Scloud$^+$: An Efficient LWE-based KEM Without Ring/Module Structure

Anyu Wang[1,6,7†], Zhongxiang Zheng[2†], Chunhuan Zhao[3], Zhiyuan Qiu[4], Guang Zeng[3], Ye Yuan[3], Changchun Mu[5], and Xiaoyun Wang[1,4,5,7,8(✉)]

[1] Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China
{anyuwang,xiaoyunwang}@tsinghua.edu.cn
[2] School of Computer and Cyber Sciences, Communication University of China, Beijing, China
zhengzx@cuc.edu.cn
[3] Shield Lab, Huawei Technologies, Beijing, China
{zhaochunhuan,zengguang13,yuanye44}@huawei.com
[4] Shandong Institute of Blockchain, Jinan, China
qiuzhiyuan@sdibc.cn
[5] Digital Currency Institute, the People's Bank of China
mchangchun@pbc.gov.cn
[6] Zhongguancun Laboratory, Beijing, China
[7] National Financial Cryptography Research Center, Beijing, China
[8] Key Laboratory of Cryptologic Technology and Information Security, School of Cyber Science and Technology, Shandong University, Qingdao, China

Abstract. We present Scloud$^+$, an LWE-based key encapsulation mechanism (KEM). The key feature of Scloud$^+$ is its use of the unstructured-LWE problem (i.e., without algebraic structures such as rings or modules) and its incorporation of ternary secrets and lattice coding to enhance performance. A notable advantage of the unstructured-LWE problem is its resistance to potential attacks exploiting algebraic structures, making it a conservative choice for constructing high-security schemes. However, a key disadvantage of such schemes is their limited computational and communication efficiency. Scloud$^+$ utilizes ternary secrets and BW_{32} lattice codes to enhance noise control and ensure robust error correction during decryption, enabling smaller parameters while maintaining low decryption failure probabilities. Equipped with these techniques, Scloud$^+$ exhibits a significant improvement in efficiency. When compared with FrodoKEM for parameter sets targeting 128, 192, and 256 bits of security respectively, Scloud$^+$ achieves practical performance with a public key size approximately $0.71 \sim 0.87$x and a ciphertext size approximately $0.56 \sim 0.78$x that of FrodoKEM. The encapsulation plus decapsulation time is approximately $0.74 \sim 0.77$x that of FrodoKEM.

Keywords: post-quantum cryptography · key encapsulation mechanism · learning with errors · lattice code · Barnes-Wall lattice

† These authors contributed equally to this work.
✉ Corresponding author.

1 Introduction

Shor's quantum algorithm [1] makes the migration to post-quantum public key cryptography inevitable. Among the post-quantum public key schemes, those based on the *learning with errors* (LWE) problem have gained particularly prevalent. The LWE problem was first introduced by Regev in 2005 [2], which roughly requires to solve a noisy linear equations modulo a known positive integer. Concretely, the goal of LWE is to find the secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, given the instance $(\mathbf{A}, \mathbf{b} = \mathbf{As} + \mathbf{e})$ where $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ is an uniformly-random matrix and $\mathbf{e}$ is an error vector with small components sampled from some probability distributions over $\mathbb{Z}$. It has been proven that the LWE problem is at least as hard as the approximate *shortest vector problem* (SVP) and the *shortest independent vectors problem* (SIVP) on lattices, which remain difficult even in the sense of quantum computing. This reduction also establishes the average-case hardness of LWE, making it a strong candidate for cryptographic constructions.

Since Regev proposed the first LWE-based public key encryption algorithm [2], various schemes have been developed based on the hardness of LWE. These schemes can be broadly divided into two categories, depending on whether they introduce algebraic structure into the LWE problem. The first category includes schemes that base their security purely on the hardness of the LWE problem without any additional algebraic structure (referred to as *unstructured-LWE*), such as FrodoKEM [3]. The second category includes schemes built on variants of the LWE problem that incorporate algebraic structures (referred to as *structured-LWE*), such as the Ring-LWE problem [4,5] and the Module-LWE problem [6]. Examples of schemes in this category include CRYSTALS-Kyber [7], Saber [8], LAC [9], Aigis [10], and etc.

The primary benefit of introducing algebraic structure is that it enables the construction of LWE-based schemes that are more 'compact', i.e., more efficient in terms of computation and communication complexity. However, the algebraic structure also complicates the ability to reduce the hardness of the structured-LWE problems to the hardness of random lattice problems (which lack such structure), such as the approximate SVP and SIVP. Instead, it is known that these LWE variants can be reduced to hard problems on algebraically structured lattices. Specifically, the Ring-LWE problem has been shown to be at least as hard as the approximate Ideal-SVP [4], and the Module-LWE problem is known to be at least as hard as the approximate Module-SVP [6]. Unlike the approximate SVP and SIVP, the hardness of the approximate Ideal-SVP and approximate Module-SVP under quantum computation remains a topic of debate. In fact, several efficient quantum algorithms for the approximate Ideal-SVP have been discovered in recent years. In 2016, Cramer et al. demonstrated that the approximate Ideal-SVP for specific cyclotomic fields with an approximation factor of $2^{\tilde{O}(\sqrt{n})}$ can be solved in quantum polynomial time [11], whereas the best-known algorithm for the approximate SVP with the same approximation factor is still sub-exponential [12]. This result has been extended to general cyclotomic fields [13,14,15,16], and arbitrary number fields [17,18]. Although it seems unlikely that these approaches can be directly extended to address the ap-

proximate Module-SVP or the Ring-LWE/Module-LWE problems, the impact of algebraic structure on security remains unclear.

As a result, schemes based on the unstructured LWE problem, such as FrodoKEM, are often regarded as conservative choices for high-security applications [19], and are considered suitable for ensuring long-term confidentiality [20]. However, a key disadvantage of such schemes is that their computation and communication efficiency is much worse than that of structured-LWE-based schemes, posing a major obstacle to their deployment in practical systems. Thus, a natural question arises regarding how the performance of unstructured-LWE-based schemes can be improved.

Two primary approaches have been explored for improving the performance of LWE-based schemes. The first approach focuses on modifying the distribution of the secret in LWE for easier sampling. In the original LWE problem, the secret is uniformly distributed over $\mathbb{Z}_q^n$. Applebaum et al. [21] demonstrated that the LWE problem remains hard if both $\mathbf{s}$ and $\mathbf{e}$ follow a Gaussian distribution, and this idea was refined in Kyber and Aigis, where $\mathbf{s}$ and $\mathbf{e}$ are set to follow a binomial distribution. LAC [9] showed that using a ternary secret, where each entry is in $\{0, \pm 1\}$, can significantly reduce parameter sizes and improve the scheme's efficiency. It is worth noting that ternary secrets are also widely adopted in homomorphic encryption schemes such as BGV [22], BFV [23], and CKKS [24]. The second approach leverages error-correction methods for improved communication efficiency. One line of such work involves using linear error-correcting codes, such as BCH codes [9,25], LDPC codes [26], and others [27,28,29]. Another line of work involves lattice coding, such as the D_4 lattice [30], the E_8 lattice [31,32], the Leech lattice [33], and others [34]. Although these methods have proven effective in boosting the efficiency of LWE-based schemes, the challenge of how to achieve performance approaching optimal for unstructured LWE-based schemes persists as a significant and unresolved problem.

1.1 Our Contributions

We present Scloud$^+$, a key encapsulation mechanism (KEM) based on the unstructured-LWE problem. In a nutshell, Scloud$^+$ leverages ternary secrets and lattice coding to significantly enhance both computational and communication efficiency. Our detailed contributions are as follows.

Ternary Secret. For all parameters, Scloud$^+$ employs a ternary secret with a Hamming weight equal to half its length. We observe that in unstructured-LWE-based schemes, two of the most time-consuming operations are the generation of the matrix $\mathbf{A}$ and the matrix-vector multiplication, i.e., the computation of $\mathbf{As}$. Employing a ternary secret in Scloud$^+$ improves noise control during decryption, enabling the use of a smaller ciphertext modulus to ensure correct decryption. This provides an opportunity to reduce matrix sizes while maintaining the same security level, thereby facilitating faster matrix sampling and more efficient matrix-vector multiplication for implementation. Furthermore, fixing

the Hamming weight of the secret to half its length prevents it from becoming overly sparse, addressing potential security concerns for sparse-secret LWE.

Lattice Coding. Scloud$^+$ designs a robust error correction method based on BW_{32} lattice codes, ensuring a smaller choice of parameters while maintaining the appropriate decryption failure probability. While the use of lattice coding is common in LWE-based constructions, previous schemes often involve lattice codes with dimensions 4 to 16. Although larger-dimensional lattice codes generally offer better signal-to-noise ratios and thus stronger error correction capabilities, they require specially designed labeling and delabeling processes to efficiently map the message to the lattice code or vice versa, which poses a challenge for high-dimensional lattice codes. For example, the 24-dimensional Leech lattice-based PKE proposed in [33] suffers from a lack of a labeling technique, making it impractical [34]. Scloud$^+$ overcomes this by designing efficient labeling and delabeling for Barnes-Wall lattice codes, enabling the use of 32-dimensional lattice codes for error correction without compromising the scheme's performance.

Security and Parameters. Scloud$^+$ provides three sets of parameters, targeting 128, 192, and 256 bits of security. Benefiting from the aforementioned techniques, we can achieve a very flexible parameter selection for Scloud$^+$, making it possible to maintain a moderate security margin (about 8 bits) for all sets of parameters while ensuring the conformed decryption failure probability. The security is comprehensively analyzed using potentially the most effective attacks for LWE, including primal attack, dual attack, and hybrid attack.

Combining the above, Scloud$^+$ achieves a remarkable improvement in its performance. Compared with FrodoKEM, Scloud$^+$ achieves a public key size approximately $0.71 \sim 0.87$x, and a ciphertext size approximately $0.56 \sim 0.78$x that of FrodoKEM, and achieves an encapsulation + decapsulation time approximately $0.74 \sim 0.77$x that of FrodoKEM.

1.2 Related Works

FrodoKEM. FrodoKEM is the first Key Encapsulation Mechanism (KEM) based on the unstructured-LWE problem. One of its distinguishing features is that both the secret and error terms follow a rounded Gaussian distribution, closely resembling the discrete Gaussian distribution from the original LWE formulation. A key modification in Scloud$^+$ is the adoption of ternary secrets. We note that ternary secrets are commonly used in homomorphic encryption and NTRU schemes, and they are generally not believed to significantly weaken the hardness of the underlying problem. To ensure the security of our parameter choices, we perform a thorough security analysis of Scloud$^+$, incorporating potentially effective LWE attacks.

Table 1. Summary of the performance of Scloud$^+$.KEM.

Scheme	Scloud$^+$-128	Scloud$^+$-192	Scloud$^+$-256
Classical Security (bits)	136.07	200.42	263.11
Decryption Failure Rates	$2^{-134.21}$	$2^{-200.64}$	$2^{-265.74}$
Public Key Size (bytes)	7200	11136	18744
Cipertext Size (bytes)	5456	10832	16916
Shared Secret Size (bytes)	16	24	32
KeyGen (10^3cycles)	1052	2034	3564
Encaps (10^3cycles)	1115	2226	3738
Decaps (10^3cycles)	1109	2262	3884

Lattice Coding for Unstructured-LWE-based Schemes. Several efforts have been made to leverage lattice coding to reduce the communication cost of unstructured-LWE-based schemes. As previously mentioned, a theoretical analysis of applying the Leech lattice to unstructured-LWE-based schemes is provided in [33], but it suffers from a lack of a labeling technique. In [32], it is demonstrated that using the E_8 lattice can reduce the communication cost of FrodoKEM by 7%. Similarly, in [34], the authors analyze the impact of applying various lattice codes up to dimension 64 on FrodoKEM, achieving a communication cost reduction of approximately 7%. However, both [32] and [34] lack computational performance evaluations, leaving it uncertain whether these improvements can be practically implemented in real-world schemes.

1.3 Outline

Section 2 lays out the preliminaries. Section 3 delves into lattice coding and our tailored approach for Scloud$^+$. The PKE and KEM are detailed in Section 4 and Section 5, respectively. Section 6 discusses the parameters and security analysis. Finally, Section 7 assesses the performance of Scloud$^+$.

2 Preliminaries

2.1 Notations

- Vectors are denoted by bold lower-case letters, such as $\mathbf{v}$, while matrices are represented by bold upper-case letters, such as $\mathbf{A}$.
- The Hamming weight of a vector $\mathbf{v}$ is denoted as $w_H(\mathbf{v})$. For any integer $0 \leq n < 2^k$, we define $\mathrm{Bit}(n,k) = (b_0, \ldots, b_{k-1}) \in \{0,1\}^k$ as the base-2 expansion of n, where $n = \sum_{i=0}^{k-1} b_i \cdot 2^i$. The Hamming weight of the vector $\mathrm{Bit}(n,k)$ is denoted by $w_H(n)$.

- The inner product of vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ is expressed as $\langle \mathbf{u}, \mathbf{v} \rangle$, and the Euclidean norm of a vector $\mathbf{v}$ is expressed as $\|\mathbf{v}\| = \langle \mathbf{v}, \mathbf{v} \rangle$. The distance between a vector $\mathbf{v}$ and a set $\mathcal{S} \subseteq \mathbb{R}^n$ is defined by $\mathrm{dist}(\mathbf{v}, \mathcal{S}) \coloneqq \min_{\mathbf{t} \in \mathcal{S}} \|\mathbf{t} - \mathbf{v}\|$.
- For any real number $x \in \mathbb{R}$, we use $\lfloor x \rfloor$ to denote the greatest integer less than or equal to x, and $\lfloor x \rceil = \lfloor x + 1/2 \rfloor$ to denote the integer closest to x. Additionally, $\lfloor x \rceil_{\mathrm{odd}}$ denotes the nearest integer to x, where any half-integer $n + 1/2$ is rounded to the closest odd integer. For integers n and $q > 0$, we denote by $[n]_q$ the integer such that $q \mid (n - [n]_q)$ and $0 \leq [n]_q < q$. These notations extend to vectors by applying the operations component-wise.
- Sampling from a distribution χ is denoted by $x \leftarrow \chi$. The uniform discrete distribution over a finite set $\mathcal{S}$ is denoted by $U(\mathcal{S})$.

2.2 Lattices and Related Problems

A lattice $\mathcal{L}$ of rank m and dimension n (with $m \leq n$) is a discrete subset of $\mathbb{R}^n$ defined as

$$\mathcal{L} = \{c_1 \mathbf{b}_1 + \cdots + c_m \mathbf{b}_m \mid c_i \in \mathbb{Z} \text{ for } 1 \leq i \leq m\}, \tag{1}$$

where $\mathbf{b}_1, \ldots, \mathbf{b}_m$ are linearly independent vectors in $\mathbb{R}^n$. The matrix $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$ is called a *basis* for $\mathcal{L}$, and we denote the lattice generated by $\mathbf{B}$ as $\mathcal{L}(\mathbf{B})$. A lattice $\mathcal{L}$ is said to be of *full rank* if $m = n$. We use $\lambda_1(\mathcal{L})$ to denote the norm of shortest non-zero lattice vector in $\mathcal{L}$.

The *fundamental parallelepiped* of a basis $\mathbf{B}$ is defined as

$$\mathcal{P}(\mathbf{B}) = \{a_1 \mathbf{b}_1 + \cdots + a_m \mathbf{b}_m \mid a_i \in [0, 1) \text{ for } 1 \leq i \leq m\}.$$

The *Voronoi cell* $\mathcal{V}(\mathcal{L})$ of a lattice $\mathcal{L}$ is the set of all points in $\mathbb{R}^n$ for which the closest lattice point is the origin $\mathbf{0}$.

Definition 1 (CVP). *Given a lattice $\mathcal{L}$ and a target vector $\mathbf{t} \in \mathbb{R}^n$, the Closest Vector Problem (CVP) asks to find a lattice point $\mathbf{v} \in \mathcal{L}$ that is closest to $\mathbf{t}$, i.e., $\|\mathbf{v} - \mathbf{t}\| \leq \|\mathbf{v}' - \mathbf{t}\|$ for all $\mathbf{v}' \in \mathcal{L}$.*

Definition 2 (BDD). *Given a lattice $\mathcal{L}$ and a target vector $\mathbf{t} \in \mathbb{R}^n$ such that $\mathit{dist}(\mathbf{t}, \mathcal{L}) \leq r$ for some radius r, the Bounded Distance Decoding (BDD) problem asks to find a lattice point $\mathbf{v} \in \mathcal{L}$ that is closest to $\mathbf{t}$, i.e., $\|\mathbf{v} - \mathbf{t}\| \leq \|\mathbf{v}' - \mathbf{t}\|$ for all $\mathbf{v}' \in \mathcal{L}$.*

Algorithms solve the above two problems are typically referred as Maximum Likelihood Decoding (MLD) algorithm and BDD algorithm (with decoding radius r) respectively.

2.3 Cryptographic Definitions

Definition 3 (PKE). *A public-key encryption* (PKE) *scheme is a tuple of algorithms* (KeyGen, Enc, Dec) *along with a message space $\mathcal{M}$.*

- *The probabilistic key generation algorithm* KeyGen *outputs a pair of public key and secret key* (pk, sk).
- *The probabilistic encryption algorithm* Enc *takes as input* pk *and a message* $m \in \mathcal{M}$*, and outputs a ciphertext* c.
- *The deterministic decryption algorithm* Dec *takes as input* sk *and* c*, and outputs either a message* $m' \in \mathcal{M}$ *or a special error symbol* $\perp \notin \mathcal{M}$.

A PKE scheme is δ-*correct* if $\mathbb{E}\left[\max_{m\in\mathcal{M}} \Pr[\mathrm{Dec}(sk, \mathrm{Enc}(pk, m)) \neq m]\right] \leq \delta$, where the expectation is taken over $(pk, sk) \leftarrow \mathrm{KeyGen}()$, and the probability is taken over the randomness of Enc. The PKE scheme in our construction is considered to satisfy IND-CPA security *(indistinguishability under chosen plaintext attack)*. Specifically, the advantage of an adversary A is defined as

$$\mathbf{Adv}^{\mathrm{CPA}}_{\mathrm{PKE}}(\mathsf{A}) = \left|\Pr\left[b = b' \mid \begin{array}{c}(pk, sk) \leftarrow \mathrm{KeyGen}(), (m_0, m_1, s) \leftarrow \mathsf{A}(pk) \\ b \leftarrow \{0,1\}, c^* \leftarrow \mathrm{Enc}(pk, m_b), b' \leftarrow \mathsf{A}(pk, c^*, s)\end{array}\right] - \frac{1}{2}\right|.$$

Definition 4 (KEM). *A key encapsulation mechanism* (KEM) *is a tuple of algorithms* (KeyGen, Encaps, Decaps) *along with a key space* $\mathcal{K}$.

- *The probabilistic key generation algorithm* KeyGen *outputs a pair of public key and secret key* (pk, sk).
- *The probabilistic encapsulation algorithm* Encaps *takes as input* pk *and outputs an ciphertext* c *and a shared secret* $\mathbf{ss} \in \mathcal{K}$.
- *The deterministic decapsulation algorithm* Decaps *takes as input* sk *and* c*, and outputs a shared secret* $\mathbf{ss}' \in \mathcal{K}$.

A KEM is δ-*correct* if $\Pr[\mathrm{Decaps}(sk, c) \neq \mathbf{ss} \mid (c, \mathbf{ss}) \leftarrow \mathrm{Encaps}(pk)] \leq \delta$, where the probability is taken over $(pk, sk) \leftarrow \mathrm{KeyGen}()$ and the randomness of *Encaps*. The KEM in our construction is considered to satisfy IND-CCA security *(indistinguishability under chosen ciphertext attack, or IND-CCA2)*. Specifically, the advantage of an adversary A is defined as

$$\mathbf{Adv}^{\mathrm{CCA}}_{\mathrm{KEM}}(\mathsf{A}) = \left|\Pr\left[b = b' \mid \begin{array}{c}(pk, sk) \leftarrow \mathrm{KeyGen}(), b \leftarrow \{0,1\} \\ (c^*, \mathbf{ss}_0) \leftarrow \mathrm{Encaps}(), \mathbf{ss}_1 \leftarrow U(\mathcal{K}) \\ b' \leftarrow \mathsf{A}^{\mathtt{DECAPS}(\cdot)}(pk, \mathbf{ss}_b, c^*)\end{array}\right] - \frac{1}{2}\right|,$$

where the $\mathtt{DECAPS}$ oracle is defined as $\mathtt{DECAPS}(\cdot) \coloneqq \mathrm{Decaps}(sk, \cdot)$, and the adversary A is not allowed to make queries with input c^*.

3 Lattice Coding and Barnes-Wall Lattices

In this section, we first introduce fundamental concepts related to lattice codes and Barnes-Wall lattices. We then present the specific lattice codes used in Scloud$^+$, along with our proposed efficient labeling and delabeling methods.

3.1 Lattice Codes

Definition 5 (Lattice Code). *Let $\mathcal{L}_s, \mathcal{L}_c \subseteq \mathbb{R}^n$ be two full-rank lattices such that $\mathcal{L}_s \subseteq \mathcal{L}_c$. A lattice code $\mathcal{C}$ (based on the nested lattices $\mathcal{L}_s \subseteq \mathcal{L}_c$) is defined as a subset of $\mathcal{L}_c$ that forms a complete set of representatives for the quotient group $\mathcal{L}_c/\mathcal{L}_s$.*

Given a basis $\mathbf{B}_s$ for $\mathcal{L}_s$, the set $\mathcal{C}$ is typically chosen to lie within the fundamental parallelepiped $\mathcal{P}(\mathbf{B}_s)$. The lattices $\mathcal{L}_s$ and $\mathcal{L}_c$ are commonly referred to as the *shaping lattice* and *coding lattice*, respectively.[1]

When a lattice code $\mathcal{C}$ is employed for message transmission over a noisy channel, the following steps are typically involved:

- *Labeling*: Each message $\mathbf{m} \in \mathcal{M}$ is mapped to a lattice vector $\mathbf{x} \in \mathcal{C}$, where $\mathcal{M}$ is the message space.
- *Lattice Decoding*: Given a noisy received vector $\mathbf{y} = \mathbf{x} + \mathbf{e} \in \mathbb{R}^n$, where $\mathbf{e}$ represents noise, this step employs an MLD or BDD algorithm. The algorithm takes $\mathbf{y}$ as the target vector and $\mathcal{L}_c$ as the lattice, producing a lattice vector $\mathbf{x}' \in \mathcal{C}$.
- *Delabeling*: The decoded lattice vector $\mathbf{x}' \in \mathcal{L}_c$ is mapped back to a message $\mathbf{m}' \in \mathcal{M}$, essentially reversing the labeling step.

Note that in the lattice decoding step, the output $\mathbf{x}'$ of the MLD or BDD algorithm may not initially fall within the lattice code $\mathcal{C}$. In such cases, an additional operation is required to reduce $\mathbf{x}'$ modulo $\mathcal{L}_s$ to ensure it belongs to $\mathcal{C}$. If $\mathcal{C}$ is chosen to lie within the fundamental parallelepiped $\mathcal{P}(\mathbf{B}_s)$, this reduction can be achieved by expressing $\mathbf{x}'$ in terms of the basis $\mathbf{B}_s$ and reducing it to lie within $\mathcal{P}(\mathbf{B}_s)$.

A sufficient condition for correct decoding (i.e., $\mathbf{m} = \mathbf{m}'$) is that the labeling function is injective, and the noise vector $\mathbf{e}$ lies either within the Voronoi cell of $\mathcal{L}_c$ (if an MLD algorithm is employed) or within a decoding radius r (if a BDD algorithm with radius r is employed). To guarantee injectivity of the labeling map, we require that $|\mathcal{M}| \leq |\mathcal{C}| = \frac{\det(\mathcal{L}_s)}{\det(\mathcal{L}_c)}$. In this work, we focus on the message space $\mathcal{M}$ consisting of all bit strings of length μ, which necessitates $\mu = \log_2(|\mathcal{M}|) \leq \log_2\left(\frac{\det(\mathcal{L}_s)}{\det(\mathcal{L}_c)}\right)$. The quantity $\frac{1}{n}\log_2\left(\frac{\det(\mathcal{L}_s)}{\det(\mathcal{L}_c)}\right)$ is referred to as the *code rate* of $\mathcal{C}$, representing the average number of encoded bits per dimension.

3.2 Barnes-Wall Lattices

The Barnes-Wall lattices form a sequence of lattices defined for dimensions n that are powers of 2. In addition to their lattice structure, it is known that Barnes-Wall lattices can be viewed as $\mathbb{Z}[\mathtt{i}]$-submodules of $\mathbb{Z}[\mathtt{i}]^{n/2}$, i.e., lattices over the Gaussian integers $\mathbb{Z}[\mathtt{i}]$ [35]. In this paper, we adopt this construction of Barnes-Wall lattices, which simplifies the labeling and delabeling processes we

[1] In some literature, these are also called the *coarse lattice* and *fine lattice*.

propose later. These lattices can be easily converted to lattices over the integers via a mapping from $\mathbb{C}^{n/2}$ to $\mathbb{R}^n$, as follows:

$$(a_1 + b_1\mathtt{i}, a_2 + b_2\mathtt{i}, \ldots, a_{n/2} + b_{n/2}\mathtt{i}) \mapsto (a_1, \ldots, a_{n/2}, b_1, \ldots, b_{n/2}), \quad (2)$$

where $a_i, b_i \in \mathbb{R}$ for $1 \leq i \leq n/2$. Under this mapping, we naturally extend the norm and distance notations, defined in Section 2 for real vectors, to complex vectors.

Throughout the remainder of this section, we denote $\phi = 1 + \mathtt{i}$, so that $\phi^{-1} = \frac{1}{2}(1 - \mathtt{i}) = \frac{1}{2}\bar{\phi}$. The Barnes-Wall lattices can be defined recursively as follows.

Definition 6 (Barnes-Wall lattice). *For any positive integer $n = 2^k \geq 4$, the n-dimensional Barnes-Wall lattice* BW_n *is defined as*

$$\mathrm{BW}_n = \{[\mathbf{u}, \mathbf{u} + \phi\mathbf{v}] \mid \mathbf{u}, \mathbf{v} \in \mathrm{BW}_{n/2}\}, \quad (3)$$

with the initial case $\mathrm{BW}_2 = \mathbb{Z}[\mathtt{i}]$.

Alternatively, it can be deduced that the Barnes-Wall lattices can also be expressed as

$$\mathrm{BW}_n = \{\mathbf{W}_n \cdot \mathbf{v} \mid \mathbf{v} \in \mathbb{Z}[i]^{\frac{n}{2}}\}, \text{ where } \mathbf{W}_n = \begin{pmatrix} 1 & 0 \\ 1 & \phi \end{pmatrix}^{\otimes(k-1)} \in \mathbb{C}^{\frac{n}{2} \times \frac{n}{2}} \quad (4)$$

is the Kronecker product of $(k-1)$ matrices $\begin{pmatrix} 1 & 0 \\ 1 & \phi \end{pmatrix}$.

For $n = 2, 4, 8$, the Barnes-Wall lattices correspond, under the mapping defined in (2), to $\mathbb{Z}^2$, D_4, and E_8, respectively. These are known to be the densest packings in their respective dimensions.

Lemma 1 ([36]). *For the Barnes-Wall lattice* BW_n*, where $n = 2^k \geq 2$, the following properties hold:*

(1) The minimum distance $\lambda_1 = \sqrt{\frac{n}{2}}$, and the packing radius $\rho = \frac{1}{2}\lambda_1 = \sqrt{\frac{n}{8}}$.
(2) The determinant $\det(\mathrm{BW}_n) = 2^{\frac{n}{4}(k-1)}$.
(3) $2^{\lfloor \frac{k}{2} \rfloor} \cdot \mathbb{Z}[\mathtt{i}]^{\frac{n}{2}} \subseteq \mathrm{BW}_n$.

Decoding Algorithms for Barnes-Wall Lattices. Several algorithms have been proposed to decode Barnes-Wall lattices, which can be broadly categorized into three main types. The first category focuses on MLD. Efficient MLD algorithms are known for Barnes-Wall lattices of specific low dimensions, such as BW_4 and BW_8 [37,38]. The only known MLD algorithm for arbitrary Barnes-Wall lattices was proposed by Forney in 1988, utilizing the trellis representation of BW_n [36]. However, the computational complexity of this algorithm is exponential in n, making it impractical for dimensions $n \geq 32$, particularly in the

Algorithm 1: BDD Algorithm for Barnes-Wall lattices

Input: A target vector $\mathbf{t} \in \mathbb{C}^{n/2}$ and the lattice BW_n
Output: A lattice vector $\mathbf{y} \in \mathrm{BW}_n$

if $n = 2$ **then**
  return $\lfloor \mathbf{t} \rceil$
else
  Write $\mathbf{t} = (\mathbf{t_1}, \mathbf{t_2})$ such that $\mathbf{t}_1, \mathbf{t}_2 \in \mathbb{C}^{n/4}$
  Compute $\mathbf{y}_1 = \mathrm{BDD}(\mathbf{t_1}, \mathrm{BW}_{n/2})$, $\mathbf{y}_2 = \mathrm{BDD}(\mathbf{t_2}, \mathrm{BW}_{n/2})$
  Compute $\mathbf{z}_1 = \mathrm{BDD}(\phi^{-1}(\mathbf{y}_1 - \mathbf{t_2}), \mathrm{BW}_{n/2})$, $\mathbf{z}_2 = \mathrm{BDD}(\phi^{-1}(\mathbf{y}_2 - \mathbf{t_1}), \mathrm{BW}_{n/2})$
  if $\|\mathbf{y}_1 - \mathbf{t}_1\|^2 + 2\|\mathbf{z}_1\|^2 < \|\mathbf{y}_2 - \mathbf{t}_2\|^2 + 2\|\mathbf{z}_2\|^2$ **then**
    return $(\mathbf{y}_1, \phi\mathbf{z}_1 + \mathbf{t}_2)$
  else
    return $(\phi\mathbf{z}_2 + \mathbf{t}_1, \mathbf{y}_2)$
  end if
end if

context of constructing efficient cryptographic schemes. The second category addresses BDD for Barnes-Wall lattices. Micciancio and Nicolosi first demonstrated that BDD can be performed in polynomial time for Barnes-Wall lattices [35]. Further improvements to this approach have been made in subsequent works [39,40]. The third category focuses on list decoding for Barnes-Wall lattices, which seeks to output all lattice vectors within a ball of radius r around a given target vector $\mathbf{t}$. A comprehensive analysis of this approach has been provided by Grigorescu and Peikert [41].

In this paper, we focus on BDD for Barnes-Wall lattices, specifically using a variant of the BDD algorithm proposed in [39]. The decoding procedure is recursive, and its details are presented in Algorithm 1. Given an input target vector $\mathbf{t}$ such that $\mathrm{dist}(\mathbf{t}, \mathrm{BW}_n) \le r = \sqrt{\frac{n}{8}}$, Algorithm 1 guarantees to return a lattice vector $\mathbf{y}$ satisfying $\|\mathbf{t} - \mathbf{y}\| \le r$.

3.3 Lattice Coding Based on Barnes-Wall Lattices

In this subsection, we focus on the message space $\mathcal{M} = \{0,1\}^{\mu}$, and present lattice codes using Barnes-Wall Lattices.

For dimension $n = 2^k \ge 4$, let the coding lattice $\mathcal{L}_c = \mathrm{BW}_n$ and the shaping lattice $\mathcal{L}_s = 2^{\tau} \cdot \mathbb{Z}[\mathtt{i}]^{n/2}$, where $\tau \ge \lfloor \frac{k}{2} \rfloor$ is a positive integer. Then by Lemma 1, it has $\mathcal{L}_s \subseteq 2^{\lfloor \frac{k}{2} \rfloor} \cdot \mathbb{Z}[\mathtt{i}]^{n/2} \subseteq \mathcal{L}_c$. Denote $\mathcal{C}$ to be the lattice code defined based on the nested lattices $\mathcal{L}_s \subseteq \mathcal{L}_c$, where each vector of $\mathcal{C}$ is required to be within the region $\mathcal{P} = \{(a_1 + b_1\mathtt{i}, a_2 + b_2\mathtt{i}, \ldots, a_{n/2} + b_{n/2}\mathtt{i}) \mid 0 \le a_j, b_j < 2^{\tau} \text{ for all } 1 \le j \le n/2\}$, i.e., the fundamental parallelepiped with respect to the standard basis of $\mathcal{L}_s = 2^{\tau} \cdot \mathbb{Z}[\mathtt{i}]^{n/2}$. By Lemma 1 it has

$$|\mathcal{C}| = \frac{\det(\mathcal{L}_s)}{\det(\mathcal{L}_c)} = \frac{2^{\tau n}}{2^{\frac{n}{4}(k-1)}} = 2^{\tau n - \frac{n}{4}(k-1)}. \tag{5}$$

Next we assume that $\mu \leq \tau n - \frac{n}{4}(k-1)$ and present the explicit labeling and delabeling methods between $\mathcal{M}$ and $\mathcal{C}$.

Algorithm 2: The Labeling Method

Input: A message vector $\mathbf{m} \in \{0,1\}^{\mu}$
Input: Positive integers n, τ such that $n = 2^k \geq 4$, $\lfloor \frac{k}{2} \rfloor \leq \tau$, and $\mu \leq \tau n - \frac{n}{4}(k-1)$
Output: A lattice vector $\mathbf{x} \in \mathcal{C}$, where $\mathcal{C}$ is the lattice code defined based on the nested lattices $2^{\tau} \cdot \mathbb{Z}[\mathtt{i}]^{n/2} \subseteq \mathrm{BW}_n$

1: Pad $\mathbf{m}$ with 0's to obtain a vector $\mathbf{m}'$ of length $\tau n - \frac{n}{4}(k-1)$
2: Write $\mathbf{m}' = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{\frac{n}{2}-1})$ such that $\mathbf{u}_j \in \{0,1\}^{2\tau - w_H(j)}$
3: Compute $\mathbf{v} = (v_0, \ldots, v_{\frac{n}{2}-1})$, where $v_j = f_{2\tau - w_H(j)}(\mathbf{u}_j)$ for $0 \leq j < n/2$
4: **for** l from 1 to $k-1$ **do**
5: Write $\mathbf{v} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{\frac{n}{2^l}})$, where $\mathbf{w}_j \in \mathbb{Z}[\mathtt{i}]^{2^{l-1}}$
6: Update $\mathbf{v} \leftarrow (\mathbf{w}_1, \mathbf{w}_1 + \phi\mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_3 + \phi\mathbf{w}_4, \ldots, \mathbf{w}_{\frac{n}{2^l}-1}, \mathbf{w}_{\frac{n}{2^l}-1} + \phi\mathbf{w}_{\frac{n}{2^l}})$
7: **end for**
8: Compute $\mathbf{w} = [\mathbf{v}]_{2^{\tau}}$
9: **return** $\mathbf{w}$

Labeling. Firstly, we pad the message $\mathbf{m}$ by appending 0's to obtain a vector

$$\mathbf{m}' \in \{0,1\}^{\tau n - \frac{n}{4}(k-1)}. \tag{6}$$

Next, we consider the mapping of $\mathbf{m}'$ to a vector in the lattice $\mathcal{C}$, which is a two-step process. The first step maps $\mathbf{m}'$ to a vector $\mathbf{v} \in \mathbb{Z}[i]^{n/2}$, while the second step maps $\mathbf{v}$ to a lattice vector $\mathbf{x} \in \mathcal{C}$. The detailed algorithm for this labeling method is outlined in Algorithm 2.

Step 1. Define a map $f_l : \{0,1\}^l \to \mathbb{Z}[\mathtt{i}]$ such that $f_l(\mathbf{u}) = a + b\mathtt{i}$, where $0 \leq a < 2^{\lceil \frac{l}{2} \rceil}, 0 \leq b < 2^{\lfloor \frac{l}{2} \rfloor}$. The first $\lceil \frac{l}{2} \rceil$ bits of $\mathbf{u}$ encode a, while the remaining $\lfloor \frac{l}{2} \rfloor$ bits encode b. Specifically, $a = \sum_{j=0}^{\lceil \frac{l}{2} \rceil - 1} u_j \cdot 2^j$ and $b = \sum_{j=\lceil \frac{l}{2} \rceil}^{l-1} u_j \cdot 2^j$, where $\mathbf{u} = (u_0, \ldots, u_{l-1})$.

Next, divide $\mathbf{m}'$ into $\frac{n}{2}$ sub-vectors, $\mathbf{m}' = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{\frac{n}{2}-1})$, where $\mathbf{u}_j \in \{0,1\}^{2\tau - w_H(j)}$. It can be verified that the sum of the lengths of $\mathbf{u}_j$ is

$$\sum_{0 \leq j < n/2} (2\tau - w_H(j)) = \tau n - \frac{n}{4}(k-1). \tag{7}$$

Finally, we map $\mathbf{m}'$ to $\mathbf{v} = (v_0, \ldots, v_{\frac{n}{2}-1}) \in \mathbb{Z}[i]^{n/2}$, where $v_j = f_{2\tau - w_H(j)}(\mathbf{u}_j)$.

Step 2. This step computes $\mathbf{x}' = \mathbf{W}_n \cdot \mathbf{v}$, where $\mathbf{W}_n$ is defined in (4). Due to the tensor product structure of $\mathbf{W}_n$, this computation is performed iteratively as shown in Algorithm 2 (line 4 to line 7). Finally, we compute $\mathbf{x} = [\mathbf{x}']_{2^{\tau}}$, producing a lattice vector in $\mathcal{C}$.

Algorithm 3: The Delabeling Method

Input: Positive integers n, τ, μ such that $n = 2^k \geq 4$, $\lfloor \frac{k}{2} \rfloor \leq \tau$, $\mu \leq \tau n - \frac{n}{4}(k-1)$
Input: A lattice vector $\mathbf{w} \in \mathcal{C}$, where $\mathcal{C}$ is the lattice code defined based on the nested lattices $2^\tau \cdot \mathbb{Z}[\mathtt{i}]^{n/2} \subseteq \mathrm{BW}_n$
Output: A message vector $\mathbf{m} \in \{0,1\}^\mu$

for l from $k-1$ to 1 **do**
  Write $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{n/2^l})$, where $\mathbf{w}_j \in \mathbb{Z}[\mathtt{i}]^{2^{l-1}}$ for $1 \leq j \leq n/2^l$
  Update $\mathbf{w} \leftarrow (\mathbf{w}_1, \phi^{-1}(\mathbf{w}_2 - \mathbf{w}_1), \mathbf{w}_3, \phi^{-1}(\mathbf{w}_4 - \mathbf{w}_3), \ldots)$
end for
Write $\mathbf{w} = (v_0, \ldots, v_{\frac{n}{2}-1})$
for j from 0 to $\frac{n}{2} - 1$ **do**
  Write $v_j = a + b\mathtt{i}$
  Compute $b' = [b]_{2^{\tau - \lceil w_H(j)/2 \rceil}}$, $a' = [a - (b - b')]_{2^{\tau - \lfloor w_H(j)/2 \rfloor}}$
  Set $v'_j = a' + b'\mathtt{i}$
end for
Compute $\mathbf{u}_j = f^{-1}_{2\tau - w_H(j)}(v'_j)$ for $0 \leq j < n/2$
Set $\mathbf{m}' = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{\frac{n}{2}-1})$ and define $\mathbf{m}$ as the first μ bits of $\mathbf{m}'$
return $\mathbf{m}$

Lemma 2. *The labeling in Algorithm 2 defines an injective map from the message space $\mathcal{M}$ to the lattice code $\mathcal{C}$.*

Proof. Let $\mathcal{S}_l$ denote the image space of the map f_l, i.e., $\mathcal{S}_l = \{a + b\mathtt{i} \mid a, b \in \mathbb{Z},\ 0 \leq a < 2^{\lceil \frac{l}{2} \rceil},\ 0 \leq b < 2^{\lfloor \frac{l}{2} \rfloor}\}$. It is evident that the first three lines of Algorithm 2 define an injective map from the message space $\mathcal{M}$ to the product space $\prod_{0 \leq j < n/2} \mathcal{S}_{2\tau - w_H(j)}$.

Next, we show that, for $\mathbf{v} \in \prod_{0 \leq j < n/2} \mathcal{S}_{2\tau - w_H(j)}$, the map $\mathbf{v} \mapsto [\mathbf{W}_n \cdot \mathbf{v}]_{2^\tau}$ is injective. Suppose there exist vectors $\mathbf{v}, \mathbf{v}' \in \prod_{0 \leq j < n/2} \mathcal{S}_{2\tau - w_H(j)}$ such that $\mathbf{W}_n \cdot \mathbf{v} = \mathbf{W}_n \cdot \mathbf{v}' \bmod 2^\tau$, i.e., $\mathbf{W}_n \cdot (\mathbf{v} - \mathbf{v}') = 0 \bmod 2^\tau$. Let $\mathbf{v} - \mathbf{v}' = (w_0, w_1, \ldots, w_{\frac{n}{2}-1})$. We will prove by induction that $w_j = 0$ for all $0 \leq j < \frac{n}{2}$.

Note that $\mathbf{W}_n$ is a lower triangular matrix, with the j-th diagonal entry given by $\phi^{w_H(j)}$. For $j = 0$, the condition $\mathbf{W}_n \cdot (\mathbf{v} - \mathbf{v}') = 0 \bmod 2^\tau$ implies $w_0 = 0 \bmod 2^\tau$. Writing $w_0 = a + b\mathtt{i}$, with $-2^\tau < a, b < 2^\tau$ (since $\mathbf{v}, \mathbf{v}' \in \prod_{0 \leq j < n/2} \mathcal{S}_{2\tau - w_H(j)}$), it follows that $a = b = 0$. Now, assume that $w_0 = \cdots = w_{j-1} = 0$. We will show that $w_j = 0$. Since $\mathbf{W}_n$ is lower triangular, the condition $\mathbf{W}_n \cdot (\mathbf{v} - \mathbf{v}') = 0 \bmod 2^\tau$ implies $w_j \cdot \phi^{w_H(j)} = 0 \bmod 2^\tau$. If $w_H(j)$ is even, we have $\phi^{w_H(j)} = (\phi^2)^{w_H(j)/2} = (-2\mathtt{i})^{w_H(j)/2}$, so $w_j \cdot \phi^{w_H(j)} = 0 \bmod 2^\tau$ implies $w_j \cdot 2^{w_H(j)/2} = 0 \bmod 2^\tau$. Writing $w_j = a + b\mathtt{i}$, with $-2^{\tau - w_H(j)/2} < a, b < 2^{\tau - w_H(j)/2}$, it follows that $a = b = 0 \bmod 2^{\tau - w_H(j)/2}$, and hence $a = b = 0$. If $w_H(j)$ is odd, a similar argument shows that $w_j \cdot \phi \cdot 2^{(w_H(j)-1)/2} = 0 \bmod 2^\tau$. Writing $w_j = a + b\mathtt{i}$, where $|a| < 2^{\tau - (w_H(j)-1)/2}$ and $|b| < 2^{\tau - (w_H(j)+1)/2}$, we get $a - b = a + b = 0 \bmod 2^{\tau - (w_H(j)-1)/2}$. From $b = \frac{1}{2}((a+b) - (a-b))$, it follows that $b = 0 \bmod 2^{\tau - (w_H(j)-1)/2 - 1}$, and thus $b = 0$. Consequently, $a = 0 \bmod 2^{\tau - (w_H(j)-1)/2}$, implying $a = 0$.

Putting all these deductions together, we conclude that the labeling is injective, completing the proof. □

Delabeling. The delabeling is the reverse of the labeling, as described in Algorithm 3. It is important to note that the computation in line 8 of Algorithm 2 cannot be completely reversed due to the modulo operation. To address this, in the delabeling, we adjust the entries of the vector $\mathbf{w} = (v_0, \ldots, v_{\frac{n}{2}-1})$ so that each v_j belongs to $\mathcal{S}_{2\tau-w_H(j)}$ (lines 6 to 10), where $\mathcal{S}_{2\tau-w_H(j)}$ is defined in the proof of Lemma 2. In fact, this ensures that $\phi^{2\tau-w_H(j)}$ divides $(v_j - v'_j)$ and thus leads the following lemma.

Lemma 3. *The delabeling in Algorithm 3 is the inverse of the labeling in Algorithm 2.*

Proof. First, we show that the computation in lines 6 to 10 ensures that $\phi^{2\tau-w_H(j)}$ divides $(v_j - v'_j)$. Observe that $v_j - v'_j = (a + b\mathtt{i}) - (a' + b'\mathtt{i}) = (a - a' - (b - b')) + (b - b')\phi$. Then by $\phi^2 \mid 2$ and the definitions of a' and b' we can deduce that $\phi^{2\tau-w_H(j)} \mid (v_j - v'_j)$.

For any $\mathbf{v} \in \prod_{0\le j<n/2} \mathcal{S}_{2\tau-w_H(j)}$, let $\mathbf{w} = [\mathbf{W}_n \cdot \mathbf{v}]_{2^\tau}$ (corresponding to lines 4 to 8 in Algorithm 2). Let $\mathbf{W}_n^{-1} \cdot \mathbf{w} = (v_0, \ldots, v_{\frac{n}{2}-1})$ and $\mathbf{v}' = (v'_0, \ldots, v'_{\frac{n}{2}-1})$ such that $\mathbf{v}' \in \prod_{0\le j<n/2} \mathcal{S}_{2\tau-w_H(j)}$ and $\phi^{2\tau-w_H(j)} \mid (v_j - v'_j)$ (corresponding to lines 1 to 10 in Algorithm 3). It suffices to show that $\mathbf{v} = \mathbf{v}'$ to complete the proof.

Let $\boldsymbol{\delta} = (\delta_0, \ldots, \delta_{\frac{n}{2}-1}) = \mathbf{W}_n^{-1}\mathbf{w} - \mathbf{v}'$. Then $\phi^{2\tau-w_H(j)} \mid \delta_j$ and

$$\mathbf{W}_n \cdot (\boldsymbol{\delta} + \mathbf{v}') = \mathbf{W}_n \cdot \mathbf{v} \bmod 2^\tau. \tag{8}$$

Since each entry of the j-th column of $\mathbf{W}_n$ is divisible by $\phi^{w_H(j)}$, it follows that each entry of $\mathbf{W}_n \cdot \boldsymbol{\delta}$ is divisible by $\phi^{2\tau}$. Therefore, $\mathbf{W}_n \cdot \boldsymbol{\delta} = 0 \bmod 2^\tau$, and (8) implies that $\mathbf{W}_n \cdot \mathbf{v}' = \mathbf{W}_n \cdot \mathbf{v} \bmod 2^\tau$. By the same reasoning used in Lemma 2, we conclude that $\mathbf{v} = \mathbf{v}'$, which completes the proof. □

4 The IND-CPA-Secure PKE

Scloud$^+$.PKE consists of three algorithms: key generation, encryption, and decryption, which are outlined in Algorithm 4 to Algorithm 6. The algorithms utilize the following parameters:

- Moduli: powers of 2 integers $q > q_1, q_2$;
- Matrix size parameters: positive integers $m, n, \bar{m}, \bar{n}$;
- Secret weight parameters: h_1, h_2;
- Error parameters: η_1, η_2;
- Message length: $l_m \in \{128, 192, 256\}$.

Algorithm 4: Scloud$^+$.PKE.KeyGen()

Output: Public key $pk \in \mathbb{Z}_q^{m\times\bar{n}} \times \{0,1\}^{128}$
Output: Secret key $sk \in \mathbb{Z}_q^{n\times\bar{n}}$
1: $\boldsymbol{\alpha} \leftarrow \{0,1\}^{256}$
2: $(\mathbf{seed_A}, \mathbf{r}_1, \mathbf{r}_2) = \mathtt{F}(\boldsymbol{\alpha}) \in \{0,1\}^{128} \times \{0,1\}^{256} \times \{0,1\}^{256}$
3: $\mathbf{A} = \mathtt{gen}(\mathbf{seed_A}) \in \mathbb{Z}_q^{m\times n}$
4: $\mathbf{S} = \Psi(\mathbf{r}_1, (n,\bar{n}), h_1) \in \mathbb{Z}^{n\times\bar{n}}$, $\mathbf{E} = \mathtt{CenBinom}(\mathbf{r}_2, (m,\bar{n}), \eta_1) \in \mathbb{Z}^{m\times\bar{n}}$
5: $\mathbf{B} = \mathbf{A}\cdot\mathbf{S} + \mathbf{E} \in \mathbb{Z}_q^{m\times\bar{n}}$
6: **return** $pk = (\mathbf{B}, \mathbf{seed_A})$, $sk = \mathbf{S}$

Algorithm 5: Scloud$^+$.PKE.Enc(pk, $\mathbf{m}$, $\mathbf{r}$)

Input: Public key $pk = (\mathbf{B}, \mathbf{seed_A}) \in \mathbb{Z}_q^{m\times\bar{n}} \times \{0,1\}^{128}$
Input: Message $\mathbf{m} \in \{0,1\}^l$
Input: Random coins $\mathbf{r} \in \{0,1\}^{256}$
Output: Ciphertext $\mathbf{C} \in \mathbb{Z}_{q_1}^{\bar{m}\times n} \times \mathbb{Z}_{q_2}^{\bar{m}\times\bar{n}}$
1: $\mathbf{A} = \mathtt{gen}(\mathbf{seed_A})$
2: $(\mathbf{r}'_1, \mathbf{r}'_2) = \mathtt{F}(\mathbf{r}) \in \{0,1\}^{256\times 2}$
3: $\mathbf{S}' = \Phi(\mathbf{r}'_1, (\bar{m}, m), h_2) \in \mathbb{Z}^{\bar{m}\times m}$
4: $\mathbf{E}' = (\mathbf{E_1}, \mathbf{E_2}) = \mathtt{CenBinom}(\mathbf{r}'_2, (\bar{m}, n+\bar{n}), \eta_2)$, where $\mathbf{E}_1 \in \mathbb{Z}^{\bar{m}\times n}$, $\mathbf{E}_2 \in \mathbb{Z}^{\bar{m}\times\bar{n}}$
5: $\mathbf{M} = \mathtt{MsgEnc}(\mathbf{m}) \in \mathbb{Z}_q^{\bar{m}\times\bar{n}}$
6: $\mathbf{C}_1 = \mathbf{S}'\cdot\mathbf{A} + \mathbf{E}_1$, $\mathbf{C}_2 = \mathbf{S}'\cdot\mathbf{B} + \mathbf{E}_2 + \mathbf{M}$
7: $\bar{\mathbf{C}}_1 = \lfloor \frac{q_1}{q}\cdot\mathbf{C}_1 \rceil$, $\bar{\mathbf{C}}_2 = \lfloor \frac{q_2}{q}\cdot\mathbf{C}_2 \rceil_{\mathrm{odd}}$
8: **return** $\mathbf{C} = (\bar{\mathbf{C}}_1, \bar{\mathbf{C}}_2)$

Algorithm 6: Scloud$^+$.PKE.Dec(sk, $\mathbf{C}$)

Input: Secret key $sk = \mathbf{S} \in \mathbb{Z}_q^{n\times\bar{n}}$
Input: Ciphertext $\mathbf{C} \in \mathbb{Z}_{q_1}^{\bar{m}\times n} \times \mathbb{Z}_{q_2}^{\bar{m}\times\bar{n}}$
Output: Message $\mathbf{m} \in \{0,1\}^l$
1: $\mathbf{C}'_1 = \frac{q}{q_1}\cdot\bar{\mathbf{C}}_1$, $\mathbf{C}'_2 = \frac{q}{q_2}\cdot\bar{\mathbf{C}}_2$
2: $\mathbf{D} = \mathbf{C}'_2 - \mathbf{C}'_1\mathbf{S} \in \mathbb{Z}_q^{\bar{m}\times\bar{n}}$
3: **return** $\mathbf{m} = \mathtt{MsgDec}(\mathbf{D}) \in \{0,1\}^l$

Distributions and Sub-functions. Scloud$^+$.PKE involves the use of the central binomial distribution and the constant Hamming distribution.

Central Binomial Distribution. Let $\rho(\eta)$ denote the central binomial distribution with parameter η. For a random variable $X \leftarrow \rho(\eta)$, it can be expressed as $X = \sum_{i=1}^{\eta}(x_i - y_i)$, where $x_i, y_i \leftarrow U(\{0,1\})$. In this scheme, we are interested in sampling a matrix $\mathbf{E} \leftarrow \rho(\eta)^{m\times n}$, generated by the sampling function `CenBinom`. This function takes random bits $\mathbf{r} \in \{0,1\}^{256}$ and parameters (m,n) and η as input, and outputs a matrix $\mathbf{E}$ drawn from $\rho(\eta)^{m\times n}$. Details of `CenBinom` are provided in Section 7.

Constant Hamming Distribution. Let $\mathcal{H}^{(m,n,h)}$ be the set of $m \times n$ matrices where each row contains exactly $(n-2h)$ zeros, h ones, and h negative ones. Similarly, let $\mathcal{L}^{(m,n,h)}$ be the set of $m \times n$ matrices where each column contains exactly $(m-2h)$ zeros, h ones, and h negative ones. In this scheme, we are interested in sampling matrices $\mathbf{S}' \leftarrow U(\mathcal{H}^{(m,n,h)})$ and $\mathbf{S} \leftarrow U(\mathcal{L}^{(m,n,h)})$. We define Φ and Ψ as two sampling functions that take random bits $\mathbf{r} \in \{0,1\}^*$ and parameters (m,n), h as input, and output matrices uniformly drawn from $\mathcal{H}^{(m,n,h)}$ and $\mathcal{L}^{(m,n,h)}$, respectively. Details of Φ and Ψ are provided in Section 7.

Sub-functions. Scloud$^+$.PKE employs a hash function $\mathtt{F} : \{0,1\}^{256} \to \{0,1\}^*$, and a function `gen` which generates a random $m \times n$ matrix $\mathbf{A}$ over $\mathbb{Z}_q$ using a seed $\mathbf{seed_A}$ as input. Additionally, the encoding and decoding functions `MsgEnc` and `MsgDec` are described below.

The `MsgEnc` and `MsgDec` Functions. The functions `MsgEnc` and `MsgDec` are constructed using the labeling, BDD, and delabeling algorithms outlined in Section 3. Specifically, we set the coding lattice to BW_{32} (i.e., $2^k = 32$). The labeling and delabeling parameters, μ and τ, are chosen as $\mu = 64, 96, 64$ and $\tau = 3, 4, 3$ for message lengths $l_m = 128, 192, 256$, respectively. Note that for these choices of μ and τ, the relation $\mu = \tau \cdot 2^k - \frac{2^k}{4}(k-1)$ holds, implying that no padding is required during the labeling process. The detailed constructions of `MsgEnc` and `MsgDec` are provided in Algorithm 7 and Algorithm 8.

Correctness of the PKE Scheme. The theorem below gives an estimate of the decryption failure rate of Scloud$^+$.PKE.

Theorem 1. *Scloud$^+$.PKE is δ-correct, where*

$$\delta \approx \frac{l_m}{\mu} \cdot \Gamma\left(16, q^2/(2^{2\tau-1}\sigma^2_{total})\right)/\Gamma(16,0)\,, \tag{9}$$

where $\sigma^2_{total} = \eta_1 h_2 + \eta_2(h_1 + \frac{1}{2}) + \frac{1}{12}((\frac{q^2}{q_2^2} + 2) + 2h_1(\frac{q^2}{q_1^2} - 1))$, *and* $\Gamma(z,a) = \int_a^\infty t^{z-1}e^{-t}dt$ *is the incomplete Gamma function.*

Proof. Let $\mathbf{F}_1 \coloneqq \mathbf{C}'_1 - \mathbf{C}_1$ and $\mathbf{F}_2 \coloneqq \mathbf{C}'_2 - \mathbf{C}_2$. Then during decryption, we have

$$\begin{aligned}\mathbf{D} &= \mathbf{C}'_2 - \mathbf{C}'_1\mathbf{S} = (\mathbf{C}_2 + \mathbf{F}_2) - (\mathbf{C}_1 + \mathbf{F}_1)\mathbf{S} \\ &= \mathbf{S'E} + (\mathbf{E}_2 + \mathbf{F}_2) - (\mathbf{E}_1 + \mathbf{F}_1)\mathbf{S} + \mathtt{MsgEnc}(\mathbf{m}).\end{aligned}$$

Algorithm 7: The `MsgEnc` Function

Input: Message $\mathbf{m} \in \{0,1\}^{l_m}$
Input: Parameters μ, τ and $q, \bar{m}, \bar{n}$ such that $32 \cdot \frac{l_m}{\mu} \leq \bar{m} \cdot \bar{n}$
Output: Matrix $\mathbf{M} \in \mathbb{Z}_q^{\bar{m} \times \bar{n}}$

Divide the message $\mathbf{m}$ into $\frac{l_m}{\mu}$ sub-vectors, $\mathbf{m}_j \in \{0,1\}^{\mu}$, where $1 \leq j \leq \frac{l_m}{\mu}$
for $j = 1$ to $\frac{l_m}{\mu}$ **do**
    Invoke Algorithm 2 (labeling) with input $\mathbf{m}_j$ and parameters $(2^k = 32, \tau)$ to obtain a vector $\mathbf{w}_j \in \mathcal{C}$
    Decompose $\mathbf{w}_j$ into $\mathbf{w}_j = \mathbf{u}_j + \mathbf{v}_j \mathbf{i}$, where $\mathbf{u}_j, \mathbf{v}_j \in \mathbb{Z}^{16}$
end for
Construct the vector $\mathbf{x} = (\mathbf{u}_1, \mathbf{v}_1, \mathbf{u}_2, \mathbf{v}_2, \dots) \in \mathbb{Z}^{32 \cdot \frac{l_m}{\mu}}$
Pad $\mathbf{x}$ with zeros to extend its length to $\bar{m}\bar{n}$
Compute the scaled vector $\mathbf{y} = (y_1, \dots, y_{\bar{m}\bar{n}}) = \frac{q}{2^\tau} \cdot \mathbf{x}$
Construct matrix $\mathbf{M}$ by setting the (i,j)-th element as $y_{\bar{n}(i-1)+j}$ for $1 \leq i \leq \bar{m}$
return $\mathbf{M}$

Algorithm 8: The `MsgDec` Function

Input: Matrix $\mathbf{M} \in \mathbb{Z}_q^{\bar{m} \times \bar{n}}$
Input: Parameters μ, τ, q, and l_m, such that $32 \cdot \frac{l_m}{\mu} \leq \bar{m} \cdot \bar{n}$
Output: Message $\mathbf{m} \in \{0,1\}^{l_m}$

Construct vector $\mathbf{y} = (y_1, \dots, y_{\bar{m}\bar{n}})$ where $y_{\bar{n}(i-1)+j}$ is the (i,j)-th element of $\mathbf{M}$
Truncate $\mathbf{y}$ to form vector $\mathbf{x} \in \mathbb{Z}^{32 \cdot \frac{l_m}{\mu}}$
Update $\mathbf{x} \leftarrow \frac{2^\tau}{q} \cdot \mathbf{x}$
Partition $\mathbf{x}$ as $\mathbf{x} = (\mathbf{u}_1, \mathbf{v}_1, \mathbf{u}_2, \mathbf{v}_2, \dots)$ such that $\mathbf{u}_j, \mathbf{v}_j \in \mathbb{Z}^{32}$
for $j = 1$ to $\frac{l_m}{\mu}$ **do**
    Compute $\mathbf{w}'_j = \mathbf{u}_j + \mathbf{v}_j \mathbf{i}$
    Invoke Algorithm 1 (BDD) on input $\mathbf{w}'_j$ to obtain lattice vector $\mathbf{w}_j \in \mathrm{BW}_{32}$
    Invoke Algorithm 3 (delabeling) with input $[\mathbf{w}_j]_{2^\tau}$ and parameters $(2^k = 32, \tau, \mu)$ to obtain $\mathbf{m}_j \in \{0,1\}^{\mu}$
end for
return $\mathbf{m} = (\mathbf{m}_1, \dots, \mathbf{m}_{l_m/\mu})$

Define $\mathbf{E}_{\text{total}} \coloneqq \mathbf{S}'\mathbf{E} + (\mathbf{E}_2 + \mathbf{F}_2) - (\mathbf{E}_1 + \mathbf{F}_1)\mathbf{S}$, then decryption is correct if $\mathtt{MsgDec}(\mathtt{MsgEnc}(\mathbf{m}) + \mathbf{E}_{\text{total}}) = \mathbf{m}$.

According to Algorithm 8, $\mathtt{MsgEnc}(\mathbf{m}) + \mathbf{E}_{\text{total}}$ can be truncated and partitioned into vectors $\boldsymbol{\omega}_j + \boldsymbol{\epsilon}_j$, where $1 \le j \le \frac{l_m}{\mu}$. The real and imaginary parts of $\boldsymbol{\epsilon}_j \in \mathbb{Z}[\mathtt{i}]^{16}$ correspond to components of $\mathbf{E}_{\text{total}}$, while those of $\boldsymbol{\omega}_j \in \frac{q}{2^\tau} \cdot \mathrm{BW}_{32}$ correspond to components of $\mathtt{MsgEnc}(\mathbf{m})$. Thus, $\mathtt{MsgDec}(\mathtt{MsgEnc}(\mathbf{m}) + \mathbf{E}_{\text{total}}) = \mathbf{m}$ holds if $\left\| \frac{2^\tau}{q} \cdot \boldsymbol{\epsilon}_j \right\| \le r$ for all j, where $r = \sqrt{\frac{32}{8}} = 2$ is the decoding radius. It follows that

$$\Pr\left[\mathtt{MsgDec}(\mathtt{MsgEnc}(\mathbf{m}) + \mathbf{E}_{\text{total}}) = \mathbf{m}\right] = \Pr\left[\|\boldsymbol{\epsilon}_j\| \le \frac{2q}{2^\tau} \text{ for all } j\right]. \tag{10}$$

Next, we analyze the distribution of the error matrix $\mathbf{E}_{\text{total}}$ to approximate (10). Since $\mathbf{F}_1 = \frac{q}{q_1} \cdot \left\lfloor \frac{q_1}{q} \cdot \mathbf{C}_1 \right\rceil - \mathbf{C}_1$ and $\mathbf{C}_1$ is pseudo-random based on the hardness of LWE, we assume that each component of $\mathbf{F}_1$ follows the distribution $\Omega_{\mathbf{F}_1} \coloneqq U\left(\left\{-\frac{q}{2q_1} + 1, \cdots, \frac{q}{2q_1}\right\}\right)$. Similarly, we assume that each component of $\mathbf{F}_2$ follows the distribution $\Omega_{\mathbf{F}_2}$, which equals $\pm\frac{q}{2q_2}$ with probability $\frac{q_2}{2q}$, and equals each $j \in \left\{-\frac{q}{2q_2} + 1, \cdots, \frac{q}{2q_2} - 1\right\}$ with probability $\frac{q_2}{q}$. The variance of $\Omega_{\mathbf{F}_1}$ is $\frac{1}{12}\left(\frac{q^2}{q_1^2} - 1\right)$, and the variance of $\Omega_{\mathbf{F}_2}$ is $\frac{1}{12}\left(\frac{q^2}{q_2^2} + 2\right)$. Considering that $\mathbf{S} \leftarrow U(\mathcal{H}^{(m,n,h_1)}), \mathbf{S}' \leftarrow U(\mathcal{L}^{(m,n,h_2)})$, $\mathbf{E} \leftarrow \rho(\eta_1)^{m \times \bar{n}}$, and $(\mathbf{E}_1, \mathbf{E}_2) \leftarrow \rho(\eta_2)^{\bar{m} \times (n+\bar{n})}$, we deduce that each component of $\mathbf{E}_{\text{total}}$ follows a distribution χ_{total} with mean 0 and variance

$$\sigma_{\text{total}}^2 = 2h_2 \cdot \frac{\eta_1}{2} + \left(\frac{\eta_2}{2} + \frac{1}{12}\left(\frac{q^2}{q_2^2} + 2\right)\right) + 2h_1 \cdot \left(\frac{\eta_2}{2} + \frac{1}{12}\left(\frac{q^2}{q_1^2} - 1\right)\right). \tag{11}$$

Since χ_{total} can be viewed as the sum of independent random variables, we can approximate χ_{total} as the discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z},\sigma_{\text{total}}}$. Assuming each component of $\mathbf{E}_{\text{total}}$ is independent, it follows that, for each j,

$$\begin{aligned}
\Pr\left[\|\boldsymbol{\epsilon}_j\| \le \frac{2q}{2^\tau}\right] &\approx \sum_{\mathbf{z} \in \mathbb{Z}^{32}, \frac{2^\tau}{q} \cdot \|\mathbf{z}\| \le 2} \prod_{j=1}^{32} \frac{g_{\sigma_{\text{total}}}(z_i)}{g_{\sigma_{\text{total}}}(\mathbb{Z})} \\
&\approx \int_{\mathbf{x} \in \mathbb{R}^{32}, \|\mathbf{x}\| \le \frac{2q}{2^\tau}} \frac{1}{(2\pi\sigma_{\text{total}}^2)^{16}} \cdot e^{-\|\mathbf{x}\|^2/(2\sigma_{\text{total}}^2)}\, d\mathbf{x} \\
&= 1 - \Gamma\left(16, \left(2q/(2^\tau\sqrt{2}\sigma_{\text{total}})\right)^2\right)/\Gamma(16, 0),
\end{aligned}$$

where $g_\sigma(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-z^2/(2\sigma^2)}$ is the Gaussian function. The second approximation comes from treating the discrete Gaussian distribution as continuous. Thus, the theorem follows directly from $\delta = 1 - \Pr\left[\|\boldsymbol{\epsilon}_j\| \le \frac{2q}{2^\tau} \text{ for all } j\right] \approx \frac{l_m}{\mu} \cdot \left(1 - \Pr\left[\|\boldsymbol{\epsilon}_j\| \le \frac{2q}{2^\tau}\right]\right)$. □

In Section 6, we select parameters such that the right-hand side of (9) is less than $2^{-128}, 2^{-192}, 2^{-256}$ for security levels targeting $128, 192, 256$ bits respectively. Note that in Theorem 1, we assume χ_{total} is approximately discrete Gaussian, which is a consequence of the law of large numbers. We verify this assumption for concrete parameters, showing that the difference between χ_{total} and the discrete Gaussian is negligible (approximately 2^{-300}). Another assumption is the independence of each component of $\mathbf{E}_{\text{total}}$, which is commonly adopted in the analysis of decryption failure rates of PKE schemes, such as [7,9]. Although the components of $\mathbf{E}_{\text{total}}$ are indeed dependent due to they are different linear combinations of the same variables, no method currently exploits this dependence to reduce the decryption failure rate or launch attacks. Therefore, we believe our approach sufficiently makes the decryption failure rate δ negligible.

Security of the PKE Scheme. The security of the proposed scheme relies on the LWE problem, which is defined as follows.

Definition 7 (LWE Distribution). *Let n, q be positive integers, and let χ_e be a distribution over $\mathbb{Z}$. Given $\mathbf{s} \in \mathbb{Z}_q^n$, the* LWE *distribution $\mathcal{A}_{\mathbf{s},\chi_e}$ outputs $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e \mod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $\mathbf{a} \leftarrow U(\mathbb{Z}_q^n)$ and $e \leftarrow \chi_e$.*

Definition 8 (LWE Problem). *Let n, m, q be positive integers, and let χ_e, χ_s be distributions over $\mathbb{Z}$ and $\mathbb{Z}^n$ respectively. The LWE problem is to distinguish between m samples $(\mathbf{a}_i, b_i) \leftarrow U(\mathbb{Z}_q^n \times \mathbb{Z}_q)$ and m samples $(\mathbf{a}_i, b_i) \leftarrow \mathcal{A}_{\mathbf{s},\chi_e}$, where the secret vector $\mathbf{s} \leftarrow \chi_s$ is common to all samples. More formally, for an adversary A, we define*

$$\mathbf{Adv}^{\text{LWE}}_{n,\,m,\,q,\,\chi_e,\,\chi_s}(\mathsf{A}) = \Big| \Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{b}) = 1 \;\middle|\; \begin{array}{c} \mathbf{A} \leftarrow U(\mathbb{Z}_q^{m \times n}), \mathbf{e} \leftarrow \chi_e^m \\ \mathbf{s} \leftarrow \chi_s, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \end{array} \right] - \Pr[\mathcal{A}(\mathbf{A}, \mathbf{b}) = 1 \mid \mathbf{A} \leftarrow U(\mathbb{Z}_q^{m \times n}), \mathbf{b} \leftarrow U(\mathbb{Z}_q^m)] \Big| .$$

In our scheme we choose χ_e to be the binomial distribution $\rho(\eta)$, and χ_s to be the constant Hamming distribution, i.e., $\mathbf{s}$ is uniformly sampled from the set of vectors in $\mathbb{Z}^n$ with exactly $(n-2h)$ zeros, h ones, and h negative ones. We denote the corresponding LWE problem to be $\mathbf{Adv}^{\text{LWE}}_{n,\,m,\,q,\,\eta,\,h}(\mathsf{A})$. It is important to note that while the original LWE problem was established using the discrete Gaussian distribution [42,5], the hardness of the LWE problem does not seem to be affected by the exact shape of the error distribution. Moreover, an analysis based on Rényi divergence demonstrates that $\rho(\eta)$ can be substituted with a discrete Gaussian distribution of variance $\eta/2$ without compromising security [43]. The following theorem establishes the security of Scloud$^+$.PKE.

Theorem 2. Scloud$^+$.PKE *is IND-CPA secure, assuming that the LWE problem is hard and that the matrix $\mathbf{A}$ generated by* `gen` *is uniformly distributed over $\mathbb{Z}_q^{m \times n}$. Specifically, for any adversary C against the IND-CPA security of* Scloud$^+$.PKE*, there exist adversaries $\mathsf{B}_1, \mathsf{B}_2$ with running times approximately equal to that of C, such that*

$$\mathbf{Adv}^{\text{CPA}}_{\text{Scloud}^+.\text{PKE}}(\mathsf{C}) \le \bar{n} \cdot \mathbf{Adv}^{\text{LWE}}_{n,\,m,\,q,\,\eta_1,\,h_1}(\mathsf{B}_1) + \bar{m} \cdot \mathbf{Adv}^{\text{LWE}}_{m+\bar{m},\,n,\,q,\,\eta_2,\,h_2}(\mathsf{B}_2).$$

Proof. We begin by considering the matrix-LWE problem where the secret and errors are also matrices (as in our PKE scheme). For an adversary A, we define

$$\mathbf{Adv}^{\text{matrix-LWE}}_{n,m,\bar{n},q,\eta,h}(\mathsf{A}) = \Big|\Pr\left[\mathsf{A}(\mathbf{A},\mathbf{B})=1 \;\middle|\; \begin{array}{l}\mathbf{A}\leftarrow U(\mathbb{Z}_q^{m\times n}),\mathbf{E}\leftarrow\rho(\eta)^{m\times\bar{n}}\\ \mathbf{S}\leftarrow U(\mathcal{L}^{(n,\bar{n},h)}),\mathbf{B}=\mathbf{AS}+\mathbf{E}\end{array}\right] - \Pr[\mathsf{A}(\mathbf{A},\mathbf{B})=1 \mid \mathbf{A}\leftarrow U(\mathbb{Z}_q^{m\times n}),\mathbf{B}\leftarrow U(\mathbb{Z}_q^{m\times\bar{n}})]\Big|.$$

Using a standard hybrid argument [44], it can be shown that there exists an adversary A' with approximately the same running time as A, such that:

$$\mathbf{Adv}^{\text{LWE}}_{n,m,q,\eta,h}(\mathsf{A}') \geq \frac{1}{\bar{n}}\mathbf{Adv}^{\text{matrix-LWE}}_{n,m,\bar{n},q,\eta,h}(\mathsf{A}). \tag{12}$$

Now, let C be executed in the IND-CPA security game G_0, where

$$\mathbf{Adv}^{\text{CPA}}_{\text{Scloud}^+.\text{PKE}}(\mathsf{C}) = |\Pr[b=b' \text{ in game } G_0] - 1/2|.$$

Define G_1 as the game where the matrix $\mathbf{B}$ in key generation is drawn from $U(\mathbb{Z}_q^{m\times\bar{n}})$ rather than generated via $\mathbf{B}=\mathbf{A}\cdot\mathbf{S}+\mathbf{E}$. Then, there exists an adversary B_1' with a running time comparable to that of C such that $|\Pr[b=b' \text{ in game } G_0] - \Pr[b=b' \text{ in game } G_1]| \leq \mathbf{Adv}^{\text{matrix-LWE}}_{n,m,\bar{n},q,\eta_1,h_1}(\mathsf{B}_1')$. Define game G_2 where the matrix $(\mathbf{C}_1=\mathbf{S}'\cdot\mathbf{A}+\mathbf{E}_1, \mathbf{C}_2'=\mathbf{S}'\cdot\mathbf{B}+\mathbf{E}_2)$ used to generate the challenge ciphertext is drawn from $U(\mathbb{Z}_q^{\bar{m}\times n}\times\mathbb{Z}_q^{\bar{m}\times\bar{n}})$. Similarly, there exists an adversary B_2' with a running time similar to that of C such that $|\Pr[b=b' \text{ in game } G_1] - \Pr[b=b' \text{ in game } G_2]| \leq \mathbf{Adv}^{\text{matrix-LWE}}_{m+\bar{m},n,\bar{m},q,\eta_2,h_2}(\mathsf{B}_2')$. In game G_2, the ciphertext $\mathbf{C}_2=\mathbf{C}_2'+\mathbf{M}$ is uniformly distributed over $\mathbb{Z}_q^{\bar{m}\times\bar{n}}$ and is independent of the message $\mathbf{m}$, implying that $\Pr[b=b' \text{ in game } G_2]=1/2$. Thus, from the above analysis, we obtain

$$\mathbf{Adv}^{\text{CPA}}_{\text{Scloud}^+.\text{PKE}}(\mathsf{C}) \leq \mathbf{Adv}^{\text{matrix-LWE}}_{n,m,\bar{n},q,\eta_1,h_1}(\mathsf{B}_1') + \mathbf{Adv}^{\text{matrix-LWE}}_{m+\bar{m},n,\bar{m},q,\eta_2,h_2}(\mathsf{B}_2').$$

The theorem follows directly by combining this result with (12). □

5 The IND-CCA-Secure KEM

Scloud$^+$.KEM is derived from Scloud$^+$.PKE by applying the Fujisaki-Okamoto transformation with implicit rejection [45,46]. Scloud$^+$.KEM consists of three algorithms: key generation, encapsulation, and decapsulation, which are described in Algorithm 9 to Algorithm 11. These algorithms utilize three hash functions: $\mathtt{H}:\{0,1\}^*\rightarrow\{0,1\}^{256}$, $\mathtt{G}:\{0,1\}^*\rightarrow\{0,1\}^{256\times 2}$, and $\mathtt{K}:\{0,1\}^*\rightarrow\{0,1\}^{l_{\mathbf{ss}}}$, where $l_{\mathbf{ss}}$ is the length of the shared secret $\mathbf{ss}$, which we set to be equal to l_m.

Correctness and Security of the KEM. It is evident that the failure probability δ of Scloud$^+$.KEM is equal to the failure probability of Scloud$^+$.PKE computed in Section 4. Following the approach in [45], when the hash functions H, G, and K are modeled as independent random oracles, we can derive the following security bounds, for which the proof is omitted in this work.

Algorithm 9: Scloud$^+$.KEM.KeyGen()

Output: Public key $pk \in \mathbb{Z}_q^{m\times\bar{n}} \times \{0,1\}^{128}$
Output: Secret key $sk \in \mathbb{Z}_q^{n\times\bar{n}} \times \mathbb{Z}_q^{m\times\bar{n}} \times \{0,1\}^{128+256\times 2}$
1: $(pk, sk') =$ Scloud$^+$.PKE.KeyGen()
2: $\mathbf{hpk} = \mathtt{H}(pk) \in \{0,1\}^{256}$
3: $\mathbf{z} \leftarrow U(\{0,1\}^{256})$
4: $sk = (sk', pk, \mathbf{hpk}, \mathbf{z})$
5: **return** (pk, sk)

Algorithm 10: Scloud$^+$.KEM.Encaps(pk)

Input: Public key $pk \in \mathbb{Z}_q^{m\times\bar{n}} \times \{0,1\}^{128}$
Output: Ciphertext $\mathbf{C} \in \mathbb{Z}_{q_1}^{\bar{m}\times n} \times \mathbb{Z}_{q_2}^{\bar{m}\times\bar{n}}$
Output: Shared session key $\mathbf{ss} \in \{0,1\}^{l_{\mathbf{ss}}}$
1: $\mathbf{m} \leftarrow U(\{0,1\}^{l_m})$
2: $(\mathbf{r}, \mathbf{k}) = \mathtt{G}(\mathbf{m}||\mathtt{H}(pk)) \in \{0,1\}^{256\times 2}$
3: $\mathbf{C} =$ Scloud$^+$.PKE.Enc$(pk, \mathbf{m}, \mathbf{r})$
4: $\mathbf{ss} = \mathtt{K}(\mathbf{k}||\mathbf{C})$
5: **return** $(\mathbf{C}, \mathbf{ss})$

Algorithm 11: Scloud$^+$.KEM.Decaps()

Input: Ciphertext $\mathbf{C} \in \mathbb{Z}_{q_1}^{\bar{m}\times n} \times \mathbb{Z}_{q_2}^{\bar{m}\times\bar{n}}$
Input: Secret key $sk = (sk', pk, \mathbf{hpk}, \mathbf{z}) \in \mathbb{Z}_q^{n\times\bar{n}} \times \mathbb{Z}_q^{m\times\bar{n}} \times \{0,1\}^{128+256\times 2}$
Output: Shared session key $\mathbf{ss} \in \{0,1\}^{l_{\mathbf{ss}}}$
1: $\mathbf{m}' =$ Scloud$^+$.PKE.Dec$(sk', \mathbf{C})$
2: $(\mathbf{r}', \mathbf{k}') = \mathtt{G}(\mathbf{m}'||\mathbf{hpk})$
3: $\mathbf{C}' =$ Scloud$^+$.PKE.Enc$(pk, \mathbf{m}', \mathbf{r}')$
4: **if** $\mathbf{C} = \mathbf{C}'$ **then**
5: **return** $\mathbf{ss} = \mathtt{K}(\mathbf{k}', \mathbf{C})$
6: **else**
7: **return** $\mathbf{ss} = \mathtt{K}(\mathbf{z}, \mathbf{C})$
8: **end if**

Theorem 3. *Suppose that Scloud$^+$.PKE is a δ-correct PKE with message space $\mathcal{M}$. Then for any classical adversary A against Scloud$^+$.KEM that makes at most q_{RO} random oracle queries, there exists a classical adversary B against Scloud$^+$.PKE, whose running time is approximately the same as that of A, such that*

$$\mathbf{Adv}^{\mathrm{CCA}}_{\text{Scloud}^+.\mathrm{KEM}}(\mathsf{A}) \leq 3 \cdot \mathbf{Adv}^{\mathrm{CPA}}_{\text{Scloud}^+.\mathrm{PKE}}(\mathsf{B}) + \frac{3q_{RO}}{|\mathcal{M}|} + q_{RO} \cdot \delta.$$

For security in the *quantum* random oracle model, the approach proposed in [47,48] can be applied to obtain the following bound.

Theorem 4. *Suppose that Scloud$^+$.PKE is a δ-correct PKE with message space $\mathcal{M}$. Then for any quantum adversary A against Scloud$^+$.KEM, making at most q_F quantum oracle queries to $\mathtt{K}$ and at most q_G quantum oracle queries to $\mathtt{G}$, there exists a quantum adversary B against Scloud$^+$.PKE, whose running time is approximately the same as that of A, such that*

$$\mathbf{Adv}^{\mathrm{CCA}}_{\text{Scloud}^+.\mathrm{KEM}}(\mathsf{A}) \leq 2\sqrt{q_{RO} \cdot \mathbf{Adv}^{\mathrm{CPA}}_{\text{Scloud}^+.\mathrm{PKE}}(\mathsf{B}) + \frac{2(q_{RO}+1)^2}{|\mathcal{M}|}} + \frac{2q_F}{\sqrt{|\mathcal{M}|}} + 4q_G \cdot \sqrt{\delta},$$

where $q_{RO} = q_F + q_G$.

6 Parameters and Security Analysis

We provide three parameter sets for Scloud$^+$, targeting classical security levels of 128, 192, and 256 bits respectively. The parameter sets are listed in Table 2, where the modulus q is fixed to be 2^{12} for all sets of parameters, and h_1, h_2 are fixed to be $\frac{1}{4}m, \frac{1}{4}n$ respectively. Additionally, the parameters for `MsgEnc` and `MsgDec` are provided in Table 3. We note that Scloud$^+$-256 selects $(\bar{m}, \bar{n}) = (12, 11)$ to accommodate 256 message bits. In this case, the encoded message has a length of 128, which is smaller than $\bar{m} \times \bar{n} = 132$. This implies that there are 4 positions in the message matrix $\mathbf{M} \in \mathbb{Z}^{\bar{m} \times \bar{n}}$ that are filled with 0.

Table 2. Parameters for Scloud$^+$.PKE and Scloud$^+$.KEM.

	$l_{\mathbf{ss}} = l_m$	(q, q_1, q_2)	(m, n)	$(\bar{m}, \bar{n})$	(h_1, h_2)	(η_1, η_2)
Scloud$^+$-128	128	$(2^{12}, 2^9, 2^7)$	$(600, 600)$	$(8, 8)$	$(150, 150)$	$(7, 7)$
Scloud$^+$-192	192	$(2^{12}, 2^{12}, 2^{10})$	$(928, 896)$	$(8, 8)$	$(224, 232)$	$(2, 1)$
Scloud$^+$-256	256	$(2^{12}, 2^{10}, 2^7)$	$(1136, 1120)$	$(12, 11)$	$(280, 284)$	$(3, 2)$

According to the reductions established in Section 4 and Section 5, it suffices to consider the LWE instances with parameters (n, m, q, η_1, h_1) and $(m + \bar{m}, n, q, \eta_2, h_2)$ to evaluate the security of the scheme.

Table 3. Parameters for `MsgEnc` and `MsgDec`.

	μ	τ	Coding lattice	Shaping lattice
Scloud$^+$-128	64	3	BW_{32}	$8 \cdot \mathbb{Z}[\mathtt{i}]^{16}$
Scloud$^+$-192	96	4	BW_{32}	$16 \cdot \mathbb{Z}[\mathtt{i}]^{16}$
Scloud$^+$-256	64	3	BW_{32}	$8 \cdot \mathbb{Z}[\mathtt{i}]^{16}$

Core-SVP Hardness. For most known attacks on LWE problems, algorithms designed for solving approximate SVP are inevitably invoked as subroutines. Among these, the BKZ lattice reduction algorithm has the best known computational complexity [49]. The overall complexity of the BKZ algorithm is primarily determined by its crucial component, which involves solving SVP in lower-dimensional lattices. Specifically, the BKZ algorithm with block size b invokes a b-dimensional SVP algorithm polynomially many times, and its cost can be described as $\mathrm{poly}(b) \cdot 2^{cb+o(b)}$, where the factor $2^{cb+o(b)}$ represents the complexity of solving the b-dimensional SVP. The *core-SVP* model is typically used to obtain a conservative estimate of the BKZ algorithm's complexity, where $o(b)$ and $\mathrm{poly}(b)$ factors are ignored, and only the core 2^{cb} complexity is considered. The best known constant c for classical algorithms is $c = \log_2(\sqrt{3/2}) \approx 0.292$, as derived from the sieve algorithm [50]. For quantum algorithms, the constant c is usually taken to be $c = \log_2(\sqrt{13/9}) \approx 0.265$ [51,30]. Although a quantum random walk approach has been shown to provide an improved quantum algorithm for solving SVP with $c \approx 0.257$ [52], this method has not been widely adopted in the analysis of LWE-based schemes. Therefore, in the analysis of quantum security, we adhere to the common choice of $c \approx 0.265$.

Primal Attack. For a given LWE instance $(\mathbf{A} \in \mathbb{Z}_q^{m \times n}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m)$, the primal attack first constructs a lattice $\Lambda = \{\mathbf{x} \in \mathbb{Z}^{m+n+1} \mid (\mathbf{A} \mid \mathbf{I}_m \mid -\mathbf{b})\mathbf{x} = 0 \bmod q\}$, and then attempts to find the unique shortest vector $\mathbf{x} = (\mathbf{s}, \mathbf{e}, 1) \in \Lambda$ using the BKZ algorithm. We refer to [30,53] for a detailed analysis of the primal attack.

Dual Attack. For a given LWE instance $(\mathbf{A} \in \mathbb{Z}_q^{m \times n}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m)$, the dual attack first finds short vectors $\{\mathbf{x}_j, \mathbf{y}_j\}_{1 \le j \le N}$ in the lattice $\Lambda' = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^{n+m} \mid \mathbf{x}^\top \mathbf{A} = \mathbf{y}^\top \bmod q\}$, and then tries to distinguish the samples $\{\mathbf{x}_j^\top \mathbf{b}\}_{1 \le j \le N}$ from samples drawn from $\mathbb{Z}_q$. We refer to [53] for a detailed analysis of the dual attack. Although some recent works report improvements to the dual attack using a Fast Fourier Transform (FFT) method [54,55], Ducas and Pulles have shown that the underlying statistical assumptions are not solid [56]. Therefore, we still adopt the commonly used dual attack as in [53].

Hybrid Dual Attack. Our scheme adopts a ternary secret for LWE, for which the hybrid attack approach is typically effective. A hybrid attack guesses part of

the secret key and then seeks a trade-off between the cost of guessing and the cost of the dual attack. The guessing techniques mainly include enumeration [53,57] and meet-in-the-middle (MITM) [58]. Recently, [59] proposes a combinatorial MITM attack technique, which is further explored in [60] for hybrid dual attacks, demonstrating better performance for extremely sparse instances.

Estimator. We provide a concrete security analysis of our scheme using a script that combines the primal, dual, and hybrid attacks. The results are summarized in Table 4.

Table 4. The estimated bits of security of Scloud$^+$ under the primal, dual and hybrid attacks. The decryption failure rates (DFRs) are calculated based on Theorem 1.

	Model	Classical	Quantum	DFR
Scloud$^+$-128	Primal	136.07	**123.49**	$2^{-134.21}$
	Dual	142.20	129.06	
	Hybrid	**136.07**	125.55	
Scloud$^+$-192	Primal	202.36	**183.65**	$2^{-200.64}$
	Dual	209.66	190.27	
	Hybrid	**200.42**	184.76	
Scloud$^+$-256	Primal	266.89	**242.21**	$2^{-265.74}$
	Dual	275.94	250.43	
	Hybrid	**263.11**	242.71	

7 Implementation

In this section, we provide the remaining implementation details for Scloud$^+$ and present the performance results of Scloud$^+$.KEM. The experimental results are summarized in Table 5. All experiments are conducted on a machine running Fedora 33 (Workstation Edition), equipped with an Intel Core-i9 10980XE @3.00GHz, with hyperthreading and TurboBoost disabled. For compilation, we used GNU GCC version 7.2.0 with the command 'gcc -O3 -march=native -lm'.

Cryptographic Primitives. Scloud$^+$ makes use of four hash functions $\mathtt{F} : \{0,1\}^{256} \to \{0,1\}^*$, $\mathtt{H} : \{0,1\}^* \to \{0,1\}^{256}$, $\mathtt{G} : \{0,1\}^* \to \{0,1\}^{2\times 256}$, and $\mathtt{K} : \{0,1\}^* \to \{0,1\}^{l_{\mathtt{ss}}}$. In our implementation, F and K are instantiated using SHAKE-256, H is instantiated using SHA3-256, and G is instantiated using SHA3-512.

Table 5. The performance of Scloud$^+$.KEM measured in 10^3 cycles, with each data representing the median count over 1000 measurements.

Scheme	KeyGen	Encaps	Decaps	Encaps + Decaps
Scloud$^+$.KEM-128	1052	1115	1109	2224
Scloud$^+$.KEM-192	2034	2226	2262	4488
Scloud$^+$.KEM-256	3564	3738	3884	7622

Matrix Generation and Multiplication. Matrix $\mathbf{E}$ is generated by first extending the random coin $\mathbf{r}_2$ to a sequence of random bits

$$(x_1, x_2, \ldots, x_{\eta_1 mn}, y_1, y_2, \ldots, y_{\eta_1 mn})$$

using SHAKE-256. Then $\mathbf{E}$ is constructed by setting its (i, j)-th element as $e_{i,j} = \sum_{k=1}^{\eta_1}(x_{\eta_1 l+k} - y_{\eta_1 l+k})$, where $l = n(i-1)+j-1$. Matrix $\mathbf{E}'$ is generated in a similar manner.

Matrix $\mathbf{S}$ is generated by first extending the random coin $\mathbf{r}_2$ into a sequence of random bits using SHAKE-256. These bits are then used to generate a constant weight distribution column by column. For each column of $\mathbf{S}$, we randomly select $2h_1$ indices from the n coordinates, setting h_1 of them to 1 and the other h_1 to -1. For Scloud$^+$-128, where $n = 600$, the simplest method to generate a random index is to use 10 bits of randomness to generate a random integer in $[0, 2^{10}-1]$, then reject any values outside $[0, n-1]$. A well-known optimization is to generate multiple indices at once to reduce random bit consumption. Specifically, since n^3 is slightly less than 2^{28}, we can sample a random integer $x \in [0, 2^{28}-1]$ and apply rejecting to obtain a random integer $y \in [0, n^3-1]$, and then output three indices $i_1 = [y]_n$, $i_2 = [\frac{y-i_1}{n}]_n$, $i_3 = [\frac{y-i_1-i_2 n}{n^2}]_n$. A similar approach applies to Scloud$^+$-256, where $n = 1120$, and 51 bits are used to generate 5 indices at once. Matrix $\mathbf{S}'$ is generated in a similar manner.

Matrix $\mathbf{A}$ is generated by extending the 128-bit random seed $\mathbf{seed_A}$ using AES-128 in CTR mode. Specifically, for any $0 \le i < m, 0 \le j < n$ such that $8 \mid i$, we generate 8 elements $a_{i,j}, a_{i+1,j}, \ldots, a_{i+8,j}$ from the 128-bit ciphertext $\text{AES-128}_{\mathbf{seed_A}}(\text{Bit}(i,16)||\text{Bit}(j,16)||\mathbf{0}_{96})$, using $\mathbf{seed_A}$ as the key, and then set the corresponding element of $\mathbf{A}$ to $a_{i,j}$.

In our implementation, matrix-vector multiplications are performed in a conventional manner, similar to FrodoKEM. Specifically, we use the '-O3' optimization in GCC, which automatically leverages SIMD (Single Instruction Multiple Data) capabilities on our platform, significantly accelerating matrix multiplication. An alternative approach for matrix multiplication in Scloud$^+$ involves storing only the nonzero indices of each column of $\mathbf{S}$ (or each row of $\mathbf{S}'$) and replacing the matrix-vector multiplications in $\mathbf{AS}$ with direct summation (or subtraction) of the columns corresponding to these nonzero indices. However, given the efficiency of SIMD, this alternative method offers no practical advantage so far.

Table 6. The size of the public key, ciphertext, and shared secret in Scloud$^+$.KEM (measured in bytes), with packing and unpacking functions applied.

Scheme	Public key pk	Ciphertext $\mathbf{C}$	Shared secret ss
Scloud$^+$.KEM-128	7200	5456	16
Scloud$^+$.KEM-192	11136	10832	24
Scloud$^+$.KEM-256	18744	16916	32

Packing and Unpacking. We note that the public key pk and ciphertext $\mathbf{C}$ should be packed for transmission. For instance, given the ciphertext $\mathbf{C} = (\bar{\mathbf{C}}_1, \bar{\mathbf{C}}_2) \in \mathbb{Z}_{q_1}^{\bar{m}\times n} \times \mathbb{Z}_{q_2}^{\bar{m}\times \bar{n}}$, we denote the elements of $\bar{\mathbf{C}}_1$ and $\bar{\mathbf{C}}_2$ as $c_{i,j}^{(1)}$ and $c_{i,j}^{(2)}$, respectively. Then, $\texttt{Pack}(\mathbf{C})$ returns the bit string

$$\left(\mathrm{Bit}(c_{i,j}^{(1)}, \log_2(q_1))\right)_{\substack{0\le i<\bar{m}\\ 0\le j<n}} \| \left(\mathrm{Bit}(c_{i,j}^{(2)}, \log_2(q_2))\right)_{\substack{0\le i<\bar{m}\\ 0\le j<\bar{n}}} \| \ \mathbf{0},$$

where 0's are padded to the end of the string to ensure the length is a multiple of 8. The function `Unpack` is the inverse of `Pack`. In Table 6, we present the sizes of the public key, and ciphertext for Scloud$^+$.KEM with the packing method applied.

Comparison of Scloud$^+$.KEM and FrodoKEM. We compare Scloud$^+$.KEM with FrodoKEM in terms of performance, as well as the sizes of the public key and ciphertext. For FrodoKEM, the performance data is based on the optimized implementation developed by the FrodoKEM team and Microsoft Research [61], tested on the same platform used to evaluate Scloud$^+$.KEM. The performance metrics, along with the public key and ciphertext sizes for FrodoKEM, are provided in Table 7 and Table 8 in Appendix A.

Note that FrodoKEM-640, FrodoKEM-976, and FrodoKEM-1344 target security levels of 128-bit, 192-bit, and 256-bit, respectively. Their concrete bits of security are comparable to those of Scloud$^+$.KEM-128, Scloud$^+$.KEM-192, and Scloud$^+$.KEM-256, allowing for meaningful comparisons between them. In terms of public key and ciphertext sizes, Scloud$^+$.KEM achieves a public key size approximately $0.71 \sim 0.87$x, and a ciphertext size approximately $0.56 \sim 0.78$x that of FrodoKEM. Regarding performance, Scloud$^+$.KEM demonstrates a key generation time approximately $0.91 \sim 0.95$x, and an encapsulation + decapsulation time approximately $0.74 \sim 0.77$x that of FrodoKEM.

Acknowledgments

We thank Matt Henricksen for valuable discussions. This work is supported by the National Key R&D Program of China (2020YFA0309705, 2018YFA0704701), the Major Program of Guangdong Basic and Applied Research (2019B030302008),

Shandong Key Research and Development Program (2020ZLYS09), and Tsinghua University Dushi Program.

Appednices

A Performance of FrodoKEM on the Same Platform

Table 7. The performance of FrodoKEM measured in 10^3 cycles, with each data representing the median count over 1000 measurements.

Scheme	KeyGen	Encaps	Decaps	Encaps + Decaps
FrodoKEM-640	1375	1541	1474	3015
FrodoKEM-976	2786	2993	2814	5807
FrodoKEM-1344	4906	5183	4992	10174

Table 8. The size of the public key, ciphertext, and shared secret in FrodoKEM (measured in bytes).

Scheme	Public key pk	Ciphertext $\mathbf{C}$	Shared secret $\mathbf{ss}$
FrodoKEM-640	9616	9720	16
FrodoKEM-976	15632	15744	24
FrodoKEM-1344	21520	21632	32

References

1. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. pages 124–134, 1994.
2. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. pages 84–93, 2005.
3. Michael Naehrig et al. FrodoKEM. Technical report, National Institute of Standards and Technology, 2020.
4. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. pages 1–23, 2010.
5. Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of ring-LWE for any ring and modulus. pages 461–473, 2017.

6. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015.
7. Peter Schwabe et al. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022.
8. Jan-Pieter D'Anvers et al. SABER. Technical report, National Institute of Standards and Technology, 2020.
9. Xianhui Lu et al. LAC. Technical report, National Institute of Standards and Technology, 2019.
10. Jiang Zhang, Yu Yu, Shuqin Fan, Zhenfeng Zhang, and Kang Yang. Tweaking the asymmetry of asymmetric-key cryptography on lattices: KEMs and signatures of smaller sizes. pages 37–65, 2020.
11. Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. pages 559–585, 2016.
12. Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.
13. Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short stickelberger class relations and application to ideal-SVP. pages 324–348, 2017.
14. Léo Ducas, Maxime Plançon, and Benjamin Wesolowski. On the shortness of vectors to be found by the ideal-SVP quantum algorithm. pages 322–351, 2019.
15. Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Mildly short vectors in cyclotomic ideal lattices in quantum polynomial time. *J. ACM*, 68(2):8:1–8:26, 2021.
16. Yanbin Pan, Jun Xu, Nick Wadleigh, and Qi Cheng. On the ideal shortest vector problem over random rational primes. In *Advances in Cryptology - EUROCRYPT 2021*, volume 12696, pages 559–583. Springer, 2021.
17. Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-SVP in ideal lattices with pre-processing. pages 685–716, 2019.
18. Olivier Bernard and Adeline Roux-Langlois. Twisted-PHS: Using the product formula to solve approx-SVP in ideal lattices. pages 349–380, 2020.
19. ANSSI. https://cyber.gouv.fr/en/publications/anssi-views-post-quantum-cryptography-transition, 2022.
20. BSI–Technical Guideline. Cryptographic mechanisms: Recommendations and key lengths, 2024.
21. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology - CRYPTO 2009*, volume 5677, pages 595–618, 2009.
22. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. pages 309–325, 2012.
23. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.
24. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. pages 409–437, 2017.
25. Zhongxiang Zheng et al. SCloud: Public key encryption and key encapsulation mechanism based on learning with errors. Cryptology ePrint Archive, Report 2020/095, 2020.
26. Tim Fritzmann, Thomas Pöppelmann, and Johanna Sepúlveda. Analysis of error-correcting codes for lattice-based key exchange. pages 369–390, 2019.
27. Markku-Juhani O. Saarinen. HILA5: On reliability, reconciliation, and error correction for ring-LWE encryption. pages 192–212, 2017.

28. Yunlei Zhao, Zhengzhong Jin, Boru Gong, and Guangye Sui. KCL (pka OKCN/AKCN/CNKE). Technical report, National Institute of Standards and Technology, 2017.
29. Mike Hamburg. Three Bears. Technical report, National Institute of Standards and Technology, 2017.
30. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key {Exchange—A} new hope. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 327–343, 2016.
31. Zhengzhong JIn and Yunlei Zhao. AKCN-E8: Compact and flexible KEM from ideal lattice. Cryptology ePrint Archive, Report 2020/056, 2020.
32. Charbel Saliba, Laura Luzzi, and Cong Ling. Error correction for frodokem using the gosset lattice. *CoRR*, abs/2110.01740, 2021.
33. Alex van Poppelen. Cryptographic decoding of the leech lattice. Cryptology ePrint Archive, Report 2016/1050, 2016. https://eprint.iacr.org/2016/1050.
34. Shanxiang Lyu, Ling Liu, Junzuo Lai, Cong Ling, and Hao Chen. Lattice codes for lattice-based PKE. Cryptology ePrint Archive, Report 2022/874, 2022.
35. Daniele Micciancio and Antonio Nicolosi. Efficient bounded distance decoders for barnes-wall lattices. In *ISIT 2008, Toronto, ON, Canada, July 6-11, 2008*, pages 2484–2488. IEEE, 2008.
36. G. David Forney Jr. Coset codes-ii: Binary lattices and related codes. *IEEE Trans. Inf. Theory*, 34(5):1152–1187, 1988.
37. Moshe Ran and Jakov Snyders. Efficient decoding of the gosset, coxeter-todd and the barnes-wall lattices. In *Proceedings. 1998 IEEE International Symposium on Information Theory (Cat. No. 98CH36252)*, page 92. IEEE, 1998.
38. Chun Wang, B Shen, and KK Tzeng. Generalised minimum distance decoding of reed-muller codes and barnes-wall lattices. In *Proceedings of 1995 IEEE International Symposium on Information Theory*, page 186. IEEE, 1995.
39. Vincent Corlay, Joseph J Boutros, Philippe Ciblat, and Loïc Brunel. On the decoding of barnes-wall lattices. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 519–524. IEEE, 2020.
40. Vincent Corlay. *Decoding Algorithms for Lattices. (Algorithmes de décodage pour les réseaux de points)*. PhD thesis, Polytechnic Institute of Paris, France, 2020.
41. Elena Grigorescu and Chris Peikert. List decoding barnes-wall lattices. In *2012 IEEE 27th Conference on Computational Complexity*, pages 316–325. IEEE, 2012.
42. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.
43. Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. pages 3–24, 2015.
44. Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. pages 1006–1018, 2016.
45. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. pages 341–371, 2017.
46. Zhongxiang Zheng, Anyu Wang, Haining Fan, Chunhuan Zhao, Chao Liu, and Xue Zhang. Scloud: Public key encryption and key encapsulation mechanism based on learning with errors. *IACR Cryptol. ePrint Arch.*, page 95, 2020.
47. Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. pages 96–125, 2018.

48. Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. pages 227–248, 2019.
49. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. pages 1–20, 2011.
50. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. pages 10–24, 2016.
51. Thijs Laarhoven. Search problems in cryptography: from fingerprinting to lattice sieving, 2016.
52. André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. pages 63–91, 2021.
53. Martin R Albrecht. On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In *Advances in Cryptology – EUROCRYPT 2017*, pages 103–129. Springer, 2017.
54. Qian Guo and Thomas Johansson. Faster dual lattice attacks for solving LWE with applications to CRYSTALS. pages 33–62, 2021.
55. MATZOV. Report on the security of lwe: Improved dual lattice attack, 2022.
56. Léo Ducas and Ludo N Pulles. Does the dual-sieve attack on learning with errors even work? In *Annual International Cryptology Conference*, pages 37–69. Springer, 2023.
57. Lei Bi, Xianhui Lu, Junjie Luo, Kunpeng Wang, and Zhenfei Zhang. Hybrid dual attack on lwe with arbitrary secrets. *Cybersecurity*, 5(1):15, 2022.
58. Jung Hee Cheon, Minki Hhan, Seungwan Hong, and Yongha Son. A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret lwe. *IEEE Access*, 7:89497–89506, 2019.
59. Alexander May. How to meet ternary lwe keys. In *Advances in Cryptology–CRYPTO 2021*, pages 701–731. Springer, 2021.
60. Lei Bi, Xianhui Lu, Junjie Luo, and Kunpeng Wang. Hybrid dual and meet-lwe attack. In *Australasian Conference on Information Security and Privacy*, pages 168–188. Springer, 2022.
61. Optimized Implementation of FrodoKEM. https://github.com/microsoft/pqcrypto-lweke, 2023.