomic polynomial ring with a modulus $q$ that is a power of two such that the $\Phi_{n+1}$ is irreducible modulo two. As $\Phi_{n+1}$ 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 $q$ of less than 16 bits. That is, each coefficient fits in a `uint16_t` type. For intermediate values during computations, overflows can be ignored,as overflown bits "*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 second set of parameters considers the prime cyclotomic polynomial ring with a prime modulus $q$ such that these parameters are NTT-friendly. This allows the use of NTT to speed up specific configurations.

We remark that the use of a composite modulus in [17], as well as the result from [36] suggest that the particular modulus does not have much effect on the hardness of the problem; the length of the modulus is more important from this point of view. Consequently, any modulus of similar size should deliver a similar security, and therefore we choose one that is most efficient depending on the use case.

## 2.2 Preliminaries

For each positive integer $a$, we denote the set $\{0, 1, \ldots, a-1\}$ by $\mathbb{Z}_a$.

For a set $A$, we denote by $a \xleftarrow{\$} A$ that $a$ is drawn uniformly from $A$. If $\chi$ is a probability distribution, then $a \leftarrow \chi$ means that $a$ is drawn at random according to the probability distribution $\chi$.

Logarithms are in base 2, unless specified otherwise.

**Modular reductions.** For a positive integer $\alpha$ 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 $\lfloor x \rceil$ rounding of $x$ to the closest integer, with rounding up in case of a tie.

**Compression and decompression.** Let $a, b$ be integers such that $a > b$. The functions $\text{Compress}_{a\to b} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}_b$ and $\text{Decompress}_{b\to a} : \mathbb{Z} \to \mathbb{Z}_a$ are defined as

$$\text{Compress}_{a\to b}(x, e) = \left\langle \left\lfloor \frac{b}{a} \cdot x + \frac{ge}{a} \right\rceil \right\rangle_b, \text{ where } g = \gcd(a, b) \tag{1}$$

$$\text{Decompress}_{b\to a}(x) = \left\langle \left\lfloor \frac{a}{b} \cdot x \right\rceil \right\rangle_a \tag{2}$$

If $(x, e)$ is uniformly distributed on $\mathbb{Z}_a \times (-\frac{b}{2g}, \frac{b}{2g}] \cap \mathbb{Z}$, then $\text{Compress}_{a\to b}(x, e)$ is uniformly distributed on $\mathbb{Z}_b$.

We also use the randomized compression function $\text{RCompress}_{a,b} : \mathbb{Z} \to \mathbb{Z}_b$ defined as

$$\text{RCompress}_{a\to b}(x) \xleftarrow{\$} \{\text{Compress}_{a\to b}(x, e) \mid e \in (-\frac{b}{2g}, \frac{b}{2g}] \cap \mathbb{Z}\}, \text{ where } g = \gcd(a, b). \tag{3}$$

Note that if $b$ divides $a$, then $\frac{b}{2g} = \frac{1}{2}$, so that $\text{RCompress}_{a\to b}(x) = \langle \lfloor \frac{b}{a} x \rceil \rangle_b$ for all $x \in \mathbb{Z}$ and thus $\text{RCompress}_{a\to b}$ is a deterministic function in this case.

**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 the set of polynomials of degree less than $n$ with all coefficients in $\mathbb{Z}_a$. We call a polynomial in $\mathcal{R}_n$ *trinary* if all its coefficients are 0, 1 or $-1$. Throughout this document, regular font letters denote elements from $\mathcal{R}_n$, and bold lower case letters represent vectors with coefficients in $\mathcal{R}_n$. All vectors are column vectors. Bold upper case letters are matrices. The transpose of a vector $\boldsymbol{v}$ or a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{v}^T$ or $\boldsymbol{A}^T$.

**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 trinary 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 trinary 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 trinary 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 in Round2 consist of matrices that contain (column) vectors that are distributed according to the distribution $\chi_S$ defined over the set $\mathcal{H}_{n,d/n}(h)$. For generating a secret matrix $\boldsymbol{R}$ in CPA-PKE.Encryptwithro from an input variable $\rho$, a function $f_R$ is employed.

## 2.3 Underlying Problem

The problem underlying the security of Round2 is the **General LWR Problem** formally defined as follows:

**Definition 2.3.0.1** (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 $\mathcal{R}_n^{d/n}$.*

*The search version of the GLWR problem $\texttt{sGLWR}_{d,n,m,q,p}(D_s)$ is as follows: given $m$ samples of the form $(\boldsymbol{a}_i, b_i = RCompress_{q\to p}(\langle \boldsymbol{a}_i^{\mathsf{T}} \boldsymbol{s} \rangle_q))$ 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 $\texttt{dGLWR}_{d,n,m,q,p}(D_s)$ is to distinguish between the uniform distribution on $\mathcal{R}_{n,q}^{d/n} \times \mathcal{R}_{n,p}$ and the distribution $(\boldsymbol{a}_i, b_i = RCompress_{q\to p}(\langle \boldsymbol{a}_i^T \boldsymbol{s} \rangle_q))$ with $\boldsymbol{a}_i \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n}$ and a fixed $\boldsymbol{s} \leftarrow D_s$.*

When $n = 1$, the GLWR problem is equivalent to the LWR problem [8] 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-trinary secrets $\texttt{LWR}_{\text{spt}}$. In [22] it is claimed that the hardness of $\texttt{LWR}_{\text{spt}}$ 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 [13]. 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 $D_\alpha$, to the decision version of $\texttt{LWR}_{\text{spt}}$. See Section 2.5.6 and Theorem 2.5.6.1 for more details.

When $n = d \geq 1$ 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 Ring LWR (RLWR) problem defined on $\Phi_{d+1}(x)$, 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-trinary secrets $\texttt{RLWR}_{\text{spt}}$. For brevity, when $D_s = \mathcal{U}\left(\mathcal{H}_{n,d/n}(h)\right)$, we denote $\texttt{GLWR}_{d,n,m,q,p}\left(\mathcal{U}\left(\mathcal{H}_{n,d/n}(h)\right)\right)$ as $\texttt{GLWR}_{\text{spt}}$. 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.

## 2.4 Round2

Our proposal is called Round2. It includes a CPA-KEM called Round2.KEM and a CCA-PKE called Round2.PKE. Round2.KEM is constructed using a public-key encryption scheme, CPA-PKE, as a building block. Round2.PKE is constructed using a key-encapsulation mechanism, CCA-PKE, as a building block. All algorithms in these schemes use random choices, and scheme-specific mappings that are described in the following sections with each scheme.

Section 2.4.1 contains the description of a building block for generating public scheme parameters. Round2.KEM is described in Section 2.4.3, preceded by the description of its building block CPA-PKE in Section 2.4.2. Round2.PKE is descirbed in Section 2.4.5, preceded by its building block CCA-KEM in Section 2.4.4. Figure 1 provides an overview of our proposal. It also shows the different configurations of the proposed schemes based on the underlying GLWR problem.

### 2.4.1 Internal building block: Definitions of $f_n^\tau(\sigma)$

Round2.KEM and Round2.PKE require the generation of the GLWR public parameter $\boldsymbol{A} \in \mathcal{R}_{n,q}^{d/n \times d/n}$. In order to make the choice for $\boldsymbol{A}$ explicit, a seed $\sigma$ is used, as well a description of how to construct $\boldsymbol{A}$ from $\sigma$. Round2 gives four options for $f_n^\tau(\sigma)$, the function used to compute $\boldsymbol{A}$ in CPA-PKE.Keygen and CPA-PKE.Encryptwithrho.

Given positive integers $d, n, q, \mu, B$ and $\tau \in \{0, 1, 2, 3\}$, $f_n^\tau$ is a mapping from $\{0,1\}^{\mu B}$ to $\mathcal{R}_{n,q}^{d/n \times d/n}$. The functions $f_n^0, f_n^1, f_n^2$ are to be applied when $n = 1$; the function $f_n^3$ is only to be applied if $n = d$. To emphasize this fact, the notation $f_{n=1}^\tau$ is used for $\tau = 0, 1, 2$, and $f_{n=d}^3$ instead of $f_n^3$ will be used. In the definitions of $f_n^\tau$ given below, $s' = H(0x0000|s)$ indicates the derivation of seed $s'$ from seed $s$ by using padding $0x0000$ and applying a hash function H. The expression $a_{i,j} = DRBG(s)[id + j]$ means that value $a_{i,j}$ in $\boldsymbol{A}$ is obtained as

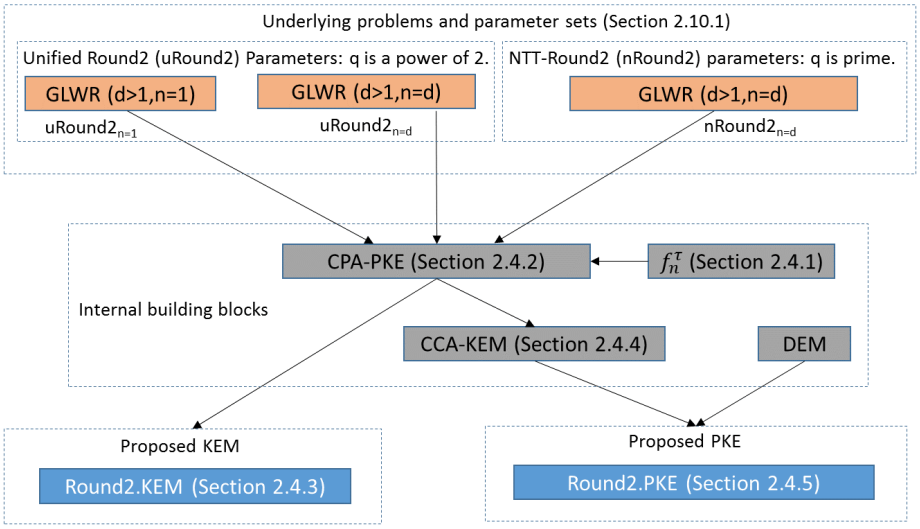


Figure 1: Submission overview

the $id + j$ element derived from the deterministic random bit generator $DRBG$ applied to seed $s$:

0. $f^0_{n=1}(\sigma)$ computes the elements in $\boldsymbol{A}$ by applying a DRBG on seed $\sigma_0$:
   $\sigma_0 = H(0x0000|\sigma)$,
   $a_{i,j} = DRBG(\sigma_0)[id + j]$.

1. $f^1_{n=1}(\sigma)$ computes the elements in $\boldsymbol{A}$ by applying a permutation to the elements of a fixed $d \times d$ matrix $\boldsymbol{A}^{\text{fixed}}$. The permutation is computed by applying a DRBG to seed $\sigma_1$ as follows:
   $\sigma_1 = H(0x0001|\sigma)$,
   $o_i = DRBG(\sigma_1)[i]$ for $0 \leq i \leq d - 1$
   $a_{i,j} = a^{\text{fixed}}_{i,(j+o_i)(mod \quad d)}$.
   The fixed matrix is computed from a fixed seed $seed^{\text{fixed}}$ by means of a DRBG. This process is done once, and $seed^{\text{fixed}}$ and the DRBG are public parameters:
   $a^{\text{fixed}}_{i,j} = DRBG(seed^{\text{fixed}})[id + j]$

2. $f^2_{n=1}(\sigma)$ computes the elements in $\boldsymbol{A}$ by applying a permutation to a set of $L = q$ elements. The permutation is computed as follows:
   $\sigma_1 = H(0x0001|\sigma)$,
   $o_i = DRBG(\sigma_1)[i]$ for $0 \leq i \leq d - 1$
   The set of elements on which the permutation is applied is computed as:

$\sigma_0 = H(0x0000|\sigma)$.

The entries of the matrix $\boldsymbol{A}$ are obtained as

$a_{i,j} = DRBG(\sigma_0)[(j + o_i)(mod\ \ L)]$.

3. $f^3_{n=d}(\sigma)$ computes the elements in $\boldsymbol{A}$ by applying a DRBG on a seed:

   $\sigma_0 = H(0x0000|\sigma)$,

   $a_i = DRBG(\sigma_0)[i]$ for $0 \leq i \leq d-1$

   In this case, $\mathbf{A}$ consists of a single polynomial with coefficients $a_0, \ldots, a_{d-1}$.

Pros and cons between security and performance for each of these functions will be presented in Sections 2.8.3 and 2.6.7. In particular, Tables 15 and 22 compare performance of the different variants of computing $\boldsymbol{A}$ for NIST level 5 of Round2.KEM and Round2.PKE.

### 2.4.2 Internal building block: CPA-PKE

This section describes CPA-PKE, the CPA-secure public key encryption that is a building block in both Round2.KEM and Round2.PKE. Apart from algorithms for keygeneration, encryption and decryption, CPA-PKE also has the algorithm CPA-PKE.Encryptwithrho that describes encryption with an explicit input variable that governs the randomness that is implicit in CPA-PKE.Encrypt. Algorithm CPA-PKE.Encryptwithrho will be used in Round2.PKE.

CPA-PKE has various system parameters, *viz* positive integers $n$, $d$, $h$, $p$, $q$, $t$, $B$, $\overline{n}$, $\overline{m}$, $\mu$. In the proposed configurations, $n \in \{1, d\}$, and $t|p$, that is, $t$ divides $p$, and $2^B|t$. It is required that

$$\mu \leq \overline{n} \cdot \overline{m} \cdot n.$$

The function $\text{Sample}_\mu : \boldsymbol{C} \in \mathcal{R}^{\overline{n} \times \overline{m}}_{n,p} \rightarrow \mathbb{Z}^\mu_p$ outputs the values of $\mu$ of the $\overline{n} \cdot \overline{m} \cdot n$ polynomial coefficients present in $\boldsymbol{C}$.

CPA-PKE.Keygen generates a secret matrix $\boldsymbol{S}$ with trinary columns drawn independently according to a distribution $\chi_S$ with support on $\left(\mathcal{H}_{n,d/n}(h)\right)^{1 \times \overline{n}}$. Algorithm CPA-PKE.Encryptwithrho employs a deterministic function $f_R$ for generating a secret matrix $\boldsymbol{R}$ from an input $\rho$. If $\rho$ is uniformly distributed, each column of $f_R(\rho)$ is distributed to $\chi_S$. Its also uses a mapping $f_{E_U} : \{0,1\}^{\mu B}$ such that for each $\rho$, all coefficients in $f_{E_U}(\rho)$ are elements from from $(-\frac{p}{2g}, \frac{p}{2g}] \cap \mathbb{Z}$, where $g = \gcd(p, q)$.

**Algorithm 1:** CPA-PKE.Keygen()

**parameters:** Integers $p, q, n, h, d, \overline{n}$

**input** : -

**output** : $pk \in \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, sk \in (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{n}}$

Choose $\tau \in \{0,1,2,3\}$

$\sigma \xleftarrow{\$} \{0,1\}^{\mu B}$

$\boldsymbol{A} = f_n^{\tau}(\sigma)$

$\boldsymbol{S} \leftarrow \chi_S^{\overline{n}}$

$\boldsymbol{B} = \text{RCompress}_{q \to p}(\langle \boldsymbol{AS} \rangle_q)$

$pk = (\tau, \sigma, \boldsymbol{B})$

$sk = \boldsymbol{S}$

**return** $(pk, sk)$

**Algorithm 2:** CPA-PKE.Encryptwithrho$(pk, m, \rho)$

**parameters:** Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, B$

**input** : $pk = (\tau, \sigma, \boldsymbol{B}) \in \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, m \in \mathbb{Z}_{2^B}^{\mu}, \rho \in \{0,1\}^{\mu B}$

**output** : $c = (\boldsymbol{U}, \boldsymbol{v}) \in \mathcal{R}_{n,p}^{d/n \times \overline{m}} \times \mathbb{Z}_t^{\mu}$

$\boldsymbol{A} = f_n^{\tau}(\sigma)$

$\boldsymbol{R} = f_R(\rho)$

$\boldsymbol{E}_U = f_{E_U}(\rho)$

$\boldsymbol{U} = \text{Compress}_{q \to p}(\langle \boldsymbol{A}^T \boldsymbol{R} \rangle_q, \boldsymbol{E}_U)$

$\boldsymbol{X} = \langle \boldsymbol{B}^T \boldsymbol{R} \rangle_p$

$\boldsymbol{v} = \langle \text{RCompress}_{p \to t}(\text{Sample}_{\mu}(\boldsymbol{X})) + \frac{t}{2^B} \cdot m \rangle_t$

$c = (\boldsymbol{U}, \boldsymbol{v})$

**return** $c$

**Algorithm 3:** CPA-PKE.Encrypt($pk, m$)

**parameters:** Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, B$
**input** : $pk \in \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, m \in \mathbb{Z}_{2^B}^{\mu}$
**output** : $c \in R_{n,p}^{d/n \times \overline{m}} \times \mathbb{Z}_t^{\mu}$

1 $\rho \xleftarrow{\$} \{0,1\}^{\mu B}$
2 $c =$CPA-PKE.Encryptwithrho($pk, m, \rho$)
3 **return** $c$

**Algorithm 4:** CPA-PKE.Decrypt($sk, c$)

**parameters:** Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, B$
**input** : $sk = \boldsymbol{S} \in (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{n}}, c = (\boldsymbol{U}, \boldsymbol{v}) \in \mathcal{R}_{n,p}^{d/n \times \overline{m}} \times \mathbb{Z}_t^{\mu}$
**output** : $\hat{m} \in \mathbb{Z}_{2^B}^{\mu}$

1 $\boldsymbol{v}_p = \text{Decompress}_{t \to p}(\boldsymbol{v})$
2 $\hat{m} = \text{RCompress}_{p \to 2^B}(\langle \boldsymbol{v}_p - \text{Sample}_{\mu}(\langle \boldsymbol{S}^T \boldsymbol{U} \rangle_p) \rangle_p)$
3 **return** $\hat{m}$

### 2.4.3 Submission proposal: Round2.KEM

This section describes Round2.KEM, an IND-CPA-secure key encapsulation method. It builds on CPA-PKE (Section 2.4.2). In addition to the parameters and functions from CPA-PKE, it uses a system parameter $\lambda_1$ and an injection $\texttt{bin}_1 : \mathcal{R}_{n,p}^{d/n \times \overline{m}} \times \mathbb{Z}_t^{\mu} \to \{0,1\}^{\lambda_1}$, and a hash function $H : \{0,1\}^* \to \{0,1\}^{\mu B}$. The system parameter $\lambda_1$ satisfies

$$\lambda_1 = d \cdot \overline{m} \log_2(p) + \mu \log_2(t).$$

The mapping $\texttt{bin}_1$ concatenates a vector consisting of the binary representation of the coefficients of the polynomial entries $d/n \times \overline{m}$ matrix, and the binary representations of the entries of a vector in $\mathbb{Z}_t^{\mu}$. Round2.KEM also uses the bijection $\phi : \mathbb{Z}_{2^B}^{\mu} \to \{0,1\}^{\mu B}$, that replaces each of the $\mu$ entries of its input by the $B$-bits binary representation of that entry.

**Algorithm 5:** Round2.KEM.Keygen()

**parameters:** Integers $p, q, n, h, d, \overline{n}$
**input** : -
**output** : $pk \in \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, sk \in (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{n}}$

1 $(pk, sk) =$ CPA-PKE.Keygen()
2 **return** $(pk, sk)$

**Algorithm 6:** Round2.KEM.Encapsulate($pk$)

**parameters:** Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, B$
**input** : $pk \in \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$
**output** : $(c, K) \in (\mathcal{R}_{n,p}^{d/n \times \overline{m}} \times \mathbb{Z}_t^{\mu}) \times \{0,1\}^{\mu B}$

$m \xleftarrow{\$} \mathbb{Z}_{2^B}^{\mu}$
$c =$ CPA-PKE.Encrypt$(pk, m))$
$K = H(\phi(m), \mathtt{bin}_1(c))$
**return** $(c, K)$

**Algorithm 7:** Round2.KEM.Decapsulate($sk, c$)

**parameters:** Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, B$
**input** : $sk \in (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{n}}, c \in \mathcal{R}_{n,p}^{d/n \times \overline{m}} \times \mathbb{Z}_t^{\mu}$
**output** : $K \in \{0,1\}^{\mu B}$

$m =$ CPA-PKE.Decrypt$(sk, c)$
$K = H(\phi(m), \mathtt{bin}_1(c))$
**return** $K$

### 2.4.4 Internal building block: CCA-KEM

This section describes CCA-KEM that is a building block for Round2.PKE. As CCA-KEM is obtained by applying a KEM variant of the Fujisaki-Okamoto transform [26] to CPA-PKE, it is IND-CCA secure.

CCA-KEM has several system parameters and functions in addition to those from CPA-PKE and Round2.KEM. It has two hash functions, $G : \{0,1\}^* \to \{0,1\}^{\mu B} \times \{0,1\}^{\mu B} \times \{0,1\}^{\mu B}$, and $H : \{0,1\}^* \to \{0,1\}^{\mu B}$. Moreover, it has the system parameter $\lambda_2$, and an injection $\mathtt{bin}_2 : \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}} \to \{0,1\}^{\lambda_2}$. The parameter $\lambda_2$ satisfies

$$\lambda_2 = 8 + \mu B + d \cdot \overline{n} \log_2(p)$$

The mapping $\mathtt{bin}_2$ concatenates an 8-bit representation of an element in $\{0,1,2,3\}$, a binary representation of an element in $\{0,1\}^{\mu B}$, and a vector consisting of the binary representations of the coefficients of the polynomial entries of a $d/n \times \overline{n}$ matrix.

**Algorithm 8:** CCA-KEM.Keygen()

**parameters:** Integers $p, q, n, h, d, \overline{n}, \mu, B$
**input** : -
**output** : $pk \in \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, sk \in (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{n}} \times \{0,1\}^{\mu B} \times (\{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}})$

$(pk, sk_{CPA-PKE})$ =CPA-PKE.Keygen()
$z \xleftarrow{\$} \{0,1\}^{\mu B}$
$sk = (sk_{CPA-PKE}, z, pk)$
**return** $(pk, sk)$

**Algorithm 9:** CCA-KEM.Encapsulate($pk$)

**parameters:** Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, B$
**input** : $pk \in \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$
**output** : $c = (\boldsymbol{U}, \boldsymbol{v}, g) \in \mathcal{R}_{n,p}^{d/n \times \overline{m}} \times \mathbb{Z}_t^{\mu} \times \{0,1\}^{\mu B}, K \in \{0,1\}^{\mu B}$

$m \xleftarrow{\$} \mathbb{Z}_{2^B}^{\mu}$
$(L, \rho, g) = G(\phi(m), \mathtt{bin}_2(pk))$
$(\boldsymbol{U}, \boldsymbol{v})$ = CPA-PKE.Encryptwithrho$(pk, m, \rho)$
$c = (\boldsymbol{U}, \boldsymbol{v}, g)$
$K = H(L, \mathtt{bin}_1(\boldsymbol{U}, \boldsymbol{v}), g)$
**return** $(c, K)$

**Algorithm 10:** CCA-KEM.Decapsulate($sk, c$)

**parameters:** Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, B$
**input** : $sk = (sk_{CPA-PKE}, z, pk) \in (\mathcal{R}_{n,d/n}(h))^{1 \times \overline{n}} \times \{0,1\}^{\mu B} \times (\{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}})$, $c = (\boldsymbol{U}, \boldsymbol{v}, g) \in \mathcal{R}_{n,p}^{d/n \times \overline{m}} \times \mathbb{Z}_t^{\mu} \times \{0,1\}^{\mu B}$
**output** : $K \in \{0,1\}^{\mu B}$

$m'$ = CPA-PKE.Decrypt$(sk_{CPA-PKE}, (\boldsymbol{U}, \boldsymbol{v}))$
$(L', \rho', g') = G(\phi(m'), \mathtt{bin}_2(pk))$
$(\boldsymbol{U'}, \boldsymbol{v'})$= CPA-PKE.Encryptwithrho$(pk, m', \rho')$
**if** $(\boldsymbol{U'}, \boldsymbol{v'}, g') = (\boldsymbol{U}, \boldsymbol{v}, g)$ **then**
    **return** $K = H(L', \mathtt{bin}_1(\boldsymbol{U}, \boldsymbol{v}), g)$
**else**
    **return** $K = H(z, \mathtt{bin}_1(\boldsymbol{U}, \boldsymbol{v}), g)$
**end if**

### 2.4.5 Submission proposal: Round2.PKE

Round2.PKE is an IND-CCA public-key encryption scheme constructed by combining CCA-KEM (Section 2.4.4) and a data encapsulation mechanism (DEM).

The DEM encapsulates a message $M$ of length $mlen$ bytes with the key $K$ encapsulated in the KEM, resulting in an encapsulated message $c2$ of length $clen$ bytes. In the inverse operation, $c2$ is decapsulated to the original message $M$ using $K$.

**Algorithm 11:** Round2.PKE.Keygen()

**parameters:** Integers $p, q, n, h, d, \overline{n}, \mu, B$
**input** : -
**output** : $pk \in \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$, $sk \in (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{n}} \times \{0,1\}^{\mu B} \times (\{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}})$

1 $(pk, sk)$ =CCA-KEM.Keygen()
2 **return** $(pk, sk)$

**Algorithm 12:** Round2.PKE.Encrypt($pk, M$)

**parameters:** Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, B$
**input** : $pk \in \{0,1,2,3\} \times \{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$, $mlen \in \mathbb{Z}, M \in \mathbb{Z}_{256}^{mlen}$
**output** : $c = (c1, clen, c2) \in (\mathcal{R}_{n,p}^{d/n \times \overline{m}} \times Z_t^{\mu} \times \{0,1\}^{\mu B}) \times \mathbb{Z} \times \mathbb{Z}_{256}^{clen}$

1 $(c1, K)$ = CCA-KEM.Encapsulate($pk$)
2 $(clen, c2)$ = DEM($K, M$)
3 $c = (c1, clen, c2)$
4 **return** $c$

**Algorithm 13:** Round2.PKE.Decrypt($sk, c$)

**parameters:** Integers $p, t, q, n, d, \overline{m}, \overline{n}, \mu, B$
**input** : $sk \in (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{n}} \times \{0,1\}^{\mu B} \times (\{0,1\}^{\mu B} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}), c = (c1, clen, c2) \in (\mathcal{R}_{n,p}^{d/n \times \overline{m}} \times Z_t^{\mu} \times \{0,1\}^{\mu B}) \times \mathbb{Z} \times \mathbb{Z}_{256}^{clen}$
**output** : $M \in \mathbb{Z}_{256}^{mlen}$

1 $K$ =CCA-KEM.Decapsulate($sk, c1$)
2 $(mlen, M)$ =DEM$^{-1}(K, c2)$
3 **return** $(mlen, M)$

## 2.5 Expected Security Strength

This section is organized as follows: Section 2.5.1 introduces notions of security based on indistinguishablity of ciphertexts or encapsulated keys. Section 2.5.2 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 trinary secrets ($\texttt{LWR}_{\text{spt}}$) and the Ring Learning with Rounding problem with sparse trinary secrets ($\texttt{RLWR}_{\text{spt}}$). Sections 2.5.4 and 2.5.5 contain the

first main result of this section – a proof of IND-CPA security for Round2.KEM (Theorem 2.5.4.1), and a proof of IND-CCA security for Round2.PKE (Theorems 2.5.5.1 and 2.5.5.2), assuming the hardness of the above problems. Finally, Section 2.5.6 and Theorem 2.5.6.1 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.

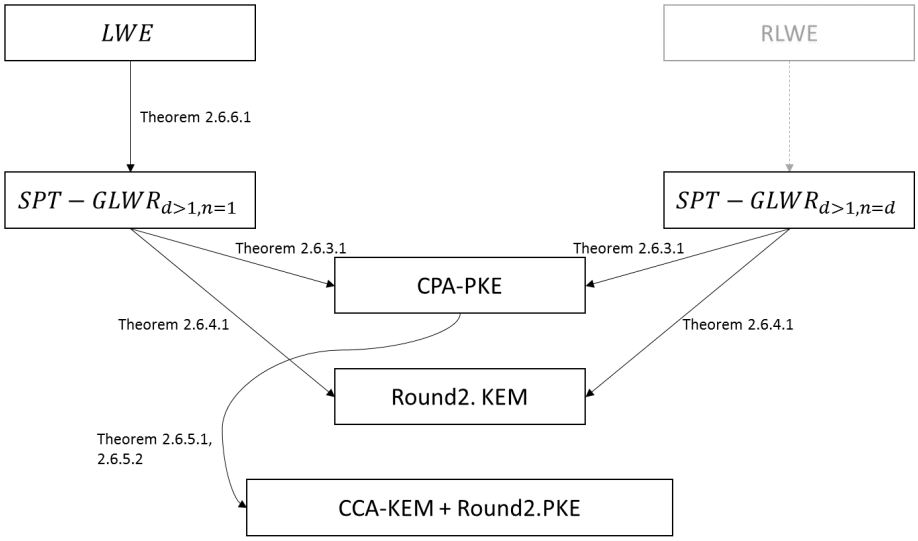


Figure 2: Summary of reductions involved in the security proofs for Round2.KEM and Round2.PKE. "SPT" refers to a variant of the problem in question where the secret is sparse-trinary, instead of uniform in $\mathbb{Z}_q^d$.

Figure 2 gives an overview of the major reductions that are contributions of this section.

### 2.5.1 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 [34, 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 [34, Section 4.A.3].

Table 1: 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 2.5.1.1** (IND-CPA Secure PKE)**.** *Let PKE=(Keygen, Enc, Dec) be a public key encryption scheme with message space $\mathcal{M}$. Let $\lambda$ be a security parameter. The IND-CPA game is defined in Table 1, and the IND-CPA advantage of an adversary $\mathcal{A}$ against PKE is defined as*

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

**Definition 2.5.1.2** (IND-CCA Secure PKE)**.** *Let PKE=(Keygen, Enc, Dec) be a public key encryption scheme with message space $\mathcal{M}$. Let $\lambda$ be a security parameter. Let $\mathcal{A}$ be an adversary against PKE. The IND-CCA game is defined as Table 1, 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*

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

Table 2: 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 2.5.1.3** (IND-CPA Secure KEM)**.** *Let KEM=(*$\texttt{Keygen}$*,* $\texttt{Encaps}$*,* $\texttt{Decaps}$*) be a key encapsulation mechanism with key space $\mathcal{K}$. Let $\lambda$ be a security parameter. The* IND-CPA *game is defined in Table 2, and the* IND-CPA *advantage of an adversary $\mathcal{A}$ against* KEM *is defined as*

$$Adv^{\textit{IND-CPA}}_{\textit{KEM}}(\mathcal{A}) = \mid Pr[\textit{IND-CPA}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2} \mid .$$

**Definition 2.5.1.4** (IND-CCA Secure KEM)**.** *Let KEM=(*$\texttt{Keygen}$*,* $\texttt{Encaps}$*,* $\texttt{Decaps}$*) be a key encapsulation mechanism with key space $\mathcal{K}$. Let $\lambda$ be a security parameter. Let $\mathcal{A}$ be an adversary against KEM. The* IND-CCA *game is defined in Table 2, 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*

$$Adv^{\textit{IND-CCA}}_{\textit{KEM}}(\mathcal{A}^{\texttt{Decaps}(\cdot)}) = \mid Pr[\textit{IND-CCA}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2} \mid .$$

In the random oracle model [9] (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 $H_Q$ [14] that can be queried up to $q_{H_Q}$ times on arbitrary superpositions of input strings.

### 2.5.2 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 2.4.2), 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-trinary 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 [8, 3, 13, 7] and established based on the hardness of the Learning with Errors (LWE) problem [37]. The most recent work are two independent reductions to LWR from LWE: the first, due to Bai et al. [7, 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. [13, 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.

Bai et al.'s reduction [7, Theorem 6.4] is an essential component that we use in Section 2.5.6 to prove a reduction from $\mathtt{dLWE}_{n,m',q,D_\alpha}\left(\mathcal{U}\left(\mathbb{Z}_q\right)\right)$ to $\mathtt{dLWR}_{n,m,q,p}\left(\mathcal{U}(\mathcal{H}_n\left(h\right))\right)$, i.e., to $\mathtt{dLWR}_{\mathrm{spt}}$.

**Provable Security in the ring case:** Next, we recall hardness results for the ring case, i.e., for Decision-RLWR [8]. To the best of our knowledge, the only existing result on the hardness of Decision-RLWR is due to [8, 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 $\{-B, \ldots, B\}$, 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 [8, 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.

### 2.5.3 IND-CPA security of CPA-PKE

In this section it is shown that the public-key encryption scheme CPA-PKE from Section 2.4.2 is IND-CPA secure, based on the hardness of the decision GLWR problem with sparse-trinary secrets. In Section 2.5.5 this result will be used to show that Round2.PKE is IND-CCA Secure.

In the following, let $S_i$ denote the event that in game $G_i$, the adversary $\mathcal{A}$ has output 1 (i.e., that bit $b$ in game $G_i$ was correctly guessed). In the proof, the following version of CPA-PKE.Keygen() is used. It takes as input the GLWR public parameter $\boldsymbol{A} \in \mathcal{R}_{n,q}^{d/n \times d/n}$ instead of generating it as part of the algorithm:

**Algorithm 14:** CPA-PKE.Keygen($\boldsymbol{A}$)

**parameters:** Integers $p, q, n, h, d, \overline{n}$
**input** : $\boldsymbol{A} \in \mathcal{R}_{n,q}^{d/n \times d/n}$.
**output** : $pk \in \mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}, sk \in (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{n}}$

$\boldsymbol{S} \xleftarrow{\$} \chi_S^{\overline{n}}$
$\boldsymbol{B} = \text{RCompress}_{q \to p}(\langle \boldsymbol{AS} \rangle_q)$
$pk = (\boldsymbol{A}, \boldsymbol{B})$
$sk = \boldsymbol{S}$
**return** $(pk, sk)$

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-trinary secrets (see Section 2.3). For the above parameters, we define the GLWR oracle $O_{m,\chi_S,\boldsymbol{s}}$ for a secret distribution $\chi_S$ that returns $m$ GLWR samples as follows:

$$O_{m,\chi_S,\boldsymbol{s}} : \boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{m \times d/n},\ \boldsymbol{s} \leftarrow \chi_S;\ \text{return}\ \left(\boldsymbol{A}, \text{RCompress}_{q \to p}(\boldsymbol{As})\right) \quad (4)$$

The decision-GLWR problem with sparse-trinary secrets is to distinguish between the distributions $\mathcal{U}(\mathcal{R}_{n,q}^{d/n}) \times \mathcal{U}(\mathcal{R}_{n,p})$ 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

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

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

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:

$$O_{m,\chi_S,\overline{n},\boldsymbol{S}} : \boldsymbol{A} \xleftarrow{\$} \mathcal{U}\left(\mathcal{R}_{n,q}^{m \times d/n}\right),\ \boldsymbol{S} \leftarrow (\chi_S)^{\overline{n}};\ \text{return}\ \left(\boldsymbol{A}, \text{RCompress}_{q \to p}(\boldsymbol{AS})\right) \quad (5)$$

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 CPA-PKE is IND-CPA secure assuming the hardness of decision-GLWR with sparse-trinary secrets.

**Theorem 2.5.3.1.** *If $f_n : \{0,1\}^{\mu B} \to \mathcal{R}_{n,q}^{d/n \times d/n}$ is a secure mapping, and $f_R$ is indistinguishable from $(\chi_S)^{\overline{m}}$, then CPA-PKE is IND-CPA secure under the hardness assumption of the Decision-GLWR problem with sparse-trinary 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 adversaries $\mathcal{B}, \mathcal{D}$ and reduction algorithms $\mathcal{C}'$, $\mathcal{E}', \mathcal{F}'$ such that*

$$\begin{aligned} Adv_{CPA\text{-}PKE}^{IND\text{-}CPA}(\mathcal{A}) \leq\ & Adv^{f_n}(\mathcal{B}) + \overline{n} \cdot Adv_{d,n,d/n,q,p}^{\texttt{dGLWR}_{spt}}(\mathcal{A} \circ \mathcal{C}') + Adv^{f_R}(\mathcal{D}) \\ & + \overline{m} \cdot Adv_{d,n,d/n,q,p}^{\texttt{dGLWR}_{spt}}(\mathcal{A} \circ \mathcal{E}') + \overline{m} \cdot Adv_{d,n,\overline{n},p,t}^{\texttt{dGLWR}_{spt}}(\mathcal{A} \circ \mathcal{F}') \end{aligned} \quad (6)$$

Table 3: IND-CPA games for CPA-PKE: Games $G_0$ and $G_1$

| **Game $G_0$** | **Game $G_1$** |
|---|---|
| 1. $((\tau, \sigma, \boldsymbol{B}), \boldsymbol{S}) = \text{CPA-PKE.Keygen}()$. | 1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $((\boldsymbol{A}, \boldsymbol{B}), \boldsymbol{S}) = \text{CPA-PKE.Keygen}(\boldsymbol{A})$. |
| 2. Choose $b \xleftarrow{\$} \{0,1\}$. | 2. Choose $b \xleftarrow{\$} \{0,1\}$. |
| 3. $(m_0, m_1, st) = \mathcal{A}(f_n(\sigma), \boldsymbol{B})$. | 3. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$. |
| 4. $(\boldsymbol{U}, \boldsymbol{v}) = \text{CPA-PKE.Encrypt}\left((\sigma, \boldsymbol{B}), m_b\right)$. | 4. $(\boldsymbol{U}, \boldsymbol{v}) = \text{CPA-PKE.Encrypt}\left((\boldsymbol{A}, \boldsymbol{B}), m_b\right)$. |
| 5. $b' = \mathcal{A}\left((f_n(\sigma), \boldsymbol{B}), (\boldsymbol{U}, \boldsymbol{v}), st\right)$. | 5. $b' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \boldsymbol{v}), st\right)$. |
| 6. **return**$[(b' = b)]$. | 6. **return**$[(b' = b)]$. |

*In this equation, $Adv^{f_n}(\mathcal{B})$ is the advantage of $\mathcal{B}$ in distinguishing the output of the mapping $f_n$ from uniform, and $Adv^{\mathtt{dGLWR}_{spt}}_{d,n,m,q_1,q_2}(\mathcal{Z})$ is the advantage of adversary $\mathcal{Z}$ in distinguishing $m$ GLWR samples (with sparse-trinary secrets) from uniform, with the GLWR problem defined for the parameters $d, n, q_1, q_2$. Finally, the adversary $\mathcal{D}$ distinguishes between*

$$U(\{f_R(\rho) \mid \rho \in \{0,1\}^{\mu B}\}) \text{ and } (\chi_S)^{\overline{m}}.$$

*The runtimes of $\mathcal{B}, \mathcal{D}, \mathcal{A} \circ \mathcal{C}', \mathcal{A} \circ \mathcal{E}', \mathcal{A} \circ \mathcal{F}'$ are essentially the same as that of $\mathcal{A}$.*

*Proof.* We prove the above theorem via a sequence of IND-CPA games shown in Tables 3, 4 and 5, following the methodology of Peikert et al. in [35, Lemma 4.1] and that of [16, Theorem 3.3]:

**Game $\mathbf{G_0}$** is the original IND-CPA game for CPA-PKE. By definition, the advantage of the adversary for game $G_0$ is:

$$\text{Adv}^{\text{IND-CPA}}_{\text{PKE}}(\mathcal{A}) = |\Pr(S_0) - 1/2| \tag{7}$$

**Game $\mathbf{G_1}$** is different from game $G_0$ *only* in the fact that $\boldsymbol{A}$ is generated fully randomly in step 1, instead of via the mapping $f_n$. An adversary that can distinguish between game $G_0$ and game $G_1$ immediately leads to a distinguisher $\mathcal{B}$ between $f_n$ and the uniform distribution:

$$|\Pr(S_0) - \Pr(S_1)| \leq \text{Adv}^{f_n}(\mathcal{B}) \tag{8}$$

Note that out of the possible choices of $f_n$ for creating $\boldsymbol{A}$ as described in the specification of algorithms, $f^0_{n=1}$ and $f^1_{n=1}$ ensure that the resulting $\boldsymbol{A}$ is defined according to the non-ring, i.e., it yields a LWR instantiation of the decision-GLWR problem. The fourth choice for $f_n$, i.e., $f^3_{n=d}$ ensures that the resulting $\boldsymbol{A}$ is defined according to the ring, i.e., it yields a RLWR instantiation of the decision-GLWR problem.

Table 4: IND-CPA games for CPA-PKE: Games $G_2$ and $G_3$

| **Game $G_2$** | **Game $G_3$** |
|---|---|
| 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. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$. |
| 2. Choose $b \xleftarrow{\$} \{0,1\}$. | 2. Choose $b \xleftarrow{\$} \{0,1\}$. |
| 3. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$. | 3. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$. |
| 4. $(\boldsymbol{U}, \boldsymbol{v}) = \text{CPA-PKE.Encrypt}\left((\boldsymbol{A}, \boldsymbol{B}), m_b\right)$. | 4. $\boldsymbol{R} \xleftarrow{\$} (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{m}}$. |
| 5. $b' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \boldsymbol{v}), st\right)$. | 5. $\boldsymbol{U} = \text{RCompress}_{q \to p}(\boldsymbol{A}^T \boldsymbol{R})$. |
| 6. $\textbf{Output}[(b' = b)]$. | 6. $\boldsymbol{X} = \langle \boldsymbol{B}^T \boldsymbol{R} \rangle_p$. |
| | 7. $\boldsymbol{v} = \langle \text{RCompress}_{p \to t}(\text{Sample}_{\mu}(\boldsymbol{X})) + \frac{t}{2^B} \cdot m_b \rangle_t$. |
| | 8. $b' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \boldsymbol{v}), st\right)$. |
| | 9. $\textbf{Output}[(b' = b)]$. |

Table 5: IND-CPA games for CPA-PKE: 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. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$. |
| 2. Choose $b \xleftarrow{\$} \{0,1\}$. | 2. Choose $b \xleftarrow{\$} \{0,1\}$. |
| 3. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$. | 3. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$. |
| 4. $\boldsymbol{R} \xleftarrow{\$} (\mathcal{H}_{n,d}(h))^{1 \times \overline{m}}$. | 4. $\boldsymbol{R} \xleftarrow{\$} (\mathcal{H}_{n,d}(h))^{1 \times \overline{m}}$. |
| 5. $\boldsymbol{U} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{m}}$. | 5. $\boldsymbol{U} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{m}}$. |
| 6. $\boldsymbol{X} = \langle \boldsymbol{B}^T \boldsymbol{R} \rangle_p$. | 6. $\boldsymbol{X} \xleftarrow{\$} \mathcal{R}_{n,p}^{\overline{n} \times \overline{m}}$. |
| 7. $\boldsymbol{v} = \langle \text{RCompress}_{p \to t}(\text{Sample}_{\mu}(\boldsymbol{X})) + \frac{t}{2^B} \cdot m_b \rangle_t$. | 7. $\boldsymbol{v} = \langle \text{RCompress}_{p \to t}(\text{Sample}_{\mu}(\boldsymbol{X})) + \frac{t}{2^B} \cdot m_b \rangle_t$. |
| 8. $b' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \boldsymbol{v}), st\right)$. | 8. $b' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \boldsymbol{v}), st\right)$. |
| 9. $\textbf{Output}[(b' = b)]$. | 9. $\textbf{Output}[(b' = b)]$. |

**Game $\mathbf{G_2}$** is different from game $G_1$ *only* in the fact that $(\boldsymbol{A}, \boldsymbol{B})$ is a sample from the uniform distribution on $\mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$ instead of from $O_{d/n, \chi_S, \overline{n}, \boldsymbol{S}}$. Under the decision GLWR($\chi_S$) assumption, these two distributions are indistinguishable with a factor $\overline{n}$. Indeed, assume $\mathcal{A}$ can distinguish between $\text{G}_1$ and $\text{G}_2$. We describe an algorithm $\mathcal{C}$ that distinguishes between samples from $O_{d/n, \chi_S, \overline{n}, \boldsymbol{S}}$ and those from $\mathcal{U}(\mathcal{R}_{n,q}^{d/n \times d/n}) \times \mathcal{U}(\mathcal{R}_{n,p}^{d/n \times \overline{n}})$ using $\mathcal{A}$.

**Algorithm** $\mathcal{C}$
**Input** $(\boldsymbol{A}, \boldsymbol{B}) \in \mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$.
1. Choose $b \xleftarrow{\$} \mathcal{U}(\{0,1\})$.
2. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$
3. $(\boldsymbol{U}, \boldsymbol{v}) = \text{CPA-PKE.Encrypt}\left((\boldsymbol{A}, \boldsymbol{B}), m_b\right)$
4. $b' = \mathcal{A}\left((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \boldsymbol{v}), st\right)$
5. $\textbf{Output}[(b' = b)]$

If $(\boldsymbol{A}, \boldsymbol{B}) \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$, then the outputs of $\mathcal{C}$ and Game $\mathrm{G}_2$ have equal distribution. If $(\boldsymbol{A}, \boldsymbol{B})$ is drawn from $O_{d/n,\chi_S,\overline{n},\boldsymbol{S}}$, then the outputs of $\mathcal{C}$ and $G_1$ have equal distribution. The advantage $\mathrm{Adv}(\mathcal{A} \circ \mathcal{C})$ in distinguishing between $O_{d/n,\chi_S,\overline{n},\boldsymbol{S}}$ and the uniform distribution thus satisfies

$$\mathrm{Adv}(\mathcal{A} \circ \mathcal{C}) = \mid \Pr(S_1) - \Pr(S_2) \mid . \tag{9}$$

By a standard hybrid argument, there exists an algorithm $\mathcal{C}'$ for distinguishing between the distributions $O_{d/n,\chi_S,\boldsymbol{s}}$ and $\mathcal{U}(\mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n})$ such that

$$\mathrm{Adv}(\mathcal{A} \circ \mathcal{C}) \leq \overline{n} \cdot \mathrm{Adv}_{d,n,d/n,q,p}^{\mathtt{dGLWR}_{\mathrm{spt}}}(\mathcal{A} \circ \mathcal{C}') \tag{10}$$

for $\chi_S := \mathcal{U}(\mathcal{H}_{n,d/n}(h))$.

**Game $\mathbf{G_3}$** is different from game $G_2$ only in the generation of $\boldsymbol{R}$. Therefore a distinguisher $\mathcal{D}$ between the two distributions for $\boldsymbol{R}$ can be created for which

$$\mathrm{Adv}^{f_R}(\mathcal{D}) = \mid \Pr(S_2) - \Pr(S_3) \mid . \tag{11}$$

The two distributions are $\mathcal{U}(\{f_R(\rho) \mid \rho \in \{0,1\}^{\mu_B}\})$ and $\chi_S^{\overline{m}} = (\mathcal{U}(\mathcal{H}_{n,d/n}(h))^{1 \times \overline{m}}$.

**Game $\mathbf{G_4}$** is different from game $G_3$ only in the generation $\boldsymbol{U}$. Therefore a distinguisher $\mathcal{E}$ between these two distributions for $\boldsymbol{U}$ can be created for which

$$\mathrm{Adv}(\mathcal{A} \circ \mathcal{E}) = \mid \Pr(S_3) - \Pr(S_4) \mid . \tag{12}$$

Similar to the reasoning with the distinguisher $\mathcal{C}$, it can be shown that there exists an algorithm $\mathcal{E}'$ for distinguishing between $O_{d/n,\chi_S,\boldsymbol{s}}$ and $\mathcal{U}(\mathcal{R}_{n,q}^{d/n} \times \mathcal{R}_{n,p})$. such that

$$\mathrm{Adv}(\mathcal{A} \circ \mathcal{E}) \leq \overline{m} \cdot \mathrm{Adv}_{d,n,d/n,q,p}^{\mathtt{dGLWR}_{\mathrm{spt}}}(\mathcal{A} \circ \mathcal{E}') \tag{13}$$

for $\chi_S := U\left(\mathcal{H}_{n,d/n}(h)\right)$.

**Game $\mathbf{G_5}$** is different from game $G_4$ only in the generation of $\boldsymbol{X}$. Consider the following algorithm $\mathcal{F}$.

**Algorithm $\mathcal{F}$**
**Input** $(\boldsymbol{B}, \boldsymbol{Y}) \in \mathcal{R}_{n,p}^{d/n \times \overline{n}} \times \mathcal{R}_{n,t}^{\overline{n} \times \overline{m}}$
1. Choose $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$
2. Choose $b \xleftarrow{\$} \{0,1\}$.
3. $(m_0, m_1, st) = \mathcal{A}(\boldsymbol{A}, \boldsymbol{B})$
4. $\boldsymbol{U} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{m}}$
5. $\boldsymbol{v} = \langle \mathrm{Sample}_{\mu}(\boldsymbol{Y}) + \frac{t}{2^B} m_b \rangle_t$
6. $b' = \mathcal{A}((\boldsymbol{A}, \boldsymbol{B}), (\boldsymbol{U}, \boldsymbol{v}), st)$
7. **Output** [ $(b' = b)$ ]

If $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$ and $\boldsymbol{Y} = \text{RCompress}_{p \to t}(\langle \boldsymbol{B}^T \boldsymbol{R} \rangle_p)$ with $\boldsymbol{R} \xleftarrow{\$} (\mathcal{H}_{n,d/n}(h))^{1 \times \overline{m}}$, then the outputs of Algorithm $\mathcal{F}$ and game $G_4$ have equal distribution. If $(\boldsymbol{B}, \boldsymbol{Y}) \xleftarrow{\$} \mathcal{U}(\mathcal{R}_{n,p}^{d/n \times \overline{n}}) \times \mathcal{U}(\mathcal{R}_{n,t}^{\overline{n} \times \overline{m}})$, then the outputs of Algorithm $\mathcal{F}$ and Game $G_5$ have equal distribution. This is so as $\text{RCompress}_{p \to t}()$ maps uniform variables to uniform variables. Algorithm $\mathcal{F}$ distinguishes between these two distributions with advantage satisfying

$$\text{Adv}(\mathcal{A} \circ \mathcal{F}) = |\Pr(S_4) - \Pr(S_5)|.$$

By a standard hybrid argument, there exists a distinguisher $\mathcal{F}'$ between the uniform distribution on $\mathcal{R}_{n,p}^{d/n \times \overline{n}} \times \mathcal{R}_{n,t}^{1 \times \overline{n}}$ and the distribution $(\boldsymbol{B}, \text{RCompress}_{p \to t}(\langle \boldsymbol{B}^T \boldsymbol{r} \rangle_p)$ with $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$ and $\boldsymbol{r} \xleftarrow{\$} \mathcal{H}_{n,d}(h)$ such that

$$\text{Adv}(\mathcal{A} \circ \mathcal{F}) \leq \overline{m} \cdot \text{Adv}_{d,n,\overline{n},p,t}^{\texttt{dGLWR}_{\text{spt}}}(\mathcal{A} \circ \mathcal{F}') \tag{14}$$

for $\chi_S := \mathcal{U}(\mathcal{H}_{n,d/n}(h))$.

In game $G_5$, as $\boldsymbol{X}$ is uniformly distributed, $\text{Sample}_\mu(\boldsymbol{X})$ is uniformly distributed. As $\text{RCompress}_{p \to t}()$ maps uniform variables to uniform variables, the variable $\text{RCompress}_{p \to t}((\text{Sample}_\mu(\boldsymbol{X}))$ is uniformly distributed. As a consequence, for each message $m_b$, the variable $\boldsymbol{v} = \langle \text{RCompress}_{p \to t}((\text{Sample}_\mu(\boldsymbol{X})) + \frac{t}{2^B} m \rangle_t$ in game $G_5$ is uniformly distributed. As $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{U}$ are distributed uniformly as well,

$$\Pr(S_5) = 1/2. \tag{15}$$

Combination of equations 7 to 15 yields the proof of Theorem 2.5.3.1.

□

### 2.5.4 IND-CPA Security of Round2.KEM

This section presents a proof that Round2.KEM is IND-CPA secure, based on the hardness of the decision GLWR problem with sparse-trinary secrets. Similar to the proof of Theorem 2.5.3.1, a sequence of games will be presented. Again, versions of Round2.KEM algorithms are used that have $\boldsymbol{A}$ as input.

**Theorem 2.5.4.1.** *If $f_n : \{0,1\}^{\mu B} \to \mathcal{R}_{n,q}^{d/n \times d/n}$ is a secure mapping, $f_R$ is in distinguishible from $(\chi_S)^{\overline{m}}$ and $H$ is a secure pseudorandom function, Round2.KEM is IND-CPA secure under the hardness assumption of the Decision-GLWR problem with sparse-trinary secrets. More precisely, if $Adv_{Round2.KEM}^{IND\text{-}CPA}(\mathcal{A})$ is the advantage of adversary $\mathcal{A}$ in the IND-CPA game, then there exist adversaries $\mathcal{B}, \mathcal{D}, \mathcal{G}$ and reduction algorithms $\mathcal{C}'$, $\mathcal{E}'$ and $\mathcal{F}'$ such that*

$$\begin{aligned} Adv_{Round2.KEM}^{IND\text{-}CPA}(\mathcal{A}) \leq\ & Adv^{f_n}(\mathcal{B}) + \overline{n} \cdot Adv_{d,n,d/n,q,p}^{\texttt{dGLWR}_{spt}}(\mathcal{A} \circ \mathcal{C}') + Adv^{f_R}(\mathcal{D}) \\ & + \overline{m} \cdot Adv_{d,n,d/n,q,p}^{\texttt{dGLWR}_{spt}}(\mathcal{A} \circ \mathcal{E}') + \overline{m} \cdot Adv_{d,n,\overline{n},p,t}^{\texttt{dGLWR}_{spt}}(\mathcal{A} \circ \mathcal{F}') + Adv^{H}(\mathcal{G}) \end{aligned} \tag{16}$$

*In this equation, $Adv^{f_n}(\mathcal{B})$ is the advantage of $\mathcal{B}$ in distinguishing the output of the mapping $f_n$ from uniform, and $Adv_{d,n,m,q_1,q_2}^{\texttt{dGLWR}_{spt}}(\mathcal{Z})$ is the advantage of an*

*adversary $\mathcal{Z}$ in distinguishing $m$ GLWR samples (with sparse-trinary secrets) from uniform, with the GLWR problem defined for the parameters $d, n, q_1, q_2$. The adversary $\mathcal{D}$ distinguishes between*

$$U(\{f_R(\rho) \mid \rho \in \{0,1\}^{\mu B}\}) \text{ and } (\chi_S)^{\overline{m}}.$$

*Finally, $Adv^H(\mathcal{G})$ is the advantage of an adversary $\mathcal{G}$ in distinguishing the output of the pseudorandom function $H$ (given uniform input) from uniform. The runtimes of $\mathcal{B}, \mathcal{D}, \mathcal{G}, \mathcal{A} \circ \mathcal{C}', \mathcal{A} \circ \mathcal{E}', \mathcal{A} \circ \mathcal{F}'$ are essentially the same as $\mathcal{A}$.*

*Proof.* The proof for Theorem 2.5.4.1, proceeds via a similar sequence of games as in the proof for Theorem 2.5.3.1 (shown in Tables 6, 7 and 8). The event that algorithm $\mathcal{A}$ ouputs a 1 in game $G_i$ is denoted by $S_i$. Game $G_0$ is the original IND-CPA game for Round2.KEM, and so

$$\mathrm{Adv}^{\text{IND-CPA}}_{\text{Round2.KEM}}(\mathcal{A}) = \mid \Pr(S_0) - 1/2 \mid \tag{17}$$

Table 6: IND-CPA games for Round2.KEM: Games $G_0$ and $G_1$

| **Game $G_0$** | **Game $G_1$** |
|---|---|
| 1. $(pk = (\tau, \sigma, \boldsymbol{B}), sk = \boldsymbol{S}) =$ Round2.KEM.Keygen(). | 1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $(pk = (\boldsymbol{A}, \boldsymbol{B}), sk = \boldsymbol{S}) =$ Round2.KEM.Keygen($\boldsymbol{A}$). |
| 2. Choose $b \xleftarrow{\$} \{0,1\}$. | 2. Choose $b \xleftarrow{\$} \{0,1\}$. |
| 3. $(c = (\boldsymbol{U}, \boldsymbol{v}), K_0) =$ Round2.KEM.Encapsulate $(pk)$. | 3. $(c = (\boldsymbol{U}, \boldsymbol{v}), K_0) =$ Round2.KEM.Encapsulate $(\boldsymbol{A}, \boldsymbol{B})$. |
| 4. $K_1 \xleftarrow{\$} \{0,1\}^{\mu B}$. | 4. $K_1 \xleftarrow{\$} \{0,1\}^{\mu B}$. |
| 5. $b' = \mathcal{A}(pk, c, K_b)$. | 5. $b' = \mathcal{A}(pk, c, K_b)$. |
| 6. Output $[(b' = b)]$. | 6. Output $[(b' = b)]$. |

Similarly to the case for CPA-PKE, there is a distinguisher $\mathcal{B}$ between $f_n$ and the uniform distribution

$$\mid \Pr(S_0) - \Pr(S_1) \mid \leq \mathrm{Adv}^{f_n}(\mathcal{B}) \tag{18}$$

Games $G_1$ and $G_2$ only differ in the way that the matrix $\boldsymbol{B}$ is generated. Consider the following algorithm.

**Algorithm** $\mathcal{C}$
**Input** $(\boldsymbol{A}, \boldsymbol{B}) \in \mathcal{R}_{n,q}^{d/n \times d/n} \times \mathcal{R}_{n,p}^{d/n \times \overline{n}}$.
1. Choose $b \xleftarrow{\$} \{0,1\}$.
2. $(c = (\boldsymbol{U}, \boldsymbol{v}), K_0) =$ Round2.KEM.Encapsulate $(pk = (\boldsymbol{A}, \boldsymbol{B}))$.
3. $K_1 \xleftarrow{\$} \{0,1\}^{\mu B}$
4. $b' = \mathcal{A}(pk = (\boldsymbol{A}, \boldsymbol{B}), c, K_b)$.
5. **Output**$[(b' = b)]$.

The input $(\boldsymbol{A}, \boldsymbol{B})$ to algorithm $\mathcal{C}$ is distributed as $O_{d/n, \chi_S, \overline{n}, \boldsymbol{S}}$ if the input is

Table 7: IND-CPA games for Round2.KEM: Games $G_2$ and $G_3$

| **Game $G_2$** | **Game $G_3$** |
|---|---|
| 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. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$. |
| 2. Choose $b \xleftarrow{\$} \{0,1\}$. | 2. Choose $b \xleftarrow{\$} \{0,1\}$. |
| 3. $(c = (\boldsymbol{U}, \boldsymbol{v}), K_0) = \text{Round2.KEM.Encapsulate}(\boldsymbol{A}, \boldsymbol{B})$. | 3. $m \xleftarrow{\$} \mathbb{Z}_{2^B}^{\mu}$. |
| | 4. $\boldsymbol{R} \xleftarrow{\$} \left(\mathcal{H}_{n,d/n}(h)\right)^{1 \times \overline{m}}$. |
| | 5. $\boldsymbol{U} = \text{RCompress}_{q \to p}(\boldsymbol{A}^T \boldsymbol{R})$. |
| | 6. $\boldsymbol{X} = \langle \boldsymbol{B}^T \boldsymbol{R} \rangle_p$. |
| | 7. $\boldsymbol{v} = \langle \text{RCompress}_{p \to t}(\text{Sample}_{\mu}(\boldsymbol{X})) + \frac{t}{2^B} \cdot m \rangle_t$. |
| | 8. $K_0 = H(\phi(m), \texttt{bin}_1(c = (\boldsymbol{U}, \boldsymbol{v})))$. |
| 4. $K_1 \xleftarrow{\$} \{0,1\}^{\mu B}$. | 9. $K_1 \xleftarrow{\$} \{0,1\}^{\mu B}$. |
| 5. $b' = \mathcal{A}(pk, c, K_b)$. | 10. $b' = \mathcal{A}(pk, c = (\boldsymbol{U}, \boldsymbol{v}), K_b)$. |
| 6. Output $[(b' = b)]$. | 11. Output $[(b' = b)]$. |

from game $G_1$; it is distributed uniformly if the input is from game $G_2$. Algorithm $\mathcal{C}$ is thus a distinguisher between $O_{d/n,\chi_S,\overline{n},\boldsymbol{S}}$ and the uniform distribution with advantage equal to $|\Pr(S_2) - \Pr(S_1)|$. Similar to the proof for IND-CPA security for CPA-PKE, it follows that there is an algorithm $\mathcal{C}'$ such that

$$|\Pr(S_1) - \Pr(S_2)| \leq \overline{n} \cdot \text{Adv}^{\texttt{dGLWR}_{\text{spt}}}_{d,n,d/n,q,p}(\mathcal{A} \circ \mathcal{C}') \tag{19}$$

Games $G_2$ and $G_3$ only differ in the generation of $\boldsymbol{R}$. It is therefore possible to create a distinguisher $\mathcal{D}$ between the two distributions for $\boldsymbol{R}$ such that

$$\text{Adv}^{f_R}(\mathcal{D}) = |\Pr(S_2) - \Pr(S_3)|\,. \tag{20}$$

The two distributions are $\mathcal{U}(\{f_R(\rho) \mid \rho \in \{0,1\}^{\mu B}\})$ and $(\chi_S)^{\overline{m}} = (\mathcal{U}(\mathcal{H}_{n,d/n}(h))^{1 \times \overline{m}}$.

Games $G_3$ and $G_4$ only differ in the generation of $\boldsymbol{U}$. Similarly as with the corresponding IND-CPA games for CPA-PKE, it can be shown that there exists an algorithm $\mathcal{E}'$ such that

$$|\Pr(S_3) - \Pr(S_4)| \leq \overline{m} \cdot \text{Adv}^{\texttt{dGLWR}_{\text{spt}}}_{d,n,d/n,q,p}(\mathcal{A} \circ \mathcal{E}') \tag{21}$$

Table 8: IND-CPA games for Round2.KEM: Games $G_4$, $G_5$ and $G_6$

| **Game $G_4$** | **Game $G_5$** | **Game $G_6$** |
|---|---|---|
| 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. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$. | 1. $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$, $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$. |
| 2. Choose $b \xleftarrow{\$} \{0,1\}$. | 2. Choose $b \xleftarrow{\$} \{0,1\}$. | 2. Choose $b \xleftarrow{\$} \{0,1\}$. |
| 3. $m \xleftarrow{\$} \mathbb{Z}_{2^B}^{\mu}$. | 3. $m \xleftarrow{\$} \mathbb{Z}_{2^B}^{\mu}$. | 3. $m \xleftarrow{\$} \mathbb{Z}_{2^B}^{\mu}$. |
| 4. $\boldsymbol{R} \xleftarrow{\$} \left(\mathcal{H}_{n,d/n}(h)\right)^{1 \times \overline{m}}$. | 4. $\boldsymbol{R} \xleftarrow{\$} \left(\mathcal{H}_{n,d/n}(h)\right)^{1 \times \overline{m}}$. | 4. $\boldsymbol{R} \xleftarrow{\$} \left(\mathcal{H}_{n,d/n}(h)\right)^{1 \times \overline{m}}$. |
| 5. $\boldsymbol{U} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{m}}$. | 5. $\boldsymbol{U} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{m}}$. | 5. $\boldsymbol{U} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{m}}$. |
| 6. $\boldsymbol{X} = \langle \boldsymbol{B}^T \boldsymbol{R} \rangle_p$. | 6. $\boldsymbol{X} \xleftarrow{\$} \mathcal{R}_{n,p}^{\overline{n} \times \overline{m}}$. | 6. $\boldsymbol{X} \xleftarrow{\$} \mathcal{R}_{n,p}^{\overline{n} \times \overline{m}}$. |
| 7. $\boldsymbol{v} = \langle \text{RCompress}_{p \to t}(\text{Sample}_{\mu}(\boldsymbol{X})) + \frac{t}{2^B} \cdot m \rangle_t$. | 7. $\boldsymbol{v} = \langle \text{RCompress}_{p \to t}(\text{Sample}_{\mu}(\boldsymbol{X})) + \frac{t}{2^B} \cdot m \rangle_t$. | 7. $\boldsymbol{v} = \langle \text{RCompress}_{p \to t}(\text{Sample}_{\mu}(\boldsymbol{X})) + \frac{t}{2^B} \cdot m \rangle_t$. |
| 8. $K_0 = H\left(\phi(m), \texttt{bin}_1\left(c = (\boldsymbol{U}, \boldsymbol{v})\right)\right)$. | 8. $K_0 = H\left(\phi(m), \texttt{bin}_1\left(c = (\boldsymbol{U}, \boldsymbol{v})\right)\right)$. | 8. $K_0 \xleftarrow{\$} \{0,1\}^{\mu B}$. |
| 9. $K_1 \xleftarrow{\$} \{0,1\}^{\mu B}$. | 9. $K_1 \xleftarrow{\$} \{0,1\}^{\mu B}$. | 9. $K_1 \xleftarrow{\$} \{0,1\}^{\mu B}$. |
| 10. $b' = \mathcal{A}\left(pk, c = (\boldsymbol{U}, \boldsymbol{v}), K_b\right)$. | 10. $b' = \mathcal{A}\left(pk, c = (\boldsymbol{U}, \boldsymbol{v}), K_b\right)$. | 10. $b' = \mathcal{A}\left(pk, c = (\boldsymbol{U}, \boldsymbol{v}), K_b\right)$. |
| 11. Output $[(b' = b)]$. | 11. Output $[(b' = b)]$. | 11. Output $[(b' = b)]$. |

Games $G_4$ and $G_5$ only differ in the generation of $\boldsymbol{X}$. Consider the following algorithm $\mathcal{F}$.

**Algorithm $\mathcal{F}$**
**Input** $(\boldsymbol{B}, \boldsymbol{Y}) \in \mathcal{R}_{n,p}^{d/n \times \overline{n}} \times \mathcal{R}_{n,t}^{\overline{n} \times \overline{m}}$
1. Choose $\boldsymbol{A} \xleftarrow{\$} \mathcal{R}_{n,q}^{d/n \times d/n}$
2. Choose $b \xleftarrow{\$} \{0,1\}$.
3. $m \xleftarrow{\$} \mathbb{Z}_{2^B}^{\mu}$.
4. $\boldsymbol{U} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{m}}$
5. $\boldsymbol{v} = \langle \text{Sample}_{\mu}(\boldsymbol{Y}) + \frac{t}{2^B} m$
6. $K_0 = H\left(\phi(m), \texttt{bin}_1\left(c = (\boldsymbol{U}, \boldsymbol{v})\right)\right)$.
7. $K_1 \xleftarrow{\$} \{0,1\}^{\mu B}$.
8. $b' = \mathcal{A}\left(pk = (\boldsymbol{A}, \boldsymbol{B}), c = (\boldsymbol{U}, \boldsymbol{v}), K_b\right)$.
9. **Output** $[\ (b' = b)\ ]$.

If $(\boldsymbol{B}, \boldsymbol{Y}) = \left(\boldsymbol{B}, \text{RCompress}_{p \to t}(\langle \boldsymbol{B}^T \boldsymbol{R} \rangle_p)\right)$ with $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}}$ and $\boldsymbol{R} \xleftarrow{\$} \left(\mathcal{H}_{n,d/n}(h)\right)^{1 \times \overline{m}}$, then the outputs of $\mathcal{F}$ and game $G_4$ have the same distribution. If $(\boldsymbol{B}, \boldsymbol{Y}) \xleftarrow{\$} \mathcal{R}_{n,p}^{d/n \times \overline{n}} \times \mathcal{R}_{n,t}^{\overline{n} \times \overline{m}}$, then the outputs of $\mathcal{F}$ and game $G_5$ have the same distribution. This is so as $\text{RCompress}_{p \to t}$ maps uniform variables to uniform variables. Similar to the proof for the IND-CPA security for CPA-PKE, it follows that there is a reduction algorithm $\mathcal{F}'$ such that

$$| \Pr(S_4) - \Pr(S_5) | = \text{Adv}(\mathcal{A} \circ \mathcal{F}) \leq \overline{m} \cdot \text{Adv}^{\texttt{dGLWR}_{\text{spt}}}_{d,n,\overline{n},p,t}(\mathcal{A} \circ \mathcal{F}') \tag{22}$$

Games $G_5$ and $G_6$ only differ in the generation of $K_0$. An adversary that

can distinguish between these two games immediately leads to a distinguisher $\mathcal{G}$ between the output of $H$ and the uniform distribution.

$$|\Pr(S_5) - \Pr(S_6)| \leq \mathrm{Adv}^{H}(\mathcal{G}) \tag{23}$$

In Game $G_6$, the input to the adversary $\mathcal{A}$ is independent of the original bit $b$ chosen, and so

$$\Pr(S_6) = 1/2. \tag{24}$$

Combining Eqs. 17 to 24 yields the proof of Theorem 2.5.4.1. □

### 2.5.5 IND-CCA security of Round2.PKE

In this section, it is shown that Round2.PKE is IND-CCA secure. As Round2.PKE is constructed from CCA.KEM and a secure data-encapsulation mechanism as proposed by Cramer and Shoup [23], it is sufficient to show the IND-CCA security of CCA-KEM. Indeed, as stated in Theorem 2.5.5.1, when the hash functions $G$ and $H$ in Algorithms 9 and 10 are modeled as random oracles, the key-encapsulation mechanism CCA-KEM defined in Section 2.4.4 is IND-CCA secure, assuming the hardness of the decision GLWR problem with sparse-trinary secrets.

**Theorem 2.5.5.1.** *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*

$$Adv^{IND\text{-}CCA}_{CCA\text{-}KEM}(\mathcal{A}) \leq 3 \cdot Adv^{IND\text{-}CPA}_{CPA\text{-}PKE}(\mathcal{B}) + q_G \cdot \delta + \frac{2q_G + q_H + 1}{2^{\mu B}} \tag{25}$$

*when CPA-PKE and CCA-KEM both have a probability of decryption/decapsulation failure that is at most $\delta$.*

*Proof.* The proof of Theorem 2.5.5.1 proceeds via two transformation reductions due to [26]. First, Lemma 2.5.5.1 establishes that the OW-PCA [1] security of the deterministic public-key encryption scheme $\mathrm{PKE}_1$ obtained from the public-key encryption scheme PKE via transformation $T$ [26], tightly reduces to IND-CPA security of $\mathrm{PKE}_1$. This lemma is a special case of [26, 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 2.5.5.1** (Adapted from [26, Theorem 3.2])**.** *Assume PKE to be $\delta$ 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 $\mathsf{P}_{CO}$, there exists an IND-CPA adversary $\mathcal{C}$ such that*

$$Adv^{OW\text{-}PCA}_{PKE_1}(\mathcal{B}) \leq q_G \cdot \delta + \frac{2q_G + 1}{|\mathcal{M}|} + 3 \cdot Adv^{IND\text{-}CPA}_{PKE}(\mathcal{C}) \tag{26}$$

*where $\mathcal{M}$ is the message/plaintext space of the public-key encryption schemes PKE and $\mathrm{PKE}_1$.*

[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.

Next, combinination of Lemma 2.5.5.1 and the reduction in [26, Theorem 3.4] shows that the IND-CCA security of a KEM with implicit rejection that is constructed using a non-deterministic PKE (like CCA-KEM), tightly reduces to the IND-CPA security of said PKE. □

Direct application of [26, Theorem 4.6], similarly as in [16, Theorem 4.2], shows that CCA-KEM is IND-CCA secure in the quantum random oracle model. The resulting security bound however is not tight.

**Theorem 2.5.5.2.** *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*

$$Adv^{\text{IND-CCA}}_{\text{CCA-KEM}}(\mathcal{A}) \leq 4q_H\sqrt{q_D \cdot q_H \cdot \delta + q_G \cdot \sqrt{Adv^{\text{IND-CPA}}_{\text{CPA-PKE}}(\mathcal{B})}} \tag{27}$$

### 2.5.6 Hardness of Sparse-Trinary LWR

In this section, we prove the hardness of the Decision-LWR problem with sparse-trinary secrets assuming that the small modulus $p$ divides the large modulus $q$. Figure 3 provides an overview of the reductions involved in the proof of the main result, Theorem 2.5.6.1.

**Theorem 2.5.6.1.** *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*

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

*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 $\zeta$, 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 $\mathtt{dLWE}_{n,m',q,D_{\alpha\sqrt{10h}}}(\mathcal{U}(\mathcal{H}_n(h)))$ to $\mathtt{dLWR}_{n,m,q,p}(\mathcal{U}(\mathcal{H}_n(h)))$.*

*Proof.* Combination of Lemma 2.5.6.1 and Lemma 2.5.6.4 with $\alpha' = \alpha\sqrt{10h}$. □

Theorem 2.5.6.1 implies the hardness of the sparse-trinary LWR problem $\mathtt{LWR}_{\text{spt}}$ based on the hardness of the $\mathtt{LWE}$ problem with uniformly random secrets in $\mathbb{Z}_q$ and Gaussian errors.

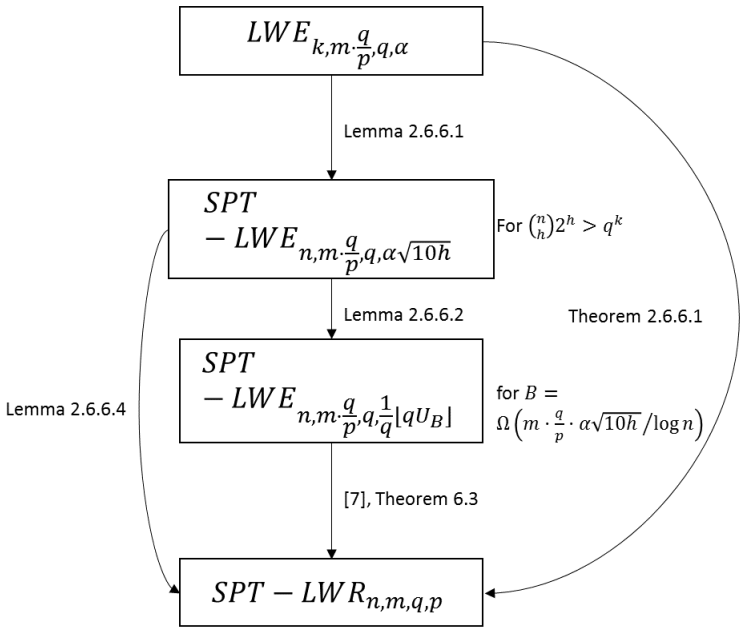


Figure 3: Summary of reductions used in Theorem 2.5.6.1. "SPT" refers to a variant of the problem in question where the secret is sparse-trinary, instead of uniform in $\mathbb{Z}_q^d$.

**Step 1: Reduction from LWE with secrets in $\mathbb{Z}_q$ and Gaussian errors to Sparse-trinary LWE:** In [22, Theorem 1], specializing [21, Theorem 4], it is shown that if $\binom{n}{h}2^h > q^{k+1}$ and $\omega > \alpha\sqrt{10h}$, then the $\mathtt{dLWE}_{n,m,q,D_\omega}(\mathcal{U}(\mathcal{H}_n(h)))$ problem is at least as hard as the $\mathtt{dLWE}_{k,m,q,D_\alpha}$ problem. More formally, generalizing [18, Theorem 4.1], the following holds.

**Lemma 2.5.6.1.** *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 $\zeta$, 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-trinary LWE to Sparse-trinary LWR:** Bai et al. provide in [7, 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 [7, 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 [7, 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 [7, Theorem 5.1] that describes a reduction from LWE with Gaussian noise and sparse trinary secrets reduces to LWR with sparse-trinary secrets. Below, we will describe such an extension. $U_B$ denotes the continuous uniform distribution in $[-B, \ldots, B]$.

**Lemma 2.5.6.2** (Adapted from [7, 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 $\texttt{LWE}_{n,m,q,D_\alpha}(\mathcal{U}(\mathcal{H}_n(h)))$ to $\texttt{LWE}_{n,m,q,\phi}(\mathcal{U}(\mathcal{H}_n(h)))$ with $\phi = \frac{1}{q}\lfloor qU_B \rceil$.*

*Proof.* The reduction proceeds similar to that of [7, Theorem 5.1], relying on five steps. Steps 1, 3, 4 below proceed exactly as in [7, Theorem 5.1]. For steps 2 and 5, we mention our adaptations in order to prove the reduction for the case of sparse-trinary secrets, and the resulting conditions. We omit details for brevity.

1. A reduction from $\texttt{dLWE}_{n,m,q,D_\alpha}$ to $\texttt{dLWE}_{n,m,q,\psi}$, with $\psi = D_\alpha + U_B$.
2. A reduction from $\texttt{dLWE}_{n,m,q,\psi}$ to $\texttt{sLWE}_{n,m,q,\psi}$. We adapt the corresponding step in [7, 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 $\texttt{sLWE}_{n,m,q,\psi}$ to $\texttt{sLWE}_{n,m,q,U_B}$.
4. A reduction from $\texttt{sLWE}_{n,m,q,U_B}$ to $\texttt{sLWE}_{n,m,q,\phi}$, 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 [7, Theorem 5.1] cannot be applied. Instead, we extend an argument due to Regev (see, e.g, [37]) to prove the search-to-decision reduction, which requires that $Bq$ is an integer. We first state an easy lemma.

   **Lemma 2.5.6.3.** *Let $a > 1$, and let $\phi$ 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 $\phi$ 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 $\phi$, and outputting $(\mathbf{a}, (\mathbf{a},\mathbf{s}) + e)$ where the additions are modulo $q$. If $qB$ is integer, then a distinguisher for $\mathtt{dLWE}_{n,m,q,\phi}(D_s)$ can be used to construct a solver for $\mathtt{sLWE}_{n,m,q,\phi}(D_s)$ for any secret distribution $D_s$ supported on $\{-1,0,1\}^n$, where $\phi$ is the discrete noise $\frac{1}{q}\lfloor qU_B \rfloor$. Note that if $Bq$ is integer, the noise $\phi$ 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 $\mathtt{dLWE}_{n,m,q,D_\alpha}$ to $\mathtt{dLWR}_{n,m,q,p}$, for the sparse-trinary secret distribution.

**Lemma 2.5.6.4.** *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 $\mathtt{dLWE}_{n,m',q,D_{\alpha'}}$ to $\mathtt{dLWR}_{n,m,q,p}$, both defined for the sparse-trinary secret distribution.*

*Proof.* Let $B = q/2p$. The reduction has two steps:

1. A reduction from $\mathtt{dLWE}_{n,m',q,D_{\alpha'}}$ to $\mathtt{dLWE}_{n,m',q,\phi}$, where $B = \Omega(m'\alpha'/\log n)$. due to Lemma 2.5.6.2.

2. A reduction from $\mathtt{dLWE}_{n,m',q,\phi}$ to $\mathtt{dLWR}_{n,m,q,p}$, due to [7, 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 2.5.6.2 indeed is applicable. □

Note that the conditions imposed by Lemma ]2.5.6.2 imply that $1/\alpha$ must at least grow linearly in $n$. This is a common bottleneck in all known LWE to LWR reductions [7, 13, 8].

## 2.6 Analysis with respect to known attacks

In this section, we analyze the concrete security of Round2. We begin by considering attacks using lattice basis reduction in Sections 2.6.1, 2.6.2 and 2.6.3, followed by specialized attacks that exploit sparse-trinary secrets used in Round2 in Sections 2.6.4 and 2.6.5. Next, we consider the issue of biases in public-keys of Round2 in Section 2.6.6. Finally, we consider precomputation and back-door attacks against the Round2 GLWR public parameter $\boldsymbol{A}$.

### 2.6.1 Lattice-based attacks

We consider lattice-reduction based attacks, namely, the *primal* or decoding attack [6] and the *dual* or distinguishing attack [1], and how they can be adapted to exploit the shortness of secrets in our schemes. We begin by detailing how an attack on Round2 can be formulated as a lattice-reduction based attack on the LWR problem. We then analyze the concrete security of Round2 against the primal attack in order to estimate secure parameters, in Section 2.6.2. We do the same for the dual attack in Section 2.6.3.

The attacker can use the public keys $\boldsymbol{B} = \mathrm{RCompress}_{q\to p}(\langle \boldsymbol{AS}\rangle_q)$ 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. For the non-ring case, $\boldsymbol{B} \in \mathbb{Z}_p^{d\times \overline{n}}$. Note also that since $q|p$ in this case, $\boldsymbol{B} = \langle \lfloor \frac{p}{q}\langle \boldsymbol{AS}\rangle_q \rceil \rangle_p$. Let $1 \le i \le d$ and $1 \le j \le \overline{n}$. 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

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

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 $\lambda$, we infer that

$$\frac{q}{p}\boldsymbol{B}_{i,j} \equiv \boldsymbol{a}_i^T \boldsymbol{s}_j + \frac{q}{p}e_{i,j} \pmod q. \tag{28}$$

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 $q$.

### 2.6.2 Primal Attack

In (28), 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 $\boldsymbol{e} \in (-\frac{q}{2p}, \frac{q}{2p}]^m$

$$\boldsymbol{b} \equiv \boldsymbol{A}_m \boldsymbol{s} + \boldsymbol{e} \pmod q \tag{29}$$

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

$$\Lambda = \{\boldsymbol{x} \in \mathbb{Z}^{d+m+1} : (\boldsymbol{A}_m | \boldsymbol{I}_m | -\boldsymbol{b})\boldsymbol{x} = \boldsymbol{0} \pmod q\} \tag{30}$$

of dimension $d' = d + m + 1$ and volume $q^m$ [15, 2]. The attacker then searches for a short vector in $\Lambda$ which hopefully equals $\boldsymbol{v}$, thus enabling him to recover the secret $\boldsymbol{s}$.

**Lattice Rescaling:** The lattice vector $\boldsymbol{v} = (\boldsymbol{s}, \boldsymbol{e}, 1)$ is unbalanced in that $\|\boldsymbol{s}\| \ll \|\boldsymbol{e}\|$. For exploiting this fact, a rescaling technique originally due to [6], and analyzed further in [22] and [1] is applied. Multiplying the first $d$ columns of $\Lambda$'s basis (see Eq. 30) with an appropriate scaling factor $\omega$ yields the following weighted or rescaled lattice,

$$\Lambda_\omega = \{\boldsymbol{x} \in \mathbb{Z}^{d+m+1} : \big((\omega \cdot \boldsymbol{A}_m^T) | \boldsymbol{I}_m | -\boldsymbol{b}\big)\boldsymbol{x} = \boldsymbol{0} \text{ (mod q)}\} \tag{31}$$

in which the attacker then searches for the shortest 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 due to a common heuristic. We explain later in this section how to choose an appropriate value for $\omega$ 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 [2] assuming that BKZ [20, 39] 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 $\Lambda$ 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 $\Lambda$ [2, Sec. 6.3],[15]. The condition that must be satisfied for the primal attack to succeed is therefore:

$$\begin{aligned} & \|\widetilde{\boldsymbol{v}}_b\| < \|\widetilde{\boldsymbol{b}}_{d'-b}\| \\ \text{i.e.,} \quad & \|\widetilde{\boldsymbol{v}}_b\| < \delta^{2b-d'-1} \cdot (\mathrm{Vol}(\Lambda))^{\frac{1}{d'}} \\ \text{where,} \quad & \delta = ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi \mathbf{e}})^{\frac{1}{2(b-1)}} \end{aligned} \tag{32}$$

The attack success condition (32) 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:

$$\begin{aligned} & \sqrt{(\omega^2 \cdot h + \sigma'^2 m) \cdot \frac{b}{d+m}} \geq \delta^{2b-d'-1} \cdot (q^m \omega^d)^{\frac{1}{d'}} \\ \text{where,} \quad & \delta = ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi \mathbf{e}})^{\frac{1}{2(b-1)}}, \\ & \sigma' = (q/2\sqrt{3}p), \\ \text{and} \quad & d' = d + m + 1. \end{aligned} \tag{33}$$

For finding an appropriate value for $\omega$, we rewrite (33) as

$$\delta^{2b-d'-1}b^{-1/2} \leq \sqrt{\omega^2 h + m\sigma'^2 \cdot \frac{1}{m+d}}\omega^{-d/d'}q^{-(d-d'-1)/d}. \tag{34}$$

Given $d, m, h$ and $\sigma'$, the attacker obtains the least stringent condition on the block size $b$ by maximizing the right hand side 34 over $\omega$, or equivalently, by maximizing

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

By differentiating with respect to $\omega$, we find that the optimizing value for $\omega$ satisfies

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

### 2.6.3 Dual Attack

The dual attack against LWE attempts to find a short vector $(\boldsymbol{v}, \boldsymbol{w}) \in \mathbb{Z}^m \times \mathbb{Z}^d$ in the dual lattice

$$\Lambda^* = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}^m \times \mathbb{Z}^d : \boldsymbol{A}_m^T\boldsymbol{x} = \boldsymbol{y} \pmod q\}. \tag{35}$$

This vector is used to construct a distinguisher using $z = \{\boldsymbol{v}^T\boldsymbol{b}\}_q$. If $\boldsymbol{b} = \boldsymbol{A}_m\boldsymbol{s} + \boldsymbol{e} \pmod q$, then $z = \{\boldsymbol{v}^T\boldsymbol{b})\}_q \equiv \{\boldsymbol{w}^T\boldsymbol{s} + \boldsymbol{v}^T\boldsymbol{e}\}_q$, and $z$ is therefore small.

For an LWR distribution with uniform rounding error $\boldsymbol{e}'$ and corresponding variance $\sigma'^2 = q^2/12p^2$, the distinguisher checks whether $z = \{\boldsymbol{v}^T\boldsymbol{b}\}_q$ is small. For a non-LWR distribution, $z$ is distributed uniformly modulo $q$. For an LWR distribution, $z$'s distribution approaches a Gaussian distribution of zero mean and variance $\|\boldsymbol{v}\|^2 \cdot \sigma'^2$ as the lengths of the vectors $\boldsymbol{v}$ and $\boldsymbol{e}'$ increase, due to the Central limit theorem. The maximal statistical distance between this Gaussian distribution and the uniform distribution modulo $q$ is bounded by $\epsilon \approx (1/\sqrt{2})\exp(-2\pi^2(\|\boldsymbol{v}\| \cdot \sigma'/q)^2)$, a more detailed derivation of this result can be found in [12, Appendix B]. The attacker uses the BKZ algorithm with block-size $b$ that outputs a short vector of length $\delta^{d'-1} \cdot \mathrm{Vol}(\Lambda^*)^{1/d'}$, where $d' = m+d$ is the dimension of the dual lattice $\Lambda^*$, and its volume is $\mathrm{Vol}(\Lambda^*) = q^d$.

As the key is hashed, a small advantage $\epsilon$ is not sufficient. As explained in [2], assuming BKZ with block size $b$, the attack must be repeated at least $R = \max(1, 1/2^{0.2075b} \cdot \epsilon^2)$ times. Consider an LWR distribution that is generated from an LWR problem instance of dimension $d$, large modulus $q$, rounding modulus $p$. The cost of using the dual attack to distinguish such an LWR distribution from uniform, employing BKZ with block size $b$ and root-Hermite

factor $\delta$, using $m$ samples is:

$$\begin{aligned}
&\text{Cost}_{\text{Dual attack}} = (b \cdot 2^{cb}) \cdot \max(1, 1/(\epsilon^2 \cdot 2^{0.2075 \cdot b})), \\
\text{where,} \quad &\epsilon = \frac{1}{\sqrt{2}} \cdot \mathbf{e}^{-2\pi^2 \left(\frac{\|\boldsymbol{v}\| \cdot \sigma'}{q}\right)^2}, \\
&\|\boldsymbol{v}\| = \delta^{m+d-1} \cdot \left(\frac{q}{\sigma' \cdot \sqrt{d/h}}\right)^{1/(m+d)}, \\
&\delta = ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi \mathbf{e}})^{\frac{1}{2(b-1)}} \\
\text{and} \quad &\sigma' = (q/2\sqrt{3}p).
\end{aligned} \tag{36}$$

The first term in the cost above, i.e., $(b \cdot 2^{cb})$ is the cost of running BKZ lattice reduction with block-size $b$, where $c$ is the BKZ sieving exponent. Finally, note that the cost in Eq. 36 also accounts for the fact that the dual attack can be adapted against the sparse-ternary LWR problem by rescaling the dual lattice using an appropriate scaling factor $\omega = \sigma\sqrt{m/h}$ [1] (so as to equalize the contributions of both parts $\boldsymbol{w}^T\boldsymbol{s}$ and $\boldsymbol{v}^T\boldsymbol{e}$ in $z$), resulting in the following rescaled dual lattice:

$$\Lambda^*_\omega = \{(\boldsymbol{x}, \boldsymbol{y}/\omega) \in \mathbb{Z}^m \times \left(\frac{1}{\omega} \cdot \mathbb{Z}^d\right) : \boldsymbol{A}_m^T\boldsymbol{x} = \boldsymbol{y} \pmod q\} \tag{37}$$

This scales the volume of the dual lattice from $q^d$ to $(q/\omega)^d$, which correspondingly scales the norm $\|\boldsymbol{v}\|$ of the short vector $\boldsymbol{v}$ and hence the distinguishing advantage.

### 2.6.4 Hybrid Attack

In this section, we consider a hybrid lattice reduction and meet-in-the-middle attack (henceforth called *hybrid attack*) originally due to [28] that targeted the NTRU [27] cryptosystem. We first describe the hybrid attack, using notation similar to that of [41], 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 $\Lambda'$ of the form

$$\boldsymbol{B}' = \begin{pmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{I}_r \end{pmatrix} \tag{38}$$

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 $\Lambda'$. 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 $\Lambda'$, there exists a $\boldsymbol{x} \in \mathbb{Z}^{d+m+1-r}$ such that

$$\boldsymbol{v} = \begin{pmatrix} \boldsymbol{v}_l \\ \boldsymbol{v}_g \end{pmatrix} = \boldsymbol{B}' \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{v}_g \end{pmatrix} = \begin{pmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{I}_r \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{v}_g \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}\boldsymbol{x} + \boldsymbol{C}\boldsymbol{v}_g \\ \boldsymbol{v}_g \end{pmatrix} \tag{39}$$

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 [41], 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 [5]. 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 [28] 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 trinary. 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 trinary 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 $\omega$. Calculated similarly to § 2.6.1 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 $\omega = \frac{q^2}{12p^2} \cdot \sqrt{\frac{d}{h}}$. This then scales up the volume of the $(d-r+m+1$ dimensional) lattice $\Lambda$ generated by $\boldsymbol{B}$ by a factor $\omega^{d-r}$.

We analyze the hybrid attack similarly as in [27]. 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 [28, 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 [27] We use the Geometric Series Assumption to approximate the norms of these vectors in terms of the Hermite constant $\delta$. The cost for obtaining a reduced basis with Hermite constant $\delta$ 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 for the sieving constant $c$ we take the value $\log_2 \sqrt{13/9} \approx 0.265$, [30, Sec. 14.2.10] corresponding to the quantum case. 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 [27], 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 [41] 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 $\delta$. 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.

### 2.6.5 Attacks against Sparse Secrets

In Sections 2.6.2 and 2.6.3, we considered attacks [6, 1, 22] 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 2.6.4, and another one due to Albrecht *et al* [1]. 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 [1] 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

$$\min\{b \cdot 2^{cb} \mid b \in \mathbb{N}, \text{ there exists } m \in \mathbb{N} \text{ such that } m \leq d \text{ and (40) is satisfied.}\}$$

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

$$\begin{aligned} \text{where,} \quad \delta &= ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi \mathbf{e}})^{\frac{1}{2(b-1)}}, \\ \omega &= \sigma' \cdot \sqrt{(d-k)/h}, \\ \sigma' &= (q/2\sqrt{3}p), \\ \text{and} \quad d' &= m + d - k + 1. \end{aligned} \tag{40}$$

The term $b \cdot 2^{cb}$ represents the cost of running BKZ lattice reduction with block-size $b$. The attack runs on a LWE problem of dimension $d-k$ with $m \leq d$ samples. Condition 40, which is essentially Condition 32, 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. 36, 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 2.6.4), 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 [1]).

Furthermore, to ensure that an exhaustive, brute-force search of each secret-key vector in the secret-key using Grover's quantum search algorithm [25] has a cost of at least $\lambda$ (in bits), the chosen Hamming weight should satisfy:

$$\sqrt{\binom{d}{h} \cdot 2^h} > 2^\lambda \tag{41}$$

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 $0.1d$.

### 2.6.6 Biases in Computed Keys

Round2 public keys are unbiased, since by definition (see Eq. 1 in Section 2.2) the rounding function $\text{RCompress}_{q \to p}()$ does not have a bias.

### 2.6.7 Pre-computation and Back-door Attacks

A pre-computation attack can happen if the GLWR public parameter $\boldsymbol{A}$ is fixed and an attacker performs lattice reduction on it over a long period of time. A back-door attack can happen if there is any public value, e.g., $\boldsymbol{A}$ is deliberately chosen so as to result in a lattice with weak security.

All the definitions of $f_n^\mu(\sigma)$ in Round2 (see Section 2.4.1) prevent both types of attacks: In the first definition of $f_n^\mu(\sigma)$, $f_{n=1}^0(\sigma)$, a new $\boldsymbol{A}$ is derived by means

of a DBRG from a randomly generated seed in each protocol instantiation. This is similar to [15] and prevents both pre-computation attacks and back-doors. In the second definition of $f_n(\sigma)$, $f^1_{n=1}(\sigma)$, $\boldsymbol{A}$ is derived from a fixed long-term matrix $\boldsymbol{A}_{\text{master}}$ of dimension $d^2$. This is done by applying a random, fresh permutation that is chosen by the initiator of the protocol at the start of each protocol exchange. This prevents any pre-computation attacks since the possible number of permuted $\boldsymbol{A}$ obtained in this way equals $n^n$. This is as done in [12]. Back-doors are avoided since $\boldsymbol{A}_{\text{master}}$ is derived by means of a pseudo-random function from a randomly generated seed. We note that in both $f^0_{n=1}(\sigma)$ and $f^1_{n=1}(\sigma)$, entries of both $\boldsymbol{A}_{\text{master}}$ and the resulting $\boldsymbol{A}$ cannot be differentiated from uniform, hence the results in Section 2.5.6 hold.

In the third definition of $f_n(\sigma)$, $f^2_{n=1}(\sigma)$ obtains $\boldsymbol{A}$ from a set containing $q$ elements. This set, represented as a vector $\boldsymbol{a}_{\text{master}}$, is randomly generated by means of a DBRG from a seed determined by the initiator in each protocol interaction. Furthermore, each row in $\boldsymbol{A}$ is obtained from this vector by means of a random permutation that is also determined by the initiator and is specific to each protocol interaction. Since only a few elements need to be generated and kept in memory, $f^2_{n=1}(\sigma)$ is efficient. Pre-computation and back-door attacks are avoided since the seed that determines $\boldsymbol{A}$ is new in each protocol interaction. Furthermore, this approach destroys any structure in the resulting $\boldsymbol{A}$ (as can be found in circulant or anti-circulant matrices for ideal lattices, for example) since it contains many more elements. This results clear from the following analysis. Suppose $a = \langle a_0, a_1, a_2, \ldots, a_{n-1}\rangle$ and $b = \langle b_0, b_1, \ldots, b_{k-1}, a_0, a_1, a_2, \ldots, a_{n-k-1}\rangle$ are two rows of $\boldsymbol{A}$. They share $n-1-k$ elements due to our rotation strategy. Then, define $a(x) \coloneqq a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ and $b(x) \coloneqq b_0 + b_1x + \cdots + b_{k-1}x^{k-1} + a_0x^k + a_1x^{k+1} + \cdots + a_{n-k-1}x^{n-1}$. We have $b(x) = a(x)x^k + (a_{n-k}+b_0) + (a_{n-k+1}+b_1)x + \cdots + (a_{n-1}+b_{k-1})x^{k-1} \bmod x^n+1$. Effectively, that is, each row can be seen as using the $x^n+1$ ring with a random shifting (due to $k$) and additional random noises, those $(a_{n-k}+b_0)+(a_{n-k+1}+b_1)x+\cdots+(a_{n-1}+b_{k-1})x^{k-1}$. From this point of view, they are at least as secure as the pure ring version of Round2. Since the length of $\boldsymbol{a}_{\text{master}}$ is smaller than $d^2$, the entries of the resulting $\boldsymbol{A}$ can be differentiated from uniform. The security of Round2 can therefore not be based on the results in Section 2.5.6 for this definition of $f_n(\sigma)$.

In the fourth definition of $f_n(\sigma)$, $f^3_{n=d}(\sigma)$, we consider that $\boldsymbol{A}$ is derived from a vector $\boldsymbol{a}_{\text{master}}$ of length $d$ that is specific to each protocol interaction. This approach is as in [2]. As in previous cases, pre-computation and back-door attacks are avoided since the resulting $\boldsymbol{A}$ is new in each protocol interaction.

In Round2, the default definitions of $f^\tau_n(\sigma)$ are $f^2_{n=1}(\sigma)$ and $f^3_{n=d}(\sigma)$ for the non-ring and ring cases due to their efficiency. Both of them are implemented in terms of a master vector $\boldsymbol{a}_{\text{master}}$ and a permutation $\Pi$ to enable a unified implementation.

## 2.7 Probability of Incorrect Decryption

The schemes in Round 2 need not always result in correct retrieval of the encrypted message or the encapsulated key. In this section, the decryption failure behavior of CPA-PKE is analyzed.

By definition of the function $\mathrm{RCompress}_{q\to p}()$

$$\boldsymbol{B} = \left\langle \left\lfloor \frac{(p/g)\langle \boldsymbol{AS}\rangle_q + \boldsymbol{E}_B}{q/g} \right\rceil \right\rangle_p = \left\langle \frac{(p/g)\boldsymbol{AS} + \boldsymbol{E}_B}{q/g} - \frac{\boldsymbol{I}_B}{q/g} \right\rangle_p,$$

where $g = \gcd(p, q)$, the components of $\boldsymbol{E}_B$ are drawn independently and uniformly from $[-\frac{p}{2g}, \frac{p}{2g}) \cap \mathbb{Z}$ and $\boldsymbol{I}_B/(q/g)$ is the effect of rounding, with each component of $\boldsymbol{I}_B$ in the set $[-\frac{q}{2g}, \frac{q}{2g}) \cap \mathbf{Z}$.

By the same reasoning, it holds that

$$\boldsymbol{U} = \left\langle \frac{(p/g)\boldsymbol{A}^T\boldsymbol{R} + \boldsymbol{E}_U}{q/g} - \frac{\boldsymbol{I}_U}{q/g} \right\rangle_p.$$

Since $t$ and $p$ are powers of 2, $t$ divides $p$,

$$\boldsymbol{v} = \left\langle \frac{\mathrm{Sample}_\mu(\boldsymbol{X}) + (p/2^B)m - \boldsymbol{I}_v}{p/t} \right\rangle_t$$

and

$$\mathrm{Decompress}_{t\to p}(\boldsymbol{v}) = \langle (p/t)\boldsymbol{v}\rangle_p = \langle \mathrm{Sample}_\mu(\boldsymbol{X}) + \frac{p}{2^B}m - \boldsymbol{I}_v\rangle_p,$$

with the components of $\boldsymbol{I}_v$ in the set $[-\frac{p}{2t}, \frac{p}{2t}) \cap \mathbb{Z}$. The decrypted message $\hat{m} = \mathrm{RCompress}_{p\to 2^B}(\boldsymbol{M})$ with, as $\boldsymbol{X} = \langle \boldsymbol{B}^T\boldsymbol{R}\rangle_p$

$$\begin{aligned}\boldsymbol{M} &= \left\langle \mathrm{Decompress}_{t\to p}(\boldsymbol{v}) - \mathrm{Sample}_\mu(\boldsymbol{S}^T\boldsymbol{U}) \right\rangle_p \\ &= \left\langle \frac{p}{2^B}m - \boldsymbol{I}_v + \mathrm{Sample}_\mu\left( \frac{(\boldsymbol{E}_B - \boldsymbol{I}_B)^T\boldsymbol{R} - \boldsymbol{S}^T(\boldsymbol{E}_U - \boldsymbol{I}_U)}{q/g} \right) \right\rangle_p.\end{aligned}$$

Decryption succeeds, i.e., $\hat{m} = m$, if and only if all components of $\{\boldsymbol{M} - \frac{p}{2^B}m\}_p$ are in the set $[-\frac{p}{2^{B+1}}, \frac{p}{2^{B+1}}) \cap \mathbb{Z}$.

Now consider the probability distribution of a component of $\{\boldsymbol{M} - \frac{p}{2^B}m\}_p$. From experiments, we find that is close to the distribution of a component of

$$\left\{ -\boldsymbol{i}_v + \mathrm{Sample}_\mu\left( \left\lfloor \frac{(\boldsymbol{e}_B - \boldsymbol{i}_B)^T\boldsymbol{r} - \boldsymbol{s}^T(\boldsymbol{e}_U - \boldsymbol{i}_U)}{q/g} \right\rceil \right) \right\}_p, \tag{42}$$

where all variables are independently and uniformly drawn from their respective sets. Therefore we shall use the latter distribution for obtaining an upper bound

(the Chernoff bound) for, and an approximation to the probability of incorrect decryption.

We shall distinguish four cases, depending on whether $q$ is a power of two or a prime, and depending on whether a polynomial ring is used or not.

In the non-ring case, the matrices $\boldsymbol{s}$ and $\boldsymbol{r}$ have columns with components from $\{-1, 0, 1\}$ with Hamming weight $h$, so each component of $-\boldsymbol{i}_B^T\boldsymbol{r} + \boldsymbol{s}^T\boldsymbol{i}_U$ is the sum of $2h$ independent terms from the set $[-\frac{q}{2g}, \frac{q}{2g}) \cap \mathbb{Z}$, each multiplied by a sign. In the ring case, we have $n+1$ prime and calculate modulo $\Phi_{n+1}(x) = x^n + x^{n-1} + \cdots + x + 1$. It can easily be shown [40] that for all polynomials $a(x)$ of $b(x)$ of degree at most $n-1$, each coefficient of $\langle a(x)b(x)\rangle_{\Phi_{n+1}(x)}$ is the sum of at most $2n-1$ products of the form $a_i b_j$, and that each $a_i$ occurs in at most two such products. As a consequence, each component of $\boldsymbol{i}_B^T\boldsymbol{r}$ is the sum of up to $2h$ independent terms from the set $[-\frac{q}{2g}, \frac{q}{2g}) \cap \mathbb{Z}$, each multiplied by a sign. The same holds for each component of $\boldsymbol{s}^T\boldsymbol{i}_U$, so in the ring case $-\boldsymbol{i}_B^T\boldsymbol{r} + \boldsymbol{s}^T\boldsymbol{i}_U$ is the sum of up to $4h$ independent terms from the set $[-\frac{q}{2g}, \frac{q}{2g}) \cap \mathbb{Z}$, each multiplied by a sign.

A similar reasoning for a component of $\boldsymbol{e}_B^T\boldsymbol{r} - \boldsymbol{s}^T\boldsymbol{e}_U$ shows that in the non-ring case it is equal to $2h$ independent terms from the set $[-\frac{p}{2g}, \frac{p}{2g}) \cap \mathbb{Z}$, each multiplied by a sign, and in the ring case it is equal to up to $4h$ such independent terms. Note that when $q$ is a power of two, $g = p$ the above set equals $\{0\}$ and these terms are absent.

The distribution of a component of $\{\boldsymbol{M} - \frac{p}{2^B}m\}_p$ thus is well approximated by that of $\{\lfloor w \rceil\}_p$, where

$$w = \frac{p}{t}U + \sum_{i=1}^{2c_\mathrm{r}h} u_i + c_\mathrm{p}\frac{p}{q}\sum_{j=1}^{2c_\mathrm{r}h} v_j, \tag{43}$$

and where $U$, the $u_i$ and the $v_j$ are i.i.d. uniform random numbers from $(-\frac{1}{2}, \frac{1}{2})$, $c_\mathrm{r} = 1$ in the non-ring case and 2 otherwise, and $c_\mathrm{p} = 1$ if $q$ is prime and 0 if $q$ is a power of two. It follows that the probability that a component of $\{\boldsymbol{M} - \frac{p}{2^B}m\}_p$ is not in the set $[-\frac{p}{2^{B+1}}, \frac{p}{2^{B+1}}) \cap \mathbb{Z}$ is close to $\Pr[|w| > p/2^{B+1}]$. If

$$\frac{p}{2t} + c_r h + c_p\frac{p}{q}c_r h < \frac{p}{2^{B+1}}$$

this probability is zero. Otherwise, since $w$ is the sum of independent random variables, this probability can be bounded by the Chernoff method, yielding the upper bound

$$2\min_{\alpha>0}\left(F\left(\frac{p\alpha}{2t}\right)\left[F\left(\frac{\alpha}{2}\right)\right]^{2c_\mathrm{r}h}\left[F\left(\frac{p\alpha}{2q}\right)\right]^{2c_\mathrm{p}c_\mathrm{r}h} e^{-\alpha p/2^{B+1}}\right), \tag{44}$$

where $F(x) = \sinh(x)/x$. This bound can be evaluated numerically.

Note that $w$ is the sum of one zero-mean random variable with relatively large variance $p^2/12t^2$ and many zero-mean random variables with smaller variances

$1/12$ or $p^2/12q^2$. For large $h$, the small ones add up to a zero-mean Gaussian with variance $v = \left(2c_{\mathrm{r}}h + 2(p/q)^2 c_{\mathrm{p}}c_{\mathrm{r}}\right)/12$.

For the parameter sets in which $q$ and $p$ are powers of two, the distribution of the random variable represented by Eq. 42 was computed explicitly by repeated convolution of probability distributions of the individual variables. The per-component decryption error probability is at most the probability that the norm of the random variable exceeds $p/2^{B+1}$. For prime $q$, explicit computation was too resource-intensive because the probability distributions have a large support. In this case, Eq. 44 was used as upper bound to the per-component decryption error probability. In both cases, by a union-bound argument, the probability that the message is not decrypted correctly is at most $\mu$ times the per-component decryption error probability.

## 2.8 Implementation considerations

The design of Round2 makes it possible to have a unique implementation for both RLWR and LWR. The optimized implementation supports all NIST security levels of uRound2 for both $n = 1$ and $n = d$ .

The parameters of nRound2 allow the use of NTT for performing polynomial multiplications. Because $n$ and $q$ are different for different NIST security levels with the nRound2 parameters, different optimizations would be required for different NIST security levels. Such optimizations have not been carried out.

### 2.8.1 Polynomial multiplication

Polynomial multiplication modulo $x^n - 1$ is equivalent to a convolution. However, the reduction polynomial used in Round2 is not $x^n - 1$. We define the following algorithm to enable polynomial multiplication modulo $\Phi_{n+1}(x)$ by polynomial multiplication modulo $x^{n+1} - 1$ preceded and followed by simple transformations:

- Compute $a'(x) = (x - 1)a(x)$
- Compute $c'(x) = a'(x)b(x) \bmod x^{n+1} - 1$
- Compute $c(x) = c'(x)/(x - 1)$

We have that $c(x) \equiv a(x)b(x) \pmod{\Phi_{n+1}(x)}$. The coefficients of $c'(x) = a'(x)b(x) \bmod x^{n+1} - 1$ can be computed as

$$c'_i = \sum_{j=0}^{n} b_j \cdot a'_{\langle i-j \rangle_{n+1}} \text{ for } 0 \le i \le n, \text{ where } b_n = 0. \tag{45}$$

### 2.8.2 A common multiplication

The polynomial multiplication expressed in (45) can be implemented as the matrix multiplication $\boldsymbol{c'} = \boldsymbol{A'b}$ where $\boldsymbol{A'}_{i,j} = a'_{\langle i-j \rangle_{n+1}}$ and $\boldsymbol{b} = (b_0, \ldots, b_n)^T$.

With the addition of the simple transformations in the case $n = d$, all multiplications in the algorithm, for both $n = 1$ and $n = d$, can be considered as matrix multiplications, and therefore can use the same implementation.

### 2.8.3 Generation of $\boldsymbol{A}$

Using the definition of $f_n^\tau$ from Section 2.4.1, $\boldsymbol{A}$ can be represented as a vector $a_{master}$ of length $L$ and offsets $o_0, o_1 \ldots, o_{d-1}$ in $\{0, 1, \ldots, d-1\}$.

- For $f_{n=1}^0$, all offsets are equal to zero and $L = d^2$.
- For $f_{n=1}^1$, we have that $L = d^2$. Entry $i, j$ of the matrix $\boldsymbol{A}$ satisfies $a_{i,j} = a^{\text{fixed}}_{i,(j+o_i) \pmod d}$ for $0 \leq i, j \leq d-1$. In order to avoid addition modulo $d$, we use the $d \times 2d$ matrix $\boldsymbol{a}^*$ defined as $a^*_{i,j} = a^*_{i,j+d} = a^{\text{fixed}}_{i,j}$ for $0 \leq i, j \leq d-1$. Then $a_{i,j} = a^*_{i,j+o_i}$.
- For $f_{n=1}^2$, we take $L = q$. We write $a_{master}[i] = DRGB(\sigma_0)[i]$ for $0 \leq i \leq L-1$. We then have $a_{i,j} = a_{master}[(j + o_i) \pmod L]$ for $0 \leq i, j \leq d-1$. In order to avoid additions modulo $L$, we use the vector $a^*$ of length $L+d$ defined as $a^*[i] = a_{master}[i]$ for $0 \leq i \leq d-1$, and $a^*[i + L] = a_{master}[i]$ for $0 \leq i \leq d-1$. Then $a_{i,j} = a^*[i + j]$.
- For $f_{n=d}^3$, we have that $L = d$ and there are no offsets.

### 2.8.4 Secret key

The columns of the secret keys $\boldsymbol{S}$ and $\boldsymbol{R}$ are trinay vectors of Hamming weight $h$. The vectors are generated as described in Section 3.6 of [11]. Radix sort is used as constant-time sorting algorithm.

The optimized implementation relies on an index representation of the secret keys. That is, a trinary vector of Hamming weight $h$ is described as a vector of length $h$. The first $h/2$ entries of the vector contain the positions of the $+1$ elements; the next $h/2$ elements contain the positions of the $-1$ elements. This allows us to only loop over the non-zero elements, adding the ones in the first half and subtracting the second half.

While this is implementation is constant-time (given $h$), it is not protected against cache attacks, as the only elements accessed are the ones for which the secret key is different from 0. It would be easy to reconstruct the positions of the non-zero entries of the secret key by observing which elements of $\boldsymbol{A}$ were accessed. Other implementations choices, e.g. straightforward vector-matrix multiplication, can be adopted to provide protection against this type of attacks.

### 2.8.5 Choice of auxiliary functions

The proposed algorithms make use of several auxiliary functions, specifically a hash and a deterministic random bit generator (DRBG). SHA3-512 is used as hash function. When several outputs are required from a hash (see line 2, Algorithm 9), the hash is iterated.

The DRBG used is based on the one provided by NIST for KAT generation and follows the recommendations in [33].

### 2.8.6 DEM in Round2.PKE

Round2.PKE can use any DEM. The implementation of Round2.PKE uses a DEM based on AES256 in GCM mode as recommended in [32]. The OpenSSL implementation of AES256 is used. Using this DEM, the ciphertext of Round2.PKE needs an additional overhead of 28 bytes: 12 bytes for the initialization vector, and 16 for the authentication tag.

For NIST security levels 1, 2, and 3, the key is padded with zeros before being used in the DEM. For NIST security level 4, the exchanged key is truncated. In all cases, the key is hashed before being used in AES256-GCM.

### 2.8.7 Use in network protocols

Different applications have different security and bandwidth requirements. The implementation of Round 2 supports all those different requirements without the need of recompiling.

In a network protocol, two parties can initiate the communication using a standard security level. After agreeing on a different security level, they simply can use the library with the new set of parameters. The choice of $n = 1$ or $n = d$ is included in this; that is, the same library can be used for a set of parameters based on LWR and a set of parameters based on RLWR.

### 2.8.8 Definition of Sample$_\mu$

The function Sample$_\mu$ picks up $\mu$ entries from a matrix. In our parameter sets, if $n = d$, then $\overline{n} = \overline{m} = 1$, and Sample$_\mu$ picks up the $\mu$ coefficients of highest order. If $n = 1$, Sample$_\mu$ picks up the last $\mu$ entries of the vector obtained by serializing the matrix row by row.

## 2.9 Round2: Configurations, Parameters and Performance

Round2.KEM and Round2.PKE can be configured to instantiate different underlying problems depending on the input configuration parameter $n$. As described in detail later, the security level depends on the input value $d$ and also the choice of $q$ and $p$. Similarly, some optimizations will also only be feasible for some choices of $q$ and $p$.

### 2.9.1 Configurations of Round2

In our submission, we consider two overall parameter sets for Round2:

**Parameter set *uRound2*: Unified Round2** The goal of this parameter set is to allow for a unified and efficient implementation of Round2, independently of the fact that it instantiates LWR or RLWR.

Thus, this parameter set allows $n = 1$ with $d$ being arbitrary, or $n = d$ where $n + 1$ is prime. We designate the first case of uRound2 as the parameter set uRound2$_{n=1}$, and the second uRound2$_{n=d}$. A further restriction is that $\Phi_{n+1}(x)$ must irreducible modulo two. Finally, $p$ and $q$ are chosen powers of two for performance reasons.

The configurations of this parameter set are meant to be used in a unified implementation of Round2 so that the same implementation can be seamlessly used in a ring or in a non-ring setting.

**Parameter set *nRound2*: NTT-friendly Round2** The goal of this parameter set is to give the possibility of even higher computational performance by obtaining parameters that are NTT-friendly.

For this parameter set, $n = d$ where $n + 1$ is a prime larger than two, and is designated as nRound2$_{n=d}$. The modulus $q$ is a prime number such that $q \equiv 1 \pmod{n+1}$, and the modulus $p$ is a power of two.

Note however that in order to achieve optimal computational/CPU performance, additional constraints may be placed on the parameter $q$. For example, to optimize reductions modulo $q$, one may wish to use a $q$ that is close to a power of 2. Such constraints may lead to less choices for $(q, n)$ which may come at the price of slightly higher bandwidth requirements.

Our reference implementation can be configured with parameters in both parameter sets, *uRound2* and *nRound2*. Our optimized implementation optimizes for *uRound2* so that the schemes can be easily and seamlessly instantiated for both the LWR and RLWR problems, since both implementations rely on $q$ and $p$ values that are powers of two. We do not include code optimizing for parameters in *nRound2*, e.g., using NTT. However, performance estimates for NTT-optimized implementations are known in the literature [2, 16].

We note that the description of our schemes could be further generalized to support even other parameter sets with other features. For instance, input parameter sets such as $(d = k * n, n)$ where $k > 1$ is a positive integer and $n > 1$ lead to an instantiation of our scheme based on module lattices. We do not include this configuration in our submission, since by restricting ourselves to instantiations with either $(d, n = 1)$ or $(d, n = d)$, we can fine-tune parameter selection and hence the performance of Round2. Furthermore, a major motivation of module lattice-based constructions is to avoid additional (ideal lattice) structure, yet achieve efficient performance compared to the non-ring case. Round2 achieves this partially through the uRound2 parameter set, since switching to the non-ring instantiation is simple and seamless even during actual deployment, by selecting the parameter set uRound2$_{n=1}$. However, the performance of uRound2$_{n=d}$ remains superior.

### 2.9.2 NIST Security Levels

NIST requires security levels such that an attacker against Round2 – targeting either the LWR or RLWR problem depending on the input parameters – must require computational resources that are comparable or greater than those required to search for a key on a block cipher with a $b$-bit key or to find a collision on a $b$-bit hash-function.

An example for the block cipher is AES with keys of 128, 192, and 256 bits. The examples for the hash functions are SHA2 and SHA3 with an output of 256 and 384 bits.

The quantum resource estimates for AES and SHA are analyzed in [10] [24] and [4]. Without going into details, these papers analyze the actual cost of finding a b-bits key in AES-b and finding a collision in SHA-b. Their conclusion is that the cost is actually higher than the bound of b/2 bits (due to Grover's algorithm [25]) for b-bit block cipher and b/3 bits (due to BrassardHyerTapp algorithm [19]) for b-bits hash function.

However, since NIST's definition of the security level is generic and refers to a block cipher and a hash function, we will not use specific estimates as in [10], [24], [4], but we will use best bounds.

In particular, we will consider the following:

- Level 1 (128-bit block cipher): Round2 encapsulates a 128 bit key and have a strength of 64 quantum bits.
- Level 2 (256-bit hash function): Round2 encapsulates a 256 bit key and have a strength of 86 quantum bits.
- Level 3 (192-bit block cipher): Round2 encapsulates a 192 bit key and have a strength of 96 quantum bits
- Level 4 (384-bit hash function): Round2 encapsulates a 384 bit key and have a strength of 128 quantum bits
- Level 5 (256-bit block cipher): Round2 encapsulates a 256 bit key and have a strength of 128 quantum bits

### 2.9.3 Development Environment

The performance numbers or Round2 have been gathered on a MacBookPro10.1 with an Intel Core i7 2.6GHz and 16GB, running macOS 10.12.6. The code has been compiled with `gcc -O3 -fomit-frame-pointer`, using Apple LLVM version 9.0.0 (clang-900.0.38).

All tests were run 1000 times. For uRound2 parameters (i.e., $q$ a power of 2), the measurements shown are the average values of all test runs combined and are for the optimized implementation of the algorithm. For nRound2 (i.e., $q$ a prime number), performance figures are based on the reference implementation.

For the memory requirements, we have provided an indication based on the same implementations as for the performance figures. 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.).

### 2.9.4 Round2.KEM: Parameters and Performance

This section contains specific parameter values for Round2.KEM fitting the uRound2 and nRound2 configurations described at the beginning of this section.

Tables 9 and 10 present ten configurations for the unified Round2 parameter set, five configurations, one per NIST security level, for the (non-ring) parameter set uRound2$_{n=1}$, and five configurations, one per NIST security level, for the (ring) parameter set uRound2$_{n=d}$. Table 11 includes the five configurations, one per NIST security level, for the nRound2 parameter set. In all of the above tables, the values under *Security Levels* represent the cost of known attacks (discussed in Section 2.6) against the underlying LWR or RLWR problem (depending on whether $n = 1$ or $n = d$, respectively). The values under *Root-hermite Factors* (RHF) give an estimate of the quality of lattice reduction algorithm that an attacker must possess to attack Round2 through its underlying problems. For estimating these bounds on the RHF, we consider the strongest attack (typically the Hybrid attack [28], see Section 2.6.4) since this leads to the most conservative security requirements on Round2 from the defender's point of view.

Table 13 show the performance and memory usage figures for the uRound2.KEM$_{n=d}$ algorithm. Table 12 show the performance and memory usage figures for the uRound2.KEM$_{n=1,\tau=2}$ algorithm.[3] Table 14 show the performance and memory usage figures for the nRound2.KEM$_{n=d}$ algorithm.[4] Table 15, finally, compares the performance of the different variants of computing $\boldsymbol{A}$ for NIST level 5 of the uRound2.KEM algorithm.

### 2.9.5 Round2.PKE: Parameters and Performance

This section contains specific parameter values for Round2.PKE fitting the uRound2 and nRound2 configurations described at the start of this section.

Tables 16 and 17 present ten configurations for the uRound2 parameter set, five configurations, one per NIST security level, for the uRound2$_{n=1}$ parameter set and five configurations, one per NIST security level, for the nRound2$_{n=d}$ parameter set. Tables 18 includes the five configurations, one per NIST security level, for the nRound2 parameter set. The values under the *Security Levels* and *Root-hermite Factors* in these tables fields have the same meaning as in the case of Round2.KEM, see Section 2.9.4.

Note that the "Encryption overhead" of the schemes includes an additional overhead of 28 bytes that results from the DEM algorithm in use (specifically,

[3]The additional parameter $\tau$ and its corresponding value in the subscript of the Round2 parameter set notation represents the exact type of the mapping $f_n^\tau$ used for generating the Round2 public parameter $\boldsymbol{A}$.

[4]Please note that performance figures were obtained using the generic, reference, implementation, not with an NTT optimized version and are only included for completeness.

AES in GCM mode) – 12 bytes due to the transport of the initialization vector, and 16 bytes due to the authentication tag.

Table 20 show the performance and memory usage figures for the uRound2.PKE$_{n=d}$ algorithm. Table 19 show the performance and memory usage figures for the uRound2.PKE$_{n=1,\tau=2}$ algorithm.[5] Table 21 show the performance and memory usage figures for the nRound2.PKE$_{n=d}$ algorithm.[6] Table 22, finally, compares the performance of the different variants of computing $\boldsymbol{A}$ for NIST level 5 of the uRound2.PKE algorithm.

[5]The additional parameter $\tau$ and its corresponding value in the subscript of the Round2 parameter set notation represents the exact type of the mapping $f_n^\tau$ used for generating the Round2 public parameter $\boldsymbol{A}$.

[6]Please note that performance figures were obtained using the generic, reference, implementation, not with an NTT optimized version and are only included for completeness.

Table 9: uRound2.KEM$_{n=1}$ parameters

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **Parameters** | | | | | |
| $d$ | 500 | 580 | 630 | 786 | 786 |
| $n$ | 1 | 1 | 1 | 1 | 1 |
| $h$ | 74 | 116 | 126 | 156 | 156 |
| $q$ | $2^{14}$ | $2^{15}$ | $2^{15}$ | $2^{15}$ | $2^{15}$ |
| $p$ | $2^{11}$ | $2^{11}$ | $2^{11}$ | $2^{11}$ | $2^{11}$ |
| $t$ | $2^{6}$ | $2^{6}$ | $2^{7}$ | $2^{8}$ | $2^{8}$ |
| $B$ | 4 | 4 | 4 | 4 | 4 |
| $\overline{n}$ | 5 | 8 | 6 | 10 | 8 |
| $\overline{m}$ | 7 | 8 | 8 | 10 | 8 |
| $\mu$ | 32 | 64 | 48 | 96 | 64 |
| **Performance** | | | | | |
| Total bandwidth (bytes) | 8292 | 12841 | 12195 | 21761 | 17389 |
| Public-key (bytes) | 3455 | 6413 | 5223 | 10857 | 8679 |
| Ciphertext (bytes) | 4837 | 6428 | 6972 | 10904 | 8710 |
| Failure rate | $2^{-101}$ | $2^{-69}$ | $2^{-88}$ | $2^{-74}$ | $2^{-74}$ |
| **Security Levels for LWR problem in ($q$, $p$)** (bits) | | | | | |
| Primal attack | 84 | 103 | 113 | 146 | 146 |
| Dual attack | 84 | 102 | 113 | 146 | 146 |
| Hybrid attack | 74 | 96 | 106 | 139 | 138 |
| Sparse-secrets attack | 83 | 102 | 112 | 146 | 146 |
| **Root-hermite Factors (for strongest attack)** | | | | | |
| $\delta$ (Hybrid attack) | 1.0055 | 1.0045 | 1.0042 | 1.0034 | 1.0035 |

Table 10: uRound2.KEM$_{n=d}$ parameters

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **Parameters** | | | | | |
| $d$ | 418 | 522 | 540 | 700 | 676 |
| $n$ | 418 | 522 | 540 | 700 | 676 |
| $h$ | 66 | 78 | 96 | 112 | 120 |
| $q$ | $2^{12}$ | $2^{15}$ | $2^{14}$ | $2^{15}$ | $2^{15}$ |
| $p$ | $2^{8}$ | $2^{8}$ | $2^{8}$ | $2^{8}$ | $2^{8}$ |
| $t$ | $2^{4}$ | $2^{3}$ | $2^{4}$ | $2^{5}$ | $2^{6}$ |
| $B$ | 1 | 1 | 1 | 1 | 1 |
| $\overline{n}$ | 1 | 1 | 1 | 1 | 1 |
| $\overline{m}$ | 1 | 1 | 1 | 1 | 1 |
| $\mu$ | 128 | 256 | 192 | 384 | 256 |
| **Performance** | | | | | |
| Total bandwidth (bytes) | 917 | 1173 | 1201 | 1689 | 1577 |
| Public-key (bytes) | 435 | 555 | 565 | 749 | 709 |
| Ciphertext (bytes) | 482 | 618 | 636 | 940 | 868 |
| Failure rate | $2^{-81}$ | $2^{-65}$ | $2^{-66}$ | $2^{-66}$ | $2^{-65}$ |
| **Security Levels for LWR problem in ($q$, $p$)** (bits) | | | | | |
| Primal attack | 81 | 106 | 112 | 151 | 147 |
| Dual attack | 83 | 108 | 114 | 154 | 149 |
| Hybrid attack | 75 | 97 | 106 | 140 | 139 |
| Sparse-secrets attack | 81 | 106 | 112 | 151 | 146 |
| **Root-hermite Factors (for strongest attack)** | | | | | |
| $\delta$ (Hybrid attack) | 1.005 | 1.0042 | 1.004 | 1.0033 | 1.0033 |

Table 11: nRound2.KEM$_{n=d}$ parameters

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **Parameters** | | | | | |
| $d$ | 400 | 486 | 556 | 658 | 658 |
| $n$ | 400 | 486 | 556 | 658 | 658 |
| $h$ | 72 | 96 | 88 | 130 | 130 |
| $q$ | 3209 | 1949 | 3343 | 1319 | 1319 |
| $p$ | $2^8$ | $2^8$ | $2^8$ | $2^8$ | $2^8$ |
| $t$ | $2^4$ | $2^4$ | $2^4$ | $2^5$ | $2^5$ |
| $B$ | 1 | 1 | 1 | 1 | 1 |
| $\overline{n}$ | 1 | 1 | 1 | 1 | 1 |
| $\overline{m}$ | 1 | 1 | 1 | 1 | 1 |
| $\mu$ | 128 | 256 | 192 | 384 | 256 |
| **Performance** | | | | | |
| Total bandwidth (bytes) | 881 | 1133 | 1233 | 1605 | 1509 |
| Public-key (bytes) | 417 | 519 | 581 | 707 | 691 |
| Ciphertext (bytes) | 464 | 614 | 652 | 898 | 818 |
| Failure rate | $2^{-69}$ | $2^{-48}$ | $2^{-54}$ | $2^{-45}$ | $2^{-45}$ |
| **Security Levels for LWR problem in ($q$, $p$)** (bits) | | | | | |
| Primal attack | 78 | 100 | 115 | 143 | 143 |
| Dual attack | 79 | 102 | 117 | 146 | 146 |
| Hybrid attack | 74 | 97 | 106 | 139 | 139 |
| Sparse-secrets attack | 78 | 100 | 115 | 143 | 143 |
| **Root-hermite Factors (for strongest attack)** | | | | | |
| $\delta$ (Hybrid attack) | 1.0051 | 1.0042 | 1.004 | 1.0033 | 1.0033 |

Table 12: uRound2.KEM$_{n=1,\tau=2}$ performance and memory usage

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **API Parameters** | | | | | |
| `CRYPTO_SECRETKEYBYTES` | 625 | 1,160 | 945 | 1,965 | 1,572 |
| `CRYPTO_PUBLICKEYBYTES` | 3,455 | 6,413 | 5,223 | 10,857 | 8,679 |
| `CRYPTO_BYTES` | 16 | 32 | 24 | 48 | 32 |
| `CRYPTO_CIPHERTEXTBYTES` | 4,837 | 6,428 | 6,972 | 10,904 | 8,710 |
| **Performance: Elapsed time (ms)** | | | | | |
| KEM Generate Key Pair | 1.3 | 2.7 | 2.6 | 4.2 | 3.6 |
| KEM Encapsulate | 1.7 | 2.9 | 3.1 | 4.6 | 3.9 |
| KEM Decapsulate | 0.1 | 0.1 | 0.1 | 0.2 | 0.1 |
| **Total** | 3.0 | 5.7 | 5.8 | 8.9 | 7.6 |
| **Performance: CPU Clock Cycles** | | | | | |
| KEM Generate Key Pair | 3.43M | 7.08M | 6.65M | 10.82M | 9.36M |
| KEM Encapsulate | 4.30M | 7.59M | 8.14M | 11.83M | 10.11M |
| KEM Decapsulate | 0.18M | 0.25M | 0.26M | 0.43M | 0.34M |
| **Total** | 7.91M | 14.92M | 15.05M | 23.08M | 19.81M |
| **Memory usage indication** | | | | | |
| KEM Generate Key Pair | 47KiB | 91KiB | 86KiB | 109KiB | 100KiB |
| KEM Encapsulate | 60KiB | 105KiB | 104KiB | 133KiB | 120KiB |
| KEM Decapsulate | 17KiB | 26KiB | 25KiB | 44KiB | 35KiB |

Table 13: uRound2.KEM$_{n=d}$ performance and memory usage

| | Security Level | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **API Parameters** | | | | | |
| CRYPTO_SECRETKEYBYTES | 105 | 131 | 135 | 175 | 169 |
| CRYPTO_PUBLICKEYBYTES | 435 | 555 | 565 | 749 | 709 |
| CRYPTO_BYTES | 16 | 32 | 24 | 48 | 32 |
| CRYPTO_CIPHERTEXTBYTES | 482 | 618 | 636 | 940 | 868 |
| **Performance: Elapsed time (ms)** | | | | | |
| KEM Generate Key Pair | 0.1 | 0.2 | 0.2 | 0.2 | 0.2 |
| KEM Encapsulate | 0.1 | 0.2 | 0.2 | 0.3 | 0.3 |
| KEM Decapsulate | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 |
| **Total** | 0.3 | 0.4 | 0.4 | 0.6 | 0.6 |
| **Performance: CPU Clock Cycles** | | | | | |
| KEM Generate Key Pair | 0.33M | 0.44M | 0.46M | 0.64M | 0.63M |
| KEM Encapsulate | 0.36M | 0.50M | 0.53M | 0.76M | 0.72M |
| KEM Decapsulate | 0.05M | 0.08M | 0.07M | 0.13M | 0.10M |
| **Total** | 0.73M | 1.02M | 1.07M | 1.53M | 1.46M |
| **Memory usage indication** | | | | | |
| KEM Generate Key Pair | 4KiB | 5KiB | 5KiB | 6KiB | 6KiB |
| KEM Encapsulate | 6KiB | 8KiB | 8KiB | 11KiB | 10KiB |
| KEM Decapsulate | 2KiB | 3KiB | 3KiB | 5KiB | 4KiB |

Table 14: nRound2.KEM$_{n=d}$ performance and memory usage (note: obtained using the generic, reference, implementation, not with an NTT optimized version)

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **API Parameters** | | | | | |
| CRYPTO_SECRETKEYBYTES | 100 | 122 | 139 | 165 | 165 |
| CRYPTO_PUBLICKEYBYTES | 417 | 519 | 581 | 707 | 691 |
| CRYPTO_BYTES | 16 | 32 | 24 | 48 | 32 |
| CRYPTO_CIPHERTEXTBYTES | 464 | 614 | 652 | 898 | 818 |
| **Performance: Elapsed time (ms)** | | | | | |
| KEM Generate Key Pair | 2.1 | 3.1 | 4.0 | 5.5 | 5.5 |
| KEM Encapsulate | 4.1 | 6.0 | 7.8 | 10.9 | 10.9 |
| KEM Decapsulate | 2.0 | 3.0 | 3.9 | 5.4 | 5.4 |
| **Total** | 8.2 | 12.1 | 15.7 | 21.8 | 21.8 |
| **Performance: CPU Clock Cycles** | | | | | |
| KEM Generate Key Pair | 5.49M | 7.99M | 10.35M | 14.35M | 14.37M |
| KEM Encapsulate | 10.68M | 15.64M | 20.30M | 28.25M | 28.26M |
| KEM Decapsulate | 5.22M | 7.66M | 9.99M | 13.96M | 13.93M |
| **Total** | 21.39M | 31.29M | 40.64M | 56.56M | 56.57M |
| **Memory usage indication** | | | | | |
| KEM Generate Key Pair | 4KiB | 4KiB | 5KiB | 6KiB | 6KiB |
| KEM Encapsulate | 6KiB | 7KiB | 8KiB | 10KiB | 10KiB |
| KEM Decapsulate | 2KiB | 3KiB | 3KiB | 4KiB | 4KiB |

Table 15: uRound2.KEM$_{n=1}$, NIST level 5 – $\boldsymbol{A}$ generation variants

| | **Variants** | | |
|---|---|---|---|
| | $f^0_{n=1}$ | $f^1_{n=1}$ | $f^2_{n=1}$ |
| **Performance: Elapsed time (ms)** | | | |
| KEM Generate Key Pair | 27.9 | 2.4 | 3.6 |
| KEM Encapsulate | 28.2 | 3.2 | 3.9 |
| KEM Decapsulate | 0.1 | 0.1 | 0.1 |
| **Total** | 56.2 | 5.8 | 7.6 |
| **Performance: CPU Clock Cycles** | | | |
| KEM Generate Key Pair | 72.40M | 6.37M | 9.36M |
| KEM Encapsulate | 73.20M | 8.36M | 10.11M |
| KEM Decapsulate | 0.34M | 0.34M | 0.34M |
| **Total** | 145.93M | 15.07M | 19.81M |
| **Memory usage indication** | | | |
| KEM Generate Key Pair | 1,241KiB | 3,655KiB | 100KiB |
| KEM Encapsulate | 1,261KiB | 3,674KiB | 120KiB |
| KEM Decapsulate | 35KiB | 35KiB | 35KiB |

Table 16: uRound2.PKE$_{n=1}$ parameters

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **Parameters** | | | | | |
| $d$ | 500 | 585 | 643 | 835 | 835 |
| $n$ | 1 | 1 | 1 | 1 | 1 |
| $h$ | 74 | 110 | 114 | 166 | 166 |
| $q$ | $2^{15}$ | $2^{15}$ | $2^{15}$ | $2^{15}$ | $2^{15}$ |
| $p$ | $2^{11}$ | $2^{11}$ | $2^{11}$ | $2^{12}$ | $2^{12}$ |
| $t$ | $2^{6}$ | $2^{9}$ | $2^{10}$ | $2^{6}$ | $2^{6}$ |
| $B$ | 4 | 4 | 4 | 4 | 4 |
| $\overline{n}$ | 5 | 8 | 6 | 10 | 8 |
| $\overline{m}$ | 7 | 8 | 8 | 10 | 8 |
| $\mu$ | 32 | 64 | 48 | 96 | 64 |
| **Performance** | | | | | |
| Total bandwidth (bytes) | 8336 | 13035 | 12515 | 25247 | 20181 |
| Public-key (bytes) | 3455 | 6468 | 5330 | 12574 | 10053 |
| Encryption overhead (bytes) | 4881 | 6567 | 7185 | 12673 | 10128 |
| Failure rate | $2^{-128}$ | $2^{-131}$ | $2^{-129}$ | $2^{-164}$ | $2^{-165}$ |
| **Security Levels for LWR problem in ($q$, $p$)** (bits) | | | | | |
| Primal attack | 84 | 103 | 115 | 146 | 146 |
| Dual attack | 84 | 103 | 115 | 146 | 146 |
| Hybrid attack | 74 | 96 | 106 | 138 | 138 |
| Sparse-secrets attack | 83 | 102 | 114 | 145 | 145 |
| **Root-hermite Factors (for strongest attack)** | | | | | |
| $\delta$ (Hybrid attack) | 1.0055 | 1.0045 | 1.0042 | 1.0035 | 1.0035 |

Table 17: uRound2.PKE$_{n=d}$ parameters

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **Parameters** | | | | | |
| $d$ | 420 | 540 | 586 | 708 | 708 |
| $n$ | 420 | 540 | 586 | 708 | 708 |
| $h$ | 62 | 96 | 104 | 140 | 140 |
| $q$ | $2^{10}$ | $2^{13}$ | $2^{13}$ | $2^{15}$ | $2^{15}$ |
| $p$ | $2^{8}$ | $2^{9}$ | $2^{9}$ | $2^{9}$ | $2^{9}$ |
| $t$ | $2^{6}$ | $2^{3}$ | $2^{3}$ | $2^{3}$ | $2^{3}$ |
| $B$ | 1 | 1 | 1 | 1 | 1 |
| $\overline{n}$ | 1 | 1 | 1 | 1 | 1 |
| $\overline{m}$ | 1 | 1 | 1 | 1 | 1 |
| $\mu$ | 128 | 256 | 192 | 384 | 256 |
| **Performance** | | | | | |
| Total bandwidth (bytes) | 997 | 1405 | 1469 | 1863 | 1783 |
| Public-key (bytes) | 437 | 641 | 685 | 846 | 830 |
| Encryption overhead (bytes) | 560 | 764 | 784 | 1017 | 953 |
| Failure rate | $2^{-300}$ | $2^{-172}$ | $2^{-154}$ | $2^{-136}$ | $2^{-137}$ |
| **Security Levels for LWR problem in ($q$, $p$)** (bits) | | | | | |
| Primal attack | 81 | 102 | 113 | 143 | 143 |
| Dual attack | 83 | 104 | 115 | 145 | 145 |
| Hybrid attack | 74 | 97 | 107 | 138 | 138 |
| Sparse-secrets attack | 81 | 102 | 112 | 143 | 143 |
| **Root-hermite Factors (for strongest attack)** | | | | | |
| $\delta$ (Hybrid attack) | 1.0051 | 1.0042 | 1.004 | 1.0033 | 1.0033 |

Table 18: nRound2.PKE$_{n=d}$ parameters

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **Parameters** | | | | | |
| $d$ | 442 | 556 | 576 | 708 | 708 |
| $n$ | 442 | 556 | 576 | 708 | 708 |
| $h$ | 74 | 88 | 108 | 140 | 140 |
| $q$ | 2659 | 3343 | 2309 | 2837 | 2837 |
| $p$ | $2^9$ | $2^9$ | $2^9$ | $2^9$ | $2^9$ |
| $t$ | $2^5$ | $2^5$ | $2^5$ | $2^5$ | $2^5$ |
| $B$ | 1 | 1 | 1 | 1 | 1 |
| $\overline{n}$ | 1 | 1 | 1 | 1 | 1 |
| $\overline{m}$ | 1 | 1 | 1 | 1 | 1 |
| $\mu$ | 128 | 256 | 192 | 384 | 256 |
| **Performance** | | | | | |
| Total bandwidth (bytes) | 1137 | 1505 | 1493 | 1959 | 1847 |
| Public-key (bytes) | 515 | 659 | 673 | 846 | 830 |
| Encryption overhead (bytes) | 622 | 846 | 820 | 1113 | 1017 |
| Failure rate | $2^{-245}$ | $2^{-225}$ | $2^{-194}$ | $2^{-164}$ | $2^{-164}$ |
| **Security Levels for LWR problem in ($q$, $p$)** (bits) | | | | | |
| Primal attack | 79 | 105 | 111 | 143 | 143 |
| Dual attack | 80 | 107 | 113 | 145 | 145 |
| Hybrid attack | 74 | 97 | 106 | 138 | 138 |
| Sparse-secrets attack | 79 | 104 | 111 | 142 | 142 |
| **Root-hermite Factors (for strongest attack)** | | | | | |
| $\delta$ (Hybrid attack) | 1.0051 | 1.0042 | 1.004 | 1.0033 | 1.0033 |

Table 19: uRound2.PKE$_{n=1,\tau=2}$ performance and memory usage

| | Security Level | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **API Parameters** | | | | | |
| CRYPTO_SECRETKEYBYTES | 4,096 | 7,670 | 6,319 | 14,710 | 11,755 |
| CRYPTO_PUBLICKEYBYTES | 3,455 | 6,468 | 5,330 | 12,574 | 10,053 |
| CRYPTO_BYTES | 4,881 | 6,567 | 7,185 | 12,673 | 10,128 |
| **Performance: Elapsed time (ms)** | | | | | |
| PKE Generate Key Pair | 2.0 | 2.7 | 2.6 | 4.4 | 3.9 |
| PKE Encrypt | 2.3 | 2.9 | 3.2 | 5.0 | 4.3 |
| PKE Decrypt | 2.3 | 3.0 | 3.2 | 5.1 | 4.4 |
| **Total** | 6.6 | 8.7 | 9.0 | 14.6 | 12.6 |
| **Performance: CPU Clock Cycles** | | | | | |
| PKE Generate Key Pair | 5.20M | 7.02M | 6.66M | 11.54M | 10.13M |
| PKE Encrypt | 5.99M | 7.63M | 8.27M | 13.03M | 11.24M |
| PKE Decrypt | 6.05M | 7.81M | 8.39M | 13.20M | 11.42M |
| **Total** | 17.24M | 22.46M | 23.32M | 37.77M | 32.79M |
| **Memory usage indication** | | | | | |
| PKE Generate Key Pair | 82KiB | 97KiB | 92KiB | 125KiB | 113KiB |
| PKE Encrypt | 92KiB | 106KiB | 105KiB | 140KiB | 125KiB |
| PKE Decrypt | 21KiB | 32KiB | 31KiB | 60KiB | 48KiB |

Table 20: uRound2.PKE$_{n=d}$ performance and memory usage

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **API Parameters** | | | | | |
| CRYPTO_SECRETKEYBYTES | 558 | 808 | 856 | 1,071 | 1,039 |
| CRYPTO_PUBLICKEYBYTES | 437 | 641 | 685 | 846 | 830 |
| CRYPTO_BYTES | 560 | 764 | 784 | 1,017 | 953 |
| **Performance: Elapsed time (ms)** | | | | | |
| PKE Generate Key Pair | 0.1 | 0.2 | 0.2 | 0.3 | 0.3 |
| PKE Encrypt | 0.1 | 0.2 | 0.2 | 0.3 | 0.3 |
| PKE Decrypt | 0.2 | 0.3 | 0.3 | 0.4 | 0.4 |
| **Total** | 0.4 | 0.7 | 0.7 | 1.0 | 1.0 |
| **Performance: CPU Clock Cycles** | | | | | |
| PKE Generate Key Pair | 0.33M | 0.52M | 0.53M | 0.73M | 0.72M |
| PKE Encrypt | 0.38M | 0.62M | 0.61M | 0.89M | 0.84M |
| PKE Decrypt | 0.41M | 0.69M | 0.68M | 1.03M | 0.94M |
| **Total** | 1.12M | 1.83M | 1.82M | 2.65M | 2.49M |
| **Memory usage indication** | | | | | |
| PKE Generate Key Pair | 4KiB | 6KiB | 6KiB | 7KiB | 7KiB |
| PKE Encrypt | 6KiB | 8KiB | 9KiB | 11KiB | 11KiB |
| PKE Decrypt | 3KiB | 4KiB | 4KiB | 6KiB | 5KiB |

Table 21: nRound2.PKE$_{n=d}$ performance and memory usage (note: obtained using the generic, reference, implementation, not with an NTT optimized version)

| | **Security Level** | | | | |
|---|---|---|---|---|---|
| | **NIST1** | **NIST2** | **NIST3** | **NIST4** | **NIST5** |
| **API Parameters** | | | | | |
| `CRYPTO_SECRETKEYBYTES` | 642 | 830 | 841 | 1,071 | 1,039 |
| `CRYPTO_PUBLICKEYBYTES` | 515 | 659 | 673 | 846 | 830 |
| `CRYPTO_BYTES` | 622 | 846 | 820 | 1,113 | 1,017 |
| **Performance: Elapsed time (ms)** | | | | | |
| PKE Generate Key Pair | 2.6 | 4.0 | 4.3 | 6.1 | 6.1 |
| PKE Encrypt | 5.0 | 7.8 | 8.4 | 12.0 | 11.9 |
| PKE Decrypt | 7.4 | 11.6 | 12.5 | 17.9 | 17.8 |
| **Total** | 15.0 | 23.4 | 25.1 | 36.0 | 35.8 |
| **Performance: CPU Clock Cycles** | | | | | |
| PKE Generate Key Pair | 6.65M | 10.32M | 11.06M | 15.85M | 15.76M |
| PKE Encrypt | 13.00M | 20.28M | 21.75M | 31.08M | 30.92M |
| PKE Decrypt | 19.30M | 30.20M | 32.33M | 46.40M | 46.29M |
| **Total** | 38.94M | 60.80M | 65.14M | 93.33M | 92.97M |
| **Memory usage indication** | | | | | |
| PKE Generate Key Pair | 5KiB | 6KiB | 6KiB | 7KiB | 7KiB |
| PKE Encrypt | 7KiB | 9KiB | 9KiB | 11KiB | 11KiB |
| PKE Decrypt | 3KiB | 4KiB | 4KiB | 6KiB | 5KiB |

Table 22: uRound2.PKE$_{n=1}$, NIST level 5 – $\boldsymbol{A}$ generation variants

| | **Variants** | | |
|---|---|---|---|
| | $f^0_{n=1}$ | $f^1_{n=1}$ | $f^2_{n=1}$ |
| **Performance: Elapsed time (ms)** | | | |
| PKE Generate Key Pair | 31.6 | 2.7 | 3.9 |
| PKE Encrypt | 32.4 | 3.9 | 4.3 |
| PKE Decrypt | 32.4 | 4.0 | 4.4 |
| **Total** | 96.3 | 10.6 | 12.6 |
| **Performance: CPU Clock Cycles** | | | |
| PKE Generate Key Pair | 81.86M | 6.99M | 10.13M |
| PKE Encrypt | 83.99M | 10.16M | 11.24M |
| PKE Decrypt | 84.00M | 10.36M | 11.42M |
| **Total** | 249.84M | 27.51M | 32.79M |
| **Memory usage indication** | | | |
| PKE Generate Key Pair | 1,409KiB | 4,133KiB | 113KiB |
| PKE Encrypt | 1,421KiB | 4,144KiB | 125KiB |
| PKE Decrypt | 48KiB | 48KiB | 48KiB |

## 2.10 Advantages and limitations

**Flexibility in the underlying problem's choice** Round2 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 the user to flexibly choose the configuration (and underlying problem instantiation) that fits best his application and security requirements, even while Round2 is already in deployment. For instance, a user 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 another user 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. Finally, as the Round2 implementation and code remains unified independent of the underlying problem instantiated, the amount of code in devices and effort required for code-review is reduced.

**Small public-keys and ciphertexts** Round2 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, Round2 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. Round2 thus is suited for bandwidth-constrained applications, and Internet protocols that allow limited packet sizes.

**Common building blocks for KEM and PKE** By design, Round2.KEM and Round2.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 Round2.KEM and Round2.PKE.

**Flexibility in achieving security levels** The design choices in Round 2, especially the choice for prime cyclotomic polynomials, allowed the fine-tuning of parameters to each NIST level. Round2 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. Round2 allows the flexible choice of parameters so that it is possible to adjust bandwidth requirements and security level according to the application needs. Round2 achieves this by using prime cyclotomic polynomials and rounding. For instance, configuration uRound.KEM$_{n=d}$ for NIST security level 1 requires the exchange of 917 Bytes so that it can be used by an application with communication

constraints; configuration uRound.KEM$_{n=1}$ for NIST security level 5 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 configuration (17389 Bytes) is larger than for the former, but is smaller than in another existing non-ring proposal for an equivalent security level [15].

**Flexibility for CPU** Different applications can have different security needs and operational constraints. Round2 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 Round2 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, configuration uRound.KEM$_{n=1}$ for NIST security level 5, intended for a high security level and not relying on a ring structure, requires around 7.5 milliseconds in our testing platform. Another application, requiring faster operation and with less security needs, can use configuration uRound.KEM$_{n=d}$ for NIST security level 1 that performs a factor 30 faster.

**Flexibility for cryptographic primitives** Round2 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 [16].

**Flexibility for integration into real-world security protocols** The Internet relies on security protocols such as TLS, IKE, SSH, IPsec, DNSSEC etc. Round2 can be easily integrated into them because it has relatively small messages and is computationally efficient. For instance, the public-key and ciphertext in uRound2.KEM$_{n=d}$ for NIST security level 5 require a total of 1577 Bytes. This is smaller than other lattice-based proposals such as [16] or [2]. At the same time, all unified Round2 configurations can be realized by means of a single library minimizing the amount of work to extend and maintain them.

**Prevention of pre-computation and back-door attacks** Round2 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 simply achieved by computing a new $\boldsymbol{A}$. In the non-ring setting, three options are provided: ($f^0_{n=1}$) randomly generating $\boldsymbol{A}$ in each protocol exchange, ($f^1_{n=1}$) permuting a fixed $\boldsymbol{A}$ in each protocol exchange, or ($f^2_{n=1}$) deriving $\boldsymbol{A}$ from a large pool of values 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 $f^2_{n=1}$ has a clear advantage compared with $f^1_{n=1}$ due to its lower memory needs.

**Post-deployment flexibility** A single implementation is used for all uRound2 configuration parameters, without the need to change or recompile the code. This approach reduces the amount of code in the devices and makes code review easier. Furthermore, it enables a unified Round2 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 Round2 uses integers of 16 bits to represent the data, which enables good performance even in architectures with constrained processors.

**Parallelization** Operations in Round2, for instance matrix multiplications, can be parallelized.

**Resistance against side channel attacks** The design of Round2 allows for efficient constant-time implementations, since by design all secrets are trinary and have a fixed number of ones and minus ones.

**Failure probability** The failure probability for Round2.KEM is at most $2^{-65}$ for all proposals in the unified parameter set. The failure probability for Round2.PKE is at most $2^{-128}$ for all proposed parameter sets. These failure probabilities can be achieved because of the usage of sparse, trinary secrets, and the large pool of parameter sets.

**Known underlying mathematical problems** Round2 builds on the well-known Learning with Rounding and Ring Learning with Rounding problems.

**Provable security** We provide security reductions from the sparse trinary LWR and RLWR problems to Round2.KEM and Round2.PKE, and from the standard Learning with Errors (LWE) problem to the sparse trinary LWR problem. Even though the latter reduction is not tight, this gives confidence in the soundness of the Round2 design.

**Easy adoption** A device that supports uRound2 can handle a wide range of configuration parameters, for both the ring and the non-ring versions, without re-compilation. This flexibility will ease a smooth adoption.

**Perfect forward secrecy** Round2'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.

# References


[1] Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. Cryptology ePrint Archive, Report 2017/047, 2017. http://eprint.iacr.org/2017/047.

[2] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092, 2015. http://eprint.iacr.org/2015/1092.

[3] Joel Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited: New reduction, properties and applications. Cryptology ePrint Archive, Report 2013/098, 2013. http://eprint.iacr.org/2013/098.

[4] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. Selected Areas in Cryptography: 23rd Annual International Workshop, SAC 2016, 03 2016.

[5] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[6] Shi Bai and Steven D. Galbraith. Lattice Decoding Attacks on Binary LWE. Cryptology ePrint Archive, Report 2013/839, 2013. http://eprint.iacr.org/2013/839.

[7] Shi Bai, Adeline Langlois, Tancrède Lepoint, Amin Sakzad, Damien Stehle, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: using the Rényi divergence rather than the statistical distance. Cryptology ePrint Archive, Report 2015/483, 2015. http://eprint.iacr.org/2015/483.

[8] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. Cryptology ePrint Archive, Report 2011/401, 2011. http://eprint.iacr.org/2011/401.

[9] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[10] Daniel J Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete, 01 2009.

[11] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime. *IACR Cryptology ePrint Archive*, 2016:461, 2016.

[12] Sauvik Bhattacharya, Oscar Garcia-Morchon, Ronald Rietman, and Ludo Tolhuizen. spKEX: An optimized lattice-based key exchange. Cryptology ePrint Archive, Report 2017/709, 2017. http://eprint.iacr.org/2017/709.

[13] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the Hardness of Learning with Rounding over Small Modulus. Cryptology ePrint Archive, Report 2015/769, 2015. http://eprint.iacr.org/2015/769.

[14] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. Cryptology ePrint Archive, Report 2010/428, 2010. http://eprint.iacr.org/2010/428.

[15] Joppe 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. Cryptology ePrint Archive, Report 2016/659, 2016. http://eprint.iacr.org/2016/659.

[16] Joppe 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. Cryptology ePrint Archive, Report 2017/634, 2017. http://eprint.iacr.org/2017/634.

[17] 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 *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 553–570, 2015.

[18] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical Hardness of Learning with Errors. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 575–584, New York, NY, USA, 2013. ACM.

[19] Gilles Brassard, Peter Hoyer, and Alain Tapp. Quantum algorithm for the collision problem. *arXiv preprint quant-ph/9705002*, 1997.

[20] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In *Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'11, pages 1–20, Berlin, Heidelberg, 2011. Springer-Verlag.

[21] Jung Hee Cheon, Kyoo Hyung Han, Jinsu Kim, Changmin Lee, and Yongha Son. A Practical Post-Quantum Public-Key Cryptosystem Based on spLWE. Cryptology ePrint Archive, Report 2016/1055, 2016. http://eprint.iacr.org/2016/1055.

[22] 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. Cryptology ePrint Archive, Report 2016/1126, 2016. http://eprint.iacr.org/2016/1126.

[23] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. Cryptology ePrint Archive, Report 2001/108, 2001. http://eprint.iacr.org/2001/108.

[24] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying Grover's Algorithm to AES: Quantum Resource Estimates. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, pages 29–43. Springer International Publishing, Cham, 2016.

[25] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[26] Dennis Hoffheinz, Kathrin Hövelmanns, and Eike Kiltz. A Modular Analysis of the Fujisaki-Okamoto Transformation. Cryptology ePrint Archive, Report 2017/604, 2017. http://eprint.iacr.org/2017/604.

[27] Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing Parameters for NTRUEncrypt, 2017.

[28] Nick Howgrave-Graham. A Hybrid Lattice-Reduction and Meet-in-the-Middle Attack Against NTRU. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings*, pages 150–169. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[29] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 232–252, 2017.

[30] Thijs Laarhoven. *Search problems in cryptography.* PhD thesis, Eindhoven University of Technology, 2015.

[31] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors Over Rings. Cryptology ePrint Archive, Report 2012/230, 2012. http://eprint.iacr.org/2012/230.

[32] National Institute of Standards and Technology. Recomendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.

NIST Special Publication 800-38D, 2007. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf.

[33] National Institute of Standards and Technology. Recomendation for Random Number Generation Using Deterministic Random Bit Generators. NIST Special Publication 800-90A, 2015. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf.

[34] National Institute of Standards and Technology. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. POST-QUANTUM CRYPTO STANDARDIZATION. Call For Proposals Announcement, 2016. http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf.

[35] Chris Peikert. Lattice cryptography for the internet. Cryptology ePrint Archive, Report 2014/070, 2014. http://eprint.iacr.org/2014/070.

[36] Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of ring-LWE for any ring and modulus. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 461–473, 2017.

[37] Oded Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.

[38] Oded Regev. The Learning with Errors Problem (Invited Survey). In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, June 9-12, 2010*, pages 191–204, 2010.

[39] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66(2):181–199, September 1994.

[40] V. Singh and A. Chopra. Even more practical key exchanges for the internet using lattice cryptography. Cryptology ePrint Archive, Report 2015/1120, 2015.

[41] Thomas Wunderer. Revisiting the hybrid attack: Improved analysis and refined security estimates. Cryptology ePrint Archive, Report 2016/733, 2016. http://eprint.iacr.org/2016/733.