HERMES: Efficient Ring Packing using MLWE Ciphertexts and Application to Transciphering

Youngjin Bae[1], Jung Hee Cheon[1,2], Jaehyung Kim[1], Jai Hyun Park[2], and Damien Stehlé[3]

[1] CryptoLab Inc., Seoul, Republic of Korea
[2] Seoul National University, Seoul, Republic of Korea
[3] CryptoLab Inc., Lyon, France

Abstract. Most of the current fully homomorphic encryption (FHE) schemes are based on either the learning-with-errors (LWE) problem or on its ring variant (RLWE) for storing plaintexts. During the homomorphic computation of FHE schemes, RLWE formats provide high throughput when considering several messages, and LWE formats provide a low latency when there are only a few messages. Efficient conversion can bridge the advantages of each format. However, converting LWE formats into RLWE format, which is called *ring packing*, has been a challenging problem.

We propose an efficient solution for ring packing for FHE. The main improvement of this work is twofold. First, we accelerate the existing ring packing methods by using bootstrapping and ring switching techniques, achieving practical runtimes. Second, we propose a new method for efficient ring packing, HERMES, by using ciphertexts in Module-LWE (MLWE) formats, to also reduce the memory. To this end, we generalize the tools of LWE and RLWE formats for MLWE formats.

On a single-thread implementation, HERMES consumes 10.2s for the ring packing of 2^{15} LWE-format ciphertexts into an RLWE-format ciphertext. This gives 41x higher throughput compared to the state-of-the-art ring packing for FHE, PEGASUS [S&P'21], which takes 51.7s for packing 2^{12} LWE ciphertexts with similar homomorphic capacity. We also illustrate the efficiency of HERMES by using it for transciphering from LWE symmetric encryption to CKKS fully homomorphic encryption, significantly outperforming the recent proposals HERA [Asiacrypt'21] and Rubato [Eurocrypt'22].

1 Introduction

Fully Homomorphic Encryption (FHE) is a form of encryption that enables computations on encrypted data without decryption. Most of the known FHE schemes are based either on the learning-with-errors (LWE) problem [Reg09] or on the ring learning-with-errors (RLWE) problem [SSTX09, LPR10] for storing plaintexts. The main practical FHE schemes with plaintexts stored in LWE-format ciphertexts were proposed in [DM15] and [CGGI16]. Efficient FHE schemes with plaintexts stored in RLWE-format ciphertexts include BFV [Bra12,

FV12], BGV [BGV14], and CKKS [CKKS17]. An RLWE ciphertext typically corresponds to a plaintext polynomial, whose degree is of the order of several thousands. Rather than placing the data in the coefficients of the polynomial, which is referred to coefficients-encoding, one often prefers slots-encoding: the data is placed in the frequency domain of a Fourier transform over a finite field (in the case of BGV/BFV) or over the complex numbers (in the case of CKKS). Each slot can be filled with a small modular integer, an element of a small finite field, or a real/complex number with moderate precision. By relying on slots, RLWE schemes can support Single Instruction Multiple Data (SIMD) additions and multiplications, which allow them to achieve amortized run-times. However, they are cumbersome for operations between data points stored in different slots or coefficients of the same ciphertext, and when there are only a few data points to be computed upon compared to the capacity of the ciphertext. On the other hand, LWE schemes are inefficient for handling many messages in parallel since they do not natively provide SIMD operations. Instead, they do not require a complex packing structure and provide low latency when only a few messages are considered.

Ring packing is the task of converting many LWE-format ciphertexts to an RLWE-format ciphertext. Below are several scenarios in which an efficient conversion can be particularly beneficial (some also require a reverse conversion, from RLWE-format to LWE-format, but this is typically easier to achieve).

- **Heterogeneous operation types**. Roughly speaking, RLWE schemes provide efficient addition and multiplication on small integers, complex numbers and finite field elements, while LWE schemes outperform them for table look-ups or computations on individual bits. Conversion applications to computations involving different types of operations are notably considered in [BGGJ20, LHH+21].
- **Heterogeneous computational widths**. A complex task may involve many data points at some stages, during which RLWE formats may be preferable, and much fewer at other stages, during which LWE formats may be considered. Notably, computations with plenty of inputs and a binary decision being taken at the end are considered in [BGGJ20].
- **Storing FHE ciphertexts**. The RLWE SIMD structure may be optimized depending on the specific computation to be performed. If the latter is not known in advance or diverse computations can be performed on the same data, it is interesting to store the data in LWE format, and convert it when a computation is launched. This is discussed, e.g., in [CGGI17].
- **Transciphering**. The large size of RLWE ciphertexts may be problematic when streaming data from small devices. It was suggested in [NLV11] to use a symmetric cipher for sending the data from a client to a server, and to let the server homomorphically decrypt it to obtain an FHE ciphertext. In [CDKS21], the authors use an LWE-based symmetric cipher, and homomorphic decryption reduces to ring packing.

Even though converting data stored in (many) LWE ciphertexts to a ciphertext in RLWE format is central to the above applications, the current

approaches remain relatively inefficient. The algorithm from [LHH+21] takes $\approx$ 51.7s to pack 2^{12} LWE ciphertexts into a slots-encoded RLWE ciphertext of degree 2^{16} with a single-thread implementation (the run-time is obtained from [LHH+21, Table V] by adding the 'LT' and '$\mathcal{F}_{\text{mod}}$' timings). In [CDKS21, Table 2], the authors report that their algorithm allows pack 2^5 LWE ciphertexts into a coefficients-encoded RLWE ciphertext of degree 2^{14}, in 1.17s. The authors of [BGGJ20] mention a 7s timing to pack LWE ciphertexts into a slots-encoded RLWE ciphertext of degree 2^{12}. Note further that the experiments from [BGGJ20, CDKS21] are not satisfactory for packing the order of several hundreds or thousands LWE ciphertexts into an RLWE format that allows fully homomorphic computations and is hence bootstrappable (i.e., of degree $\geq 2^{15}$).

Contributions. Our main result is an efficient ring packing algorithm, for CKKS computations, based on ring packing for RLWE-format ciphertexts with small parameters. Its concrete efficiency is supported by experiments. Finally, we illustrate the usefulness of our efficient ring packing by focusing on the transciphering application.

Our ring packing algorithm is inspired from the one proposed in [CGGI17], itself based on the column method for matrix-vector multiplication described in [HS14]. We adapt it to fully homomorphic RLWE computations in a way that widely differs from [BGGJ20]. We leverage several techniques to optimize the efficiency: CKKS bootstrapping [CHK+18], ring switching [GHPS13], and intermediate Module-LWE computations [BGV14, LS15].

We implemented our algorithm in the HEaaN library [Cry]. One implementation allows to ring-pack 2^{15} LWE ciphertexts into a slots-encoded CKKS ciphertext of degree 2^{15} on which homomorphic computations can be directly performed, in 10.2s with a single thread. This is $\approx$ 41 times faster in terms of throughput than [LHH+21, Table V] mentioned above, for a similar task.

We use our ring packing algorithm for the transciphering application described in [CDKS21]. As illustrated in Table 1, compared to state of the art transciphering algorithms for the CKKS scheme [CHK+21, HKL+22], our approach achieves a significant reduction of server run-time while retaining a small bandwidth consumption. We refer to Section 5.4 for a more detailed comparison.

Scheme	N	Latency (s)	Expansion ratio
RtF-HERA [CHK+21]	2^{16}	142	1.24
RtF-Rubato [HKL+22]		71.1	1.31
This work		25.7	1.58

Table 1. Comparison between different transciphering schemes. Here N denotes the degree of the output RLWE ciphertext, and latency denotes the total transciphering time including online and offline phases of the client and server. The expansion ratio is the ratio between the ciphertext bit-size and the final plaintext precision multiplied by the number of slots. The first timings are borrowed from [CHK+21, HKL+22] and all timings are for single-thread implementations parametrized to encrypt 16 messages at once.

Although we focus on the CKKS scheme, we note that our techniques are also applicable to ring packing to RLWE formats that correspond to the BFV and BGV schemes and to RLWE formats that do not necessarily enable fully homomorphic encryption. Also, we chose to focus on the transciphering application, but note that the algorithmic improvements are also beneficial to the other applications mentioned above.

Technical overview. Before delving into the technical ingredients of our ring packing algorithm, we take a step back and discuss the definition of ring packing.

Which ring packing? There have been several works on packing LWE ciphertexts into RLWE ciphertexts [CGGI17, MS18, BGGJ20, LHH$^+$21, CDKS21]. Even though their high-level goals are similar, the ring packing task is underspecified and covers several application scenarios. It is not a priori obvious which RLWE degree N and modulus q should be targeted, and whether one should aim at slots-encoding or coefficients-encoding for the RLWE ciphertexts. Other parameters include the LWE dimension and modulus, as well as the number of ciphertexts to be ring-packed. This flexibility also makes it difficult to compare the various methods. For instance, the ring packing methods from [CGGI16, CDKS21, MS18] are for coefficients-encoding and small RLWE moduli. On the other hand, the ring packing methods from [BGGJ20, LHH$^+$21] are designed for slots-encoding and large RLWE moduli.

A very important scenario for ring packing, encompassing all applications listed earlier, is to use the RLWE-formats for efficient homomorphic computations. Slot-encoding then seems a preferable choice as it enables SIMD operations. Very often, the aim is to be able to perform considerable computations on RLWE encryptions, which means the packed RLWE encryption should enjoy fully homomorphic computations. For example, consider the application of ring packing to bridging FHE computations on LWE-format ciphertexts and RLWE-format ciphertexts. Data stored in LWE-format ciphertexts can conveniently be manipulated in an atomic manner, and RLWE-format FHE schemes allow higher throughput thanks to the ring structure. With efficient ring packing methods, we can aggregate LWE ciphertexts into an RLWE ciphertext and utilize the efficient, structured computations of RLWE FHE schemes.

FHE ring packing. For the reasons mentioned above, the most desirable target for ring packing in the context of FHE computations is to obtain an RLWE-format ciphertext for parameters that support SIMD fully homomorphic computations with bootstrapping. The bootstrappability forces the RLWE degree to be sufficiently high. Also, such ring packing should enable computations immediately after the ring packing, without further processing. All known RLWE-format FHE schemes that support SIMD computations are leveled [BGV14]: ciphertexts are defined with respect to a modulus that belongs to a chain of moduli $Q_0 < Q_1 < \ldots$. A ciphertext is at the largest modulus $Q_{\mathtt{comp}}$ right after bootstrapping and its modulus moves down while performing homomorphic computations. It eventually reaches the smallest modulus $Q_{\mathtt{refresh}}$ that can be bootstrapped. In the context of FHE computations, the aim of ring packing should

be to *output slots-encoded RLWE-format FHE ciphertexts in modulus $Q_{\texttt{comp}}$ and a bootstrappable degree.*

Existing ring packing approaches. In ring packing, we are given several LWE-format ciphertexts $\mathbf{c}_i \in \mathbb{Z}_q^{K+1}$ for a common key $\mathbf{s}$: they satisfy $\mathbf{c}_i \cdot \mathbf{s} = m_i \bmod q$ for some message m_i. The goal is to obtain an RLWE-format ciphertext whose underlying plaintext polynomial contains the m_i's. Concretely, we aim at evaluating $\mathbf{C} \cdot \mathbf{s}$, where the matrix $\mathbf{C}$ whose rows are the $\mathbf{c}_i$'s is viewed as a plaintext, the LWE key $\mathbf{s}$ is given encrypted in RLWE format and the resulting vector should be encoded in RLWE format. In short, ring packing is an instance of (plaintext matrix)-(ciphertext vector) homomorphic multiplication.

Diverse approaches have been proposed for this task. In [CGGI17, MS18, BGGJ20], each entry in the key $\mathbf{s}$ is encrypted in RLWE format, each column $\mathbf{c}_j$ of $\mathbf{C}$ is interpreted as a polynomial, and one evaluates the weighted sum $\sum_j \mathbf{c}_j \cdot s_j$. This is an adaptation of the column method from [HS14] for matrix-vector multiplication. In the row method of [CDKS21], each column $\mathbf{c}_i$ of $\mathbf{C}$ is interpreted as a polynomial and the secret key $\mathbf{s}$ is stored in a single RLWE ciphertext; the ring automorphisms are used to remove the superfluous terms obtained when multiplying the polynomials corresponding to $\mathbf{c}_i$ and $\mathbf{s}$. In [LHH+21], the authors store the diagonals of $\mathbf{C}$ in polynomials and the secret key $\mathbf{s}$ is also stored in a single RLWE ciphertext; they use the SIMD property of RLWE encryption to compute $c_{ij} \cdot s_j$ for many j's in parallel. This is an extension of the diagonal method for matrix-vector multiplication described in [HS14] and attributed therein to Dan Bernstein.

Our approach. Contrary to the most recent works on ring packing [CDKS21, LHH+21], our technique is based on the column method. We improve it to an extent that makes it outperform the strategies based on the row and diagonal methods.

Our first optimization stems from boostrapping itself. Indeed, bootstrapping includes steps that take a low-modulus coefficients-encoded ciphertext and convert it to a high-modulus slots-encoded one. The sequence of relevant components of bootstrapping is called HalfBoot in [CHK+21], where it was used in the context of converting outputs of BFV computations to CKKS ciphertexts. This strategy (which we rename HalfBTS for consistency with our notations) is motivated by the fact that bootstrapping is less costly than known ring packing methods to a high modulus such as $Q_{\texttt{comp}}$. We hence ring-pack to coefficients-encoded RLWE-ciphertexts at the smallest modulus that can be handled by HalfBTS (which is even lower than $Q_{\texttt{refresh}}$) and then HalfBTS to the target modulus. We stress that *FHE ring packing* is reduced to a *base ring packing* to a coefficients-encoded RLWE ciphertext with a small modulus. The rest of our improvements concern this base ring packing.

Secondly, by using the ring switching technique from [GHPS13] , we decouple the ring packing dimension from the FHE dimension, and ring-pack from smaller LWE dimensions to smaller RLWE degrees: the only restriction is that the dimensions/degrees are high enough to provide security. As far as we are aware of, in the context of FHE computations, ours is the first work obtaining

a significant practical gain by using the ring switching technique (see, e.g., the discussion in [HS21, Section 8]). These two techniques significantly improve the run-time of the column method.

The techniques above can also be used to improve the row and diagonal methods. We now explain why they benefit the column method the most. Thanks to HalfBTS, the aim is that the output of ring packing is coefficients-encoded at a small modulus. The modulus consumption during the (base) ring packing is therefore a key factor for efficiency, as computations must be performed at a higher modulus to end with the target small modulus. This is amplified by the fact that a higher modulus consumption requires a higher dimension, to maintain security. As the column method does not consume any modulus, it can fully be performed at the lowest modulus. On the other hand, the row method from [CDKS21] requires one rescaling, and the diagonal method from [LHH$^+$21] requires many: the input modulus should be taken larger to anticipate the modulus loss. Finally, we note that the column method also benefits the most from the hoisting technique from [HS18, Section 5].

Reducing memory consumption. The column method still suffers from a significant drawback compared to the other approaches. Because each LWE secret key coefficient s_i is encrypted in an RLWE ciphertext, the whole key material requires a considerable amount of memory. To mitigate this, we propose *the block method*, which relies on ciphertexts in module-learning-with-errors (MLWE) format [BGV14, LS15].

We proceed as follows. Instead of decomposing the $N \times N$ matrix $\mathbf{C}$ into N columns and mapping each column to a degree N polynomial, we view it as a $\sqrt{N} \times \sqrt{N}$ matrix of blocks of dimensions $\sqrt{N} \times \sqrt{N}$. Each one of these blocks is associated with an MLWE ciphertext with dimension N. The key $(s_1, \ldots, s_N)$ is first decomposed into $\sqrt{N}$ segments of length $\sqrt{N}$, each segment of which can be stored in a single RLWE switching key of degree N. Each row of blocks from $\mathbf{C}$ is then packed into a single block by using module key switching, leading to an $N \times \sqrt{N}$ matrix that is encrypted under a temporary key. We then complete the process using the column method with $\sqrt{N}$ RLWE switching keys of degree N. Overall, the size of the key material has decreased from N RLWE ciphertexts of degree N to $2\sqrt{N}$ of them. Our method relies on an efficient key switching procedure for MLWE ciphertexts, which extends the LWE-to-LWE key switching from [CDKS21].

The method can be extended to multiple levels of recursion, leading to smaller key material sizes. The block method with t levels of recursion requires $tN^{1/t}$ RLWE switching keys of degree N. The column method is the case of $t = 1$, requiring N switching keys. The simplest block method, with $t = 2$, requires $2\sqrt{N}$ switching keys.

Transciphering. The existing FHE schemes suffer from high ciphertext expansion, with the notable exception of RLWE-based schemes at the smallest modulus, which encodes N messages into two degree N polynomials (the message precision is a little lower than the modulus, adding to the expansion factor). To further reduce this expansion factor, it is possible to implicitly represent the first

polynomial as the output of an extendable output-format function (XOF) on a public seed. However, the granularity is very coarse, as a small expansion factor is achieved only if one starts with as many messages as the degree (e.g., set to 2^{15} or 2^{16} to enable bootstrapping in CKKS). The transciphering framework solves this problem using conversion between symmetric ciphers and homomorphic encryption schemes. In the context of CKKS, the state of the art transciphering approaches [HKL+22, CHK+21] rely on the Real-to-Finite-Field (RtF) transciphering framework from [CHK+21]. These works design stream ciphers for real numbers. The key stream is homomorphically evaluated with the BFV scheme (to perform finite field operations); the encrypted data is then added to the BFV encryption of the key stream to obtain a BFV encryption of the plaintext polynomial corresponding to the data; and then $\mathsf{HalfBTS}$ allows to obtain a slots-encoded CKKS ciphertext. This approach is quite heavy for the server, which has to run expensive BFV homomorphic computations. Further, the ciphers are specifically designed for this task, and their security is not well-established yet.

Instead of the RtF framework, we use the transciphering strategy described in [CDKS21], which consists in using an LWE-based symmetric cipher. Transciphering is then exactly ring packing. This approach was inefficient because so was ring packing, but our ring packing algorithm makes it outperform the proposals based on the RtF framework. The LWE ciphertexts are in a small dimension and with a small modulus. As all but one of the coordinates are implicitly represented using a seed, the expansion ratio is limited. Beyond efficiency, this approach achieves high granularity: each message is encrypted individually, as opposed to batches of 16 to 64 messages as in [HKL+22, CHK+21]. This granularity allows the client to send the data whenever it wishes, as it does not need to wait for completing it (alternatively, it could use a block of 16 to 64 messages with only one message, but this would damage the expansion ratio). Finally, the ring packing approach does not require to introduce any new assumption.

2 Preliminaries

For a power-of-2 integer $N \geq 2$, we define the polynomial ring $\mathcal{R}_N = \mathbb{Z}[X]/(X^N + 1)$. It is isomorphic to the ring of integers of the degree-N cyclotomic field. For $q \geq 2$, we define $\mathcal{R}_{q,N} = \mathbb{Z}_q[X]/(X^N + 1) = \mathcal{R}_N/q\mathcal{R}_N$. We will always choose q as a product of primes q_i such that $q_i = 1 \bmod 2N$ for all i. This enables fast multiplication over $\mathcal{R}_{q,N}$, based on the Chinese Remainder Theorem and the Number Theoretic Transform (NTT).

Let $K, q \geq 2$. An LWE_q^K ciphertext with modulus q and dimension K for a secret key $\mathbf{s} \in \mathbb{Z}^{K+1}$ and a plaintext $m \in \mathbb{Z}_q$ is an element $\mathbf{c} \in \mathbb{Z}_q^{K+1}$ such that $\langle \mathbf{c}, \mathbf{s} \rangle = m$. For N a power of 2, an $\mathsf{RLWE}_{q,N}$ ciphertext for a secret key $\mathbf{s} \in \mathcal{R}_N^2$ and a plaintext polynomial $m \in \mathcal{R}_{q,N}$ is an element $\mathbf{c} \in \mathcal{R}_{q,N}^2$ such that $\langle \mathbf{c}, \mathbf{s} \rangle = m$. We let $\mathsf{LWE}_q^K.\mathsf{Enc}(m)$ denote an LWE ciphertext decrypting to plaintext $m \in \mathbb{Z}_q$ and $\mathsf{RLWE}_{q,N}.\mathsf{Enc}(m)$ an RLWE ciphertext decrypting to plaintext $m \in \mathcal{R}_{q,N}$. We suppose the plaintexts in the notations contain small errors so that we can omit the contained error term e.

2.1 RLWE key switching

We describe RLWE key switching in terms of its substeps ModUp, $\mathsf{ModDown}$ and $\mathsf{MultSwk}$. Key switching can be based on gadget decomposition and on the use of an intermediate integer. Typically, the de facto choice nowadays is a combination of these from [HK20]. For simplicity, we focus on explaining the case where the gadget decomposition number ($dnum$) is 1. The general case works similarly.

- $\mathsf{ModUp} : \mathcal{R}_{q,N} \to \mathcal{R}_{qp,N}$ takes as input a polynomial in $\mathcal{R}_{q,N}$, embeds it into $\mathcal{R}_N$ by identifying $\mathbb{Z}_q$ with $[-q/2, q/2)$ and then reduces modulo qp.
- $\mathsf{MultSwk} : \mathcal{R}_{qp,N} \times \mathcal{R}^2_{qp,N} \to \mathcal{R}^2_{qp,N}$ takes as inputs a polynomial $\hat{a} \in \mathcal{R}_{qp,N}$ and a switching key $\mathsf{swk} \in \mathcal{R}^2_{qp,N}$, and simply computes $\hat{a} \cdot \mathsf{swk}$. It costs 2 Hadamard multiplications of degree N.
- $\mathsf{ModDown} : \mathcal{R}^2_{qp,N} \to \mathcal{R}^2_{q,N}$ takes as input a ciphertext $\mathsf{ct} \in \mathcal{R}^2_{qp,N}$ and computes an approximate division by p:

$$\mathsf{ct} = (\hat{b}, \hat{a}) \quad \mapsto \quad \left(\frac{\hat{b} - [\hat{b}]_p}{p}, \frac{\hat{a} - [\hat{a}]_p}{p} \right).$$

Given two secret keys s and s', the switching key from s to s' is defined as

$$\mathsf{swk} = (-\hat{a}s' + ps + \hat{e}, \hat{a}) \in \mathcal{R}^2_{qp,N},$$

where $\hat{a}$ is a uniform polynomial in $\mathcal{R}_{qp,N}$ and $\hat{e} \in \mathcal{R}_N$ is random with small-magnitude coefficients. This switching key can be used to convert a ciphertext for s into a ciphertext for s' that encrypts the same plaintext (up to a small error). Let $\mathsf{ct} = (b, a) \in \mathcal{R}^2_{q,N}$ be a ciphertext for the secret key s so that $(b, a) \cdot (1, s) = b + as = m$ for the plaintext m. Key switching on ct with switching key swk is defined as follows:

$$\mathsf{KS}(\mathsf{ct}; \mathsf{swk}) = \mathsf{ModDown}\Big(\mathsf{MultSwk}\big(\mathsf{ModUp}(a), \mathsf{swk}\big)\Big) + (b, 0) \in \mathcal{R}^2_{q,N}.$$

Depending on the context, we sometimes use the notation $\mathsf{KS}_{s \to s'}(\mathsf{ct})$. When applying several automorphisms on a single ciphertext, $\mathsf{ModUp}(a)$ can be shared through key switching for automorphisms so computed only once. On the other hand, when summing up several key switching results, one can add everything before $\mathsf{ModDown}$ to avoid redundant $\mathsf{ModDown}$'s. Such techniques are referred to as hoisting in [HS18]. We introduce a notation for the second technique for later use. Suppose that $\mathsf{ct}_j = (b_j, a_j)$, $\hat{a}_j = \mathsf{ModUp}(a_j)$ for $0 \le j < \ell$ and $b = \sum_{0 \le j < \ell} b_j$. Then we write:

$$\mathsf{KS}((\mathsf{ct}_j)_{0 \le j < \ell}; (\mathsf{swk}_j)_{0 \le j < \ell}) = \mathsf{ModDown}\Big(\sum_{0 \le j < \ell} \mathsf{MultSwk}\big(\hat{a}_j, \mathsf{swk}_j\big) \Big) + (b, 0).$$

2.2 The column method for ring packing

We consider the task of packing N LWE ciphertexts in dimension N into a single RLWE ciphertext for degree N. By default, the column method is designed to

handle as many input ciphertexts as the RLWE degree. We will see later that our algorithm offers more flexibility. By default, the column method output is a coefficients-encoded RLWE ciphertext.

Let $(b_0, \mathbf{a}_0), \ldots, (b_{N-1}, \mathbf{a}_{N-1}) \in \mathbb{Z}_q \times \mathbb{Z}_q^N$ be LWE ciphertexts such that

$$\forall i < N : \; b_i + \langle \mathbf{a}_i, \mathbf{s} \rangle = m_i$$

for some $m_0, \ldots, m_{N-1} \in \mathbb{Z}_q$ and where $\mathbf{s} \in \mathbb{Z}_q^N$ is an LWE secret key. We aim to get an RLWE ciphertext $(b, a) \in \mathcal{R}_{q,N}^2$ such that

$$(b, a) \cdot (1, s) = b(X) + a(X)s(X) = \sum_{i=0}^{N-1} m_i X^i,$$

where s is an RLWE secret key. Write $\mathbf{a}_i = (a_{i,0}, a_{i,1}, \ldots, a_{i,N-1})$ for $i < N$ and let $\alpha_j(X)$ be defined as

$$\alpha_j(X) = a_{0,j} + a_{1,j}X + \cdots + a_{N-1,j}X^{N-1},$$

for all $j < N$. Note that the coefficients of α_j correspond to the jth column obtained by stacking the $\mathbf{a}_i$'s on top of one another. This observation leads to the following equality, where we write $\mathbf{s} = (s_j)_{0 \le j < N}$:

$$\sum_{i=0}^{N-1} b_i X^{N-1} + \sum_{j=0}^{N-1} s_j \cdot \alpha_j(X) = \sum_{i=0}^{N-1} m_i X^i.$$

The column method [CGGI17] regards each s_j as a different secret key and performs a key switching from s_j to the RLWE key s for all $j < N$ and sums the results together. In more detail:

- (Switching key generation) For each j, compute

  $$\mathsf{swk}_j = (-\hat{a}_j s + p s_j + \hat{e}_j, \hat{a}_j) \in \mathcal{R}_{qp,N}^2,$$

  where $\hat{a}_j$ is a uniform polynomial in $\mathcal{R}_{qp,N}$ and $\hat{e}_j \in \mathcal{R}_N$ is random with small-magnitude coefficients.
- (Ring packing) Given the $\alpha_j(X)$'s for $j < N$ and $\sum_{i=0}^{N-1} b_i X^{N-1}$, we compute

  $$\mathsf{modDown}\Big(\sum_{j=0}^{N-1} \mathsf{modup}(\alpha_j(X)) \cdot \mathsf{swk}_j \Big) + \Big(0, \sum_{i=0}^{N-1} b_i X^{N-1}\Big) \in \mathcal{R}_{q,N}^2,$$

  which is an RLWE ciphertext that decrypts to $\approx \sum_{i=0}^{N-1} m_i X^i$.

2.3 The CKKS scheme

We now provide a brief outline of the CKKS fully homomorphic encryption scheme [CKKS17, CHK+18].

Encoding and decoding. In the CKKS scheme, messages are vectors in $\mathbb{C}^{N/2}$, while plaintexts are the elements of $\mathcal{R}_N$. There exists an approximate (scaled) isomorphism between these two spaces called encoding, whose inverse is referred to as decoding. In order to encrypt a message, one first converts the message into a plaintext via the inverse discrete Fourier transform (iDFT) and scales it up; this process is called encode and denoted as Ecd. The other direction maps a plaintext to a message and corresponds to evaluating the discrete Fourier transform (DFT) and scaling the result downwards; this process is called decode and is denoted as Dcd. To be more specific, $\mathsf{Ecd} : \mathbb{C}^{N/2} \to \mathcal{R}_N$ is defined as

$$\boldsymbol{z} \quad \mapsto \quad \lfloor \Delta \cdot \mathsf{iDFT}(\boldsymbol{z}) \rceil$$

where Δ is a scaling factor and $\lfloor \cdot \rceil$ denotes rounding. $\mathsf{Dcd} : \mathcal{R}_N \to \mathbb{C}^{N/2}$ is

$$m(x) \quad \mapsto \quad \frac{1}{\Delta} \cdot \mathsf{DFT}(m).$$

Slots and coefficients. Let $\mathsf{ct} = (b, a) \in \mathcal{R}_{q,N}^2$ be a ciphertext encrypting a plaintext $m \in \mathcal{R}_N$. Since there is an approximately scaled isomorphism between the message and the plaintext polynomial, we can also view ct as encrypting the message $\boldsymbol{z} = \mathsf{Dcd}(m)$. In this work, the *slots* of a ciphertext refer to the slots of $\boldsymbol{z}$, and the *coefficients* of a ciphertext refer to coefficients of m.

Ciphertext levels. Since encoding is a scaled homomorphism, we have an approximate identity

$$\mathsf{Ecd}(\boldsymbol{z}_1) * \mathsf{Ecd}(\boldsymbol{z}_2) \simeq \Delta \cdot \mathsf{Ecd}(\boldsymbol{z}_1 \odot \boldsymbol{z}_2)$$

where $*$ denotes the polynomial multiplication and $\odot$ denotes the componentwise multiplication. Hence, in order to maintain the scale, we should scale the result of every multiplication down by a factor Δ. This process is called rescale and reduces the ciphertext modulus q by a factor Δ: the new modulus is $\approx q/\Delta$. Since homomorphic operations including multiplication and addition can only be defined if two ciphertexts have the same ciphertext modulus, there is a notion of ciphertext level that is associated with its modulus. Concretely, the lowest modulus allowing meaningful decryption is designated as level 0 and we say that each rescale decreases the level by 1. We let Q_{Enc} denote the modulus of level 0 as it is the lowest modulus allowing meaningful encryption/decryption. We let Q_{KS} the modulus of level 1 as it is the lowest modulus at which we can perform key switching and obtain a ciphertext that still correctly decrypts.

CKKS bootstrapping. Each multiplication/rescale consumes one level. When the level reaches to 0, we cannot proceed with further multiplications. Bootstrap is an operation that allows one to raise the modulus while keeping the underlying plaintext almost the same. CKKS bootstrapping consists of four steps:

- Slots-to-coeffs StC is a homomorphic evaluation of the DFT matrix. The coefficients of the resulting ciphertext become the slots of the input ciphertext.

- Modulus raises ModRaise raises the modulus via the canonical embedding, similarly to ModUp (the moduli are distinct, which justifies the different terminology). After mod-raising a ciphertext in $\mathcal{R}_{Q_{\mathsf{Enc}},N}$ with underlying m, one obtains a ciphertext in $\mathcal{R}_{Q_{\mathsf{top}},N}$ for a larger modulus Q_{top} with underlying plaintext equal to $m + Q_{\mathsf{Enc}}I$ for some small $I \in \mathcal{R}_N$.
- coefficients-to-slots CtS is a homomorphic evaluation of the iDFT matrix. The slots of the resulting ciphertext become the coefficients of the input ciphertext. In particular, this places the $Q_{\mathsf{Enc}}I$ terms in the slots.
- Modulo evaluation EvalMod removes the $Q_{\mathsf{Enc}}I$ part by homomorphically evaluating (an approximation of) the modular reduction function.

Since CtS and EvalMod consume modulus, the ciphertext modulus Q_{comp} after bootstrapping is smaller than the modulus Q_{top} of the output of ModRaise. Similarly, as StC consumes modulus, the ciphertext modulus Q_{refresh} before bootstrapping is larger than the modulus Q_{Enc} of the input of ModRaise. We provide basic information about a typical bootstrapping parameter set in Table 2.

N	$\log_2(QP)$	$\log_2(Q_{\mathsf{Enc}})$	$\log_2(Q_{\mathsf{refresh}})$	$\log_2(Q_{\mathsf{comp}})$	$\log_2(Q_{\mathsf{top}})$	T_{BTS}	Key size
2^{16}	1555	58	184	562	1258	27.4s	667MB

Table 2. Data on the FGb parameter preset of the HEaaN library. Here $\log_2(QP)$ denotes the size of the largest ciphertext modulus that can be used with the parameters while maintaining the desired security. The bootstrapping time T_{BTS} is for a single thread and refers to the situation where all slots are used for data. The key size refers to the total size of the keys needed to bootstrap (the 'a' parts of the key components are represented with a seed, and an integer modulo q is represented on $\log_2 q$ bits).

We note that when we start from a ciphertext whose data is packed into coefficients, we no longer need StC at the beginning. Following [CHK+21], we refer to bootstrapping without StC as HalfBTS.

Ring degrees. There are several functionalities in the CKKS scheme. The simpler ones can be implemented with smaller ring degrees. We define N_{Enc} and N_{KS} as ring degrees that support RLWE encryption (with correct decryption) and RLWE key switching, respectively. We let N_{BTS} be a ring degree that supports CKKS bootstrapping. There are several possible choices for these degrees, but one typically would set $N_{\mathsf{Enc}} \leq N_{\mathsf{KS}} \leq N_{\mathsf{BTS}}$. For a message precision of 20 bits, we can set $N_{\mathsf{Enc}} = 2^{11}$ and $N_{\mathsf{KS}} = 2^{12}$. Among several options for N_{BTS}, we choose 2^{15} or 2^{16} throughout the paper.

3 Accelerating FHE Ring Packing

We present two techniques to accelerate FHE ring packing. The first one allows us to reduce FHE ring packing to a base ring packing towards coefficients-encoding RLWE ciphertexts with small modulus. The second one allows decreasing the RLWE ring degree.

3.1 Moduli optimization

The cost of homomorphic computations highly depends on the working modulus. The modulus is much larger for high CKKS levels than for low levels, and so is the cost of computations. For instance, it is folklore to optimize homomorphic computations by scheduling the moduli of FHE ciphertexts. However, moduli optimization usually does not consider using bootstrapping since bootstrapping is much heavier than other homomorphic operations.

This convention is not justified when evaluating circuits as heavy as bootstrapping, or more. In that case, we argue that bootstrapping should be considered in the computing schedule. Indeed, for massive circuits, the improvement from lowering the moduli can overwhelm the cost of bootstrapping. In our ring packing scenario, this approach allows us to aggressively optimize the modulus.

Moduli-optimized ring packing. Instead of performing ring packing in high-enough moduli to complete the computation at Q_{comp}, we perform ring packing in the lowest modulus Q_{Enc} and for coefficients-encoding, and then run HalfBTS to increase the modulus. As Q_{Enc} is much lower than Q_{comp}, the run-time saving on ring packing itself is very significant. Note that packing to coefficients rather than slots allows omitting StC during the bootstrapping, implying that HalfBTS suffices. In summary, we perform FHE ring packing as follows:

1. Ring-pack to a coefficients-encoding RLWE ciphertext with modulus Q_{Enc}. We call this step *base ring packing*.
2. Run HalfBTS.

Our moduli optimization strategy significantly improves the computational cost and key size of ring packing methods. These improvements come from the fact that ring packing is a more massive computation than bootstrapping. This aggressive moduli optimization is the most effective technique among those we introduce to decrease the runtime of ring packing.

We now explain why ring packing is a larger circuit than HalfBTS, which explains why performing it at a high level is so costly. All known ring packing methods are instances of (plaintext matrix)-(ciphertext vector) homomorphic multiplication. Further, the plaintext matrix is unstructured, making it difficult to optimize the matrix-vector product with special techniques such as decomposition to sparse matrices. HalfBTS also contains a (plaintext matrix)-(ciphertext vector) homomorphic multiplication, in its CtS step. But it can be efficiently evaluated, even if it occurs at a very high modulus, thanks to the algebraic structure of the plaintext matrix: it corresponds to an inverse DFT and can be decomposed as a product of sparse matrices. The rest of HalfBTS, namely ModRaise and EvalMod, can also be efficiently computed.

The aggressive moduli optimization is a general strategy that may be used to improve all ring packing methods (i.e., the column, row, and diagonal methods). The extent of the speed-up depends on how much the modulus can be lowered. This is why our aggressive moduli management benefits the column method the most, as shown in Table 9. The column method can be fully run at the lowest

modulus, while the others should start at a little larger moduli since their level consumption is non-zero. This makes the column method competitive with the others. However, the large key size remains an issue, which we will mitigate in Section 4.

3.2 Ring switching

Ring switching was introduced in [BGV14, Section 5.5] and further studied in [GHPS13]. It allows conversions between RLWE ciphertexts for rings of different degrees when the corresponding number fields are subfields of one another. In these works, the main focus is to lower the ring degree, but we will also switch to higher degree rings.

Ring switching revisited. We use ring switching for coefficients-encoded CKKS, while previous works mostly focus on slots-encoded BGV. This is because we utilize ring switching to optimize the base ring packing into coefficients-encoded CKKS.

Ring switching via modules. We first introduce a different view of ring switching via modules. When we have a ring extension, the large ring can be viewed as a module over the subring. Assume that we are given two powers of 2 integers, k and N, such that k divides N. Then we can view $\mathcal{R}_{q,N}$ as a rank-k $\mathcal{R}_{q,N/k}$-module. Let X_N and $X_{N/k}$ respectively denote the indeterminates corresponding to $\mathcal{R}_{q,N}$ and $\mathcal{R}_{q,N/k}$. By setting $\beta_i = X_N^i$ for all $i < k$, we obtain an $\mathcal{R}_{q,N/k}$-basis of $\mathcal{R}_{q,N}$: concretely, we have $\mathcal{R}_{q,N} \simeq \sum_{0 \le i < k} \mathcal{R}_{q,N/k} \cdot \beta_i$. We define the module decomposition map $\pi_{q,N}^k : \mathcal{R}_{q,N} \to (\mathcal{R}_{q,N/k})^k$ as $\pi_{q,N}^k(a) = (a_0, \cdots, a_{k-1})$, where $a = \sum_j a_j \beta_j$. Note that $\pi_{q,N}^k$ is an $\mathcal{R}_{q,N/k}$-linear. We also consider the ring embedding $\iota_{q,N}^k : \mathcal{R}_{q,N/k} \to \mathcal{R}_{q,N}$ defined by $\iota_{q,N}^k(X_{N/k}) = X_N^k$ and extended as an $\mathcal{R}_{q,N/k}$-homomorphism.

Consider a ciphertext $(b, a) \in \mathcal{R}_{q,N}^2$, with $b = -as + m$ and $s \in \mathcal{R}_{q,N/k}$. We have $s = \sum_j s_j \beta_j = s_0 \cdot 1$. We may then rewrite $b = -as + m$ as:

$$\sum_j b_j \beta_j = \sum_j (-a_j) s_0 \beta_j + \sum_j m_j \beta_j.$$

This is equivalent to

$$\forall j < k : \; b_j = -a_j s_0 + m_j.$$

To summarize, if a ciphertext is encrypted with a secret key that belongs to a subring of relative degree k, then the ciphertext can be split into k pieces, each encrypting a part of the plaintext. We call the conversion $(b_j, a_j)_j \mapsto (b, a)$ as Combine and $(b, a) \mapsto (b_j, a_j)_j$ as Split.

One can understand ring switching via Combine and Split. Ring switching to a higher degree is made of Combine to go from lower degree RLWE ciphertexts to a higher degree RLWE ciphertext with the same key, and of a switching key operation to obtain a higher degree key. Ring switching to a lower degree consists

in switching the key of a large-degree RLWE ciphertext to a small-degree key (which belongs to the subring) and then applying Split.

In the case of coefficients-encoded CKKS, the bijection between $(b_j, a_j)_j$ and (b, a) is a direct sum, which means that it is just a rearrangement of coefficients, thus Split and $\mathsf{Combine}$ barely have any computation cost. This is the main improvement of our revisiting of ring switching for coefficients-encoded CKKS. Note that prior works on ring switching to slots-encoded BGV (such as [GHPS13]), in contrast, require multiplication by an appropriate (small) scalar during the ring switching.

Ring switching in ring packing. Most of the existing works on ring packing do not consider parameters that would lead to bootstrappable RLWE ciphertexts. Indeed, their cost would be prohibitive, as this would require a sufficiently high ring packing degree (the ring degree of output RLWE ciphertexts), namely, it should then be no less that N_{BTS}. By using ring switching, we decouple the ring packing degree (which is much smaller) from the bootstrapping degree N_{BTS}.

The basic idea is to ring pack dimension-k LWE ciphertexts at a small modulus q into low-degree (n) RLWE ciphertexts and then combine them into a bootstrappable RLWE ciphertext of larger degree $N = N_{\mathsf{BTS}}$ by using ring switching. In particular, we first perform multiple base ring packings in the low parameters, yielding lower-degree RLWE ciphertexts. After that, we use ring switching to combine RLWE ciphertexts into a single RLWE ciphertext with a higher degree. We can then switch to slots-encoding at a high modulus Q by using $\mathsf{HalfBTS}$, as explained above. The overall procedure can be summarized as follows.

1. Base-ring-pack from LWE ciphertexts to small-degree RLWE ciphertexts:
$$N \cdot \mathsf{LWE}_q^k \xrightarrow{\text{Base ring pack}} N/n \cdot \mathsf{RLWE}_{q,n}$$
2. Ring-switch, to combine the small-degree RLWE ciphertexts into one:
$$N/n \cdot \mathsf{RLWE}_{q,n} \xrightarrow{\text{Ring switch}} 1 \cdot \mathsf{RLWE}_{q,N}$$
3. $\mathsf{HalfBTS}$ to higher modulus:
$$1 \cdot \mathsf{RLWE}_{q,N} \xrightarrow{\mathsf{HalfBTS}} 1 \cdot \mathsf{RLWE}_{Q,N}$$

By decoupling the base FHE ring packing dimension from N_{BTS}, our strategy substantially improves the computation and memory costs of FHE ring packing algorithms. Dividing the large ring packing into small ring packings does improve the speed and key size. Indeed, performing N/n ring packings to a small degree n is faster than a single ring packing to a large degree N, and the key size for a small ring packing is much smaller than it is for a large ring packing. The improvement is illustrated in Table 9.

We can lower the degree as long as RLWE remains secure under the current modulus. In particular, the only restriction is the security for the switching keys

with respect to their moduli and lowered degrees. Thanks to the moduli optimization in Section 3.1, we perform base ring packing at the lowest modulus applicable, which means we can lower the base ring packing dimension meaningfully, even to N_{Enc}.

4 HERMES

The column method with the optimizations in Section 3 achieves a practical runtime. However, it still suffers from the requirement of a large key size. In this section, we propose a new ring packing algorithm, HERMES, which uses a substantially smaller key size but has a comparable running time with the optimized column method.

Intuitively, HERMES is a *block method*. Understanding ring packing as an instance of (plaintext matrix)-(ciphertext vector) multiplication, we classified the algorithms based on the way to store the plaintext matrix $\mathbf{C}$. Specifically, the column, row, and diagonal method stores each column, row, and diagonal of $\mathbf{C}$, respectively. For HERMES, we consider *subblocks* of $\mathbf{C}$. More concretely, we view $\mathbf{C}$ as a matrix of (rectangular) blocks consisting of N_{KS} entries each. The size of each subblock should be larger or equal to N_{KS} for the security, and less or equal to N_{KS} for the efficiency. We then introduce *module packing* to pack each row of blocks into a single block, resulting in reducing the width of $\mathbf{C}$. By recursively doing this, $\mathbf{C}$ finally would be reduced into a single column, which corresponds to a RLWE instance.

In order to store each block, we use ciphertexts in a module-LWE (MLWE) format, which is a generalization of LWE and RLWE. In the simplest case with only two levels of recursion, HERMES consists of two steps: (1) store the LWE instances in MLWE formats and *module pack* them into a smaller number of MLWE formats with a lower rank, and (2) *module pack* the MLWE formats into a single MLWE format of rank 1 which is an RLWE format. Since each step consists of module packing with $\sqrt{K}$ elementary switching keys, we significantly reduce the overall key size compared to the column method, from K to $2\sqrt{K}$ elementary switching keys. Here K is the dimension of input LWE ciphertexts.

As module packing is a generalization of ring packing, we aim to utilize the techniques of the column method. However, the ingredients (i.e., key switching and ring switching) of the column method cannot be used for MLWE formats directly. To this end, we propose MLWE key switching and MLWE ring switching in Section 4.2. By using the generalized ingredients, we finally describe $\mathsf{ModPack}$, an algorithm for module packing, in Section 4.3, and its use for HERMES in Sections 4.4 and 4.5.

For ease of discussion, in Sections 4.2, 4.3 and 4.4, we mostly focus on the base ring packing problem, assuming the output modulus is Q_{Enc} and the output RLWE degree is the lowest possible, excluding the moduli optimization and the ring switching between RLWE formats introduced in Section 3. In Section 4.5, we put it all together to solve the FHE ring packing problem, introducing HERMES.

All throughout this section, we assume that ranks and degrees (N, k, ℓ, n, k' and k_i) are powers of 2, except for the input LWE dimension K which should be a multiple of a sufficiently large power of 2. Further, we use a few functions to manipulate module elements. To ease the reading, we provide a function list in Appendix A, with domains, ranges and definition locations.

4.1 Module-LWE (MLWE)

We use bold font to indicate a vector and use square brackets to index ciphertext components. We also use subscripts for indexing an element of a vector, especially in the module context. We briefly review Module-LWE (MLWE) which was introduced in [BGV14, LS15]. Let $q \geq 2$ and k, n be powers of 2. An MLWE ciphertext with modulus q, rank k and degree n is denoted as $\mathsf{MLWE}^k_{q,n}$. It is an element $\mathbf{c} = (\mathbf{c}[0], \mathbf{c}[1], \cdots, \mathbf{c}[k]) \in (\mathcal{R}_{q,n})^{k+1}$. We also use the notation of $\mathbf{c} = (b, \mathbf{a})$ where $b = \mathbf{c}[0]$ and $\mathbf{a} = (\mathbf{c}[1], \cdots, \mathbf{c}[k])$. If $\langle \mathbf{c}, (1, \mathbf{s}) \rangle = b + \langle \mathbf{a}, \mathbf{s} \rangle = m$, then the ciphertext $\mathbf{c}$ is said to be an encryption of plaintext $m \in \mathcal{R}_{q,n}$ with secret key $\mathbf{s} = (s_1, \cdots, s_k) \in (\mathcal{R}_{q,n})^k$. To simplify the notations, we assume that the error is contained in m. We call $N = n \cdot k$ the dimension of the ciphertext. MLWE can be seen as a generalization of LWE and RLWE, as $\mathsf{MLWE}^k_{q,1} = \mathsf{LWE}^k_q$ and $\mathsf{MLWE}^1_{q,n} = \mathsf{RLWE}_{q,n}$.

4.2 Building blocks for ModPack

We describe two building blocks for ModPack: MLWE ring switching and MLWE key switching. Namely, we generalize Ring switching (Section 3.2) and key switching (Section 2.1) to MLWE.

MLWE ring switching. Suppose ℓ, k, n are powers of 2 and $(\mathbf{c}_j)_{0 \leq j < \ell}$ are MLWE ciphertexts of rank k and degree n, encrypted with the same secret key $\mathbf{s}$. Define $\mathbf{c}' = \mathsf{Combine}^k_{n,\ell}((\mathbf{c}_j)_{0 \leq j < \ell})$ as

$$\mathbf{c}'[t] = (\pi^\ell_{q,n\ell})^{-1}((\mathbf{c}_j[t])_{0 \leq j < \ell})$$

for each $0 \leq t \leq k$. Then $\mathbf{c}'$ is an MLWE ciphertext of rank k and degree $n\ell$, encrypting the combined input data (i.e., its decryption contains the decryptions of the $\mathbf{c}_j$'s).

MLWE key switching. MLWE key switching is a technique to switch the key of an MLWE ciphertext using a RLWE key switching as described in Section 2.1. The method generalizes the LWE-to-LWE key switching from [CDKS21]. It consists of the following three steps:

1. Embed the input MLWE ciphertext as part of an RLWE ciphertext of the same dimension but a higher degree;
2. Switch the key of the RLWE ciphertext;

3. Extract an MLWE ciphertext that encrypts valid data from the RLWE ciphertext.

These steps will be specified below. To describe them, we need to introduce some notations. We use the same notations as in Section 3.2. Additionally, we define $\epsilon^k_{q,N} : \mathcal{R}_{q,N} \to \mathcal{R}_{q,N/k}$ as $\epsilon^k_{q,N}(a) = a_0$: the $\epsilon^k_{q,N}$ map is extracting the first coefficient of an element of $\mathcal{R}_{q,N}$ viewed as an $\mathcal{R}_{q,N/k}$-module.

Example. *For $N = 8$ and $a = \sum_j a_j X_8^j \in \mathcal{R}_{q,8}$, we have*

$$\pi^2_{q,8}(a) = (a_0 + a_2 X_4 + a_4 X_4^2 + a_6 X_4^3,\ a_1 + a_3 X_4 + a_5 X_4^2 + a_7 X_4^3).$$

Further, we have $\epsilon^4_{q,8}(a) = a_0 + a_4 X_2$.

For $\mathbf{a} = (a_0, \cdots, a_{k-1}) \in (\mathcal{R}_{q,N/k})^k$, we define $\mathbf{a}^{tw}$, the twist of $\mathbf{a}$, as follows:

$$\mathbf{a}^{tw} := (a_0, a_{k-1} \cdot X_{N/k}^{-1}, a_{k-2} \cdot X_{N/k}^{-1}, \cdots, a_1 \cdot X_{N/k}^{-1}).$$

We define the inverse of the twist as

$$\mathbf{a}^{tw^{-1}} := (a_0, a_{k-1} \cdot X_{N/k}, a_{k-2} \cdot X_{N/k}, \cdots, a_1 \cdot X_{N/k}).$$

It may be checked that $\mathbf{a}^{tw^{-1} \circ tw} = (\mathbf{a}^{tw})^{tw^{-1}} = \mathbf{a}$.

Similarly, we can define the twist and its inverse for $a \in \mathcal{R}_{q,N}$ when a rank k is specified:

$$a^{tw,k} = (\pi^k_{q,N})^{-1} \circ (\pi^k_{q,N}(a))^{tw}.$$

Then we observe the following.

Lemma 1. *For $a, s \in \mathcal{R}_{q,N}$, we have*

$$\epsilon^k_{q,N}\left(a^{tw,k} \cdot s\right) = \left\langle \pi^k_{q,N}(a), \pi^k_{q,N}(s) \right\rangle,$$

where $\langle \cdot, \cdot \rangle$ refers to the formal inner product over $(\mathcal{R}_{q,N/k})^k$.

Proof. Write $\pi^k_{q,N}(a) = (a_0, a_1, \cdots, a_{k-1})$ and $\pi^k_{q,N}(s) = (s_0, s_1, \cdots, s_{k-1})$ with $a_i, s_i \in \mathcal{R}_{q,N/k}$ for all i. Then we have

$$\epsilon^k_{q,N}\left(a^{tw,k} \cdot s\right) = \epsilon^k_{q,N}\left(a^{tw,k} \cdot \sum_{j=0}^{k-1} s_j X_N^j\right) = \sum_{j=0}^{k-1} \epsilon^k_{q,N}\left(a^{tw,k} \cdot s_j X_N^j\right).$$

As $a^{tw,k} = a_0 + \sum_{1 \le j < k} a_{k-j} X_{N/k}^{-1} X_N^j$, we have $\epsilon^k_{q,N}\left(a^{tw,k} \cdot s_j X_N^j\right) = a_j s_j$ for each $0 \le j < k$, which completes the proof. □

We now explain how to view an MLWE ciphertext as an RLWE ciphertext (embedding) and the opposite (extraction). In general, we can view an MLWE ciphertext as an MLWE ciphertext with the same dimension but with a higher degree. In that case, as depicted in Fig. 1, the plaintext of the higher degree

MLWE ciphertext includes the data of the lower degree one and the other plaintext slots are filled with random data. Also, we can extract a lower degree MLWE ciphertext from a higher degree MLWE ciphertext. The MLWE embedding map $\mathsf{Embed}^k_{q,N} : (\mathcal{R}_{q,N/k})^{k+1} \to (\mathcal{R}_{q,N})^2$ embeds an MLWE ciphertext of rank k and degree N/k into an RLWE ciphertext with dimension N. It is defined as

$$\mathsf{Embed}^k_{q,N}(b, \mathbf{a}) = \left(\iota^k_{q,N}(b), (\pi^k_{q,N})^{-1}(\mathbf{a}^{tw})\right),$$

where $\iota^k_{q,N} : \mathcal{R}_{q,N/k} \to \mathcal{R}_{q,N}$ is the ring embedding and $\pi^k_{q,N}$ is the module decomposition map (as defined in Section 3.2). Then, by Lemma 1, we have

$$\mathsf{Embed}^k_{q,N}\left(\mathsf{MLWE}^k_{q,N/k}.\mathsf{Enc}_{\mathbf{s}}(m)\right) = \mathsf{RLWE}_{q,N}.\mathsf{Enc}_{(\pi^k_{q,N})^{-1}(\mathbf{s})}(M). \tag{1}$$

for some $M \in \mathcal{R}_{q,N}$ satisfying $\epsilon^k_{q,N}(M) = m$.

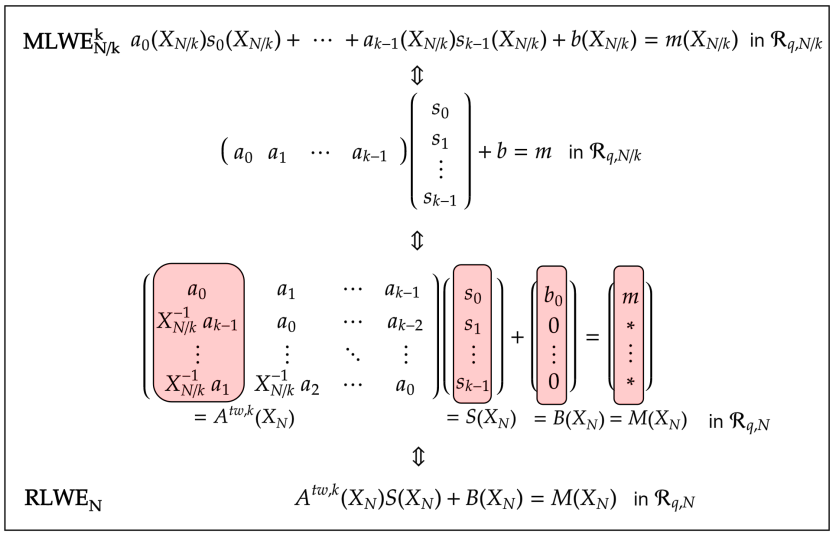

Fig. 1. MLWE embedding and extraction.

The MLWE extraction map $\mathsf{Extract}^k_{q,N} : (\mathcal{R}_{q,N})^2 \to (\mathcal{R}_{q,N/k})^{k+1}$ extracts an MLWE ciphertext of rank k and degree N/k from an RLWE ciphertext of dimension N. It is defined as

$$\mathsf{Extract}^k_{q,N}(\mathcal{B}, \mathcal{A}) = \left(\epsilon^k_{q,N}(\mathcal{B}), (\pi^k_{q,N}(\mathcal{A}))^{(tw,k)^{-1}}\right).$$

Then, by Lemma 1, we have

$$\mathsf{Extract}^k_{q,N}\left(\mathsf{RLWE}_{q,N}.\mathsf{Enc}_s(m)\right) = \mathsf{MLWE}^k_{q,N/k}.\mathsf{Enc}_{\pi^k_{q,N}(s)}(\epsilon^k_{q,N}(m)). \tag{2}$$

With Equations (1) and (2), we obtain the MLWE key switching procedure (Theorem 1). First, we view the input MLWE ciphertext as a part of an RLWE ciphertext. Then we switch the key of the RLWE ciphertext and extract an MLWE ciphertext as the result.

Theorem 1 (MLWE key switching)**.** *Let $q \geq 2$, and k, N be powers of 2 such that $k \leq N$. Suppose that $\mathbf{c} = \mathsf{MLWE}^k_{q,N/k}.\mathsf{Enc}_{\mathbf{s}}(m)$ for some plaintext m. Let $\mathcal{S} = (\pi^k_{q,N})^{-1}(\mathbf{s})$ and $\mathcal{S}' = (\pi^k_{q,N})^{-1}(\mathbf{s}')$. Then the following holds:*

$$\mathsf{Extract}^k_{q,N} \circ \mathsf{KS}_{\mathcal{S}\to\mathcal{S}'} \circ \mathsf{Embed}^k_{q,N}(\mathbf{c}) = \mathsf{MLWE}^k_{q,N/k}.\mathsf{Enc}_{\mathbf{s}'}(m).$$

Proof. Let $\mathbf{C} = \mathsf{Embed}^k_{q,N}(\mathbf{c})$. With Eq. (1), we have $\mathbf{C} = \mathsf{RLWE}_{q,N}.\mathsf{Enc}_{\mathcal{S}}(M)$, where $M \in \mathcal{R}_{q,N}$ and $\epsilon^k_{q,N}(M) = m$. From Eq. (2), we obtain

$$\begin{aligned}\mathsf{Extract}^k_{q,N} \circ \mathsf{KS}_{\mathcal{S}\to\mathcal{S}'}(\mathbf{C}) &= \mathsf{Extract}^k_{q,N}(\mathsf{RLWE}_{q,N}.\mathsf{Enc}_{\mathcal{S}'}(M)) \\ &= \mathsf{MLWE}^k_{q,N/k}.\mathsf{Enc}_{\mathbf{s}'}(m).\end{aligned}$$

This completes the proof. □

4.3 ModPack: an algorithm for module packing

In this section, we describe $\mathsf{ModPack}^{K,k,k'}_N$, an algorithm for module packing, with the following signature:

$$k/k' \cdot \mathsf{MLWE}^K_{q,N/k} \to \mathsf{MLWE}^{k'}_{q,N/k'}. \tag{3}$$

Here K refers to the rank of input MLWE ciphertexts. The degrees of input and output ciphertexts are N/k and N/k', respectively. For simplicity, we consider the case of an input data size that is the same as the output's. Since a degree d MLWE ciphertext has d plaintext slots, the number of input MLWE ciphertexts must be $(N/k')/(N/k) = k/k'$.

Let $\mathbf{s}$ be an $\mathsf{MLWE}^K_{q,N/k}$ secret key and $\mathbf{s}'$ an $\mathsf{MLWE}^{k'}_{q,N/k'}$ secret key. By viewing a K-dimensional vector as a concatenation of K/k' vectors of dimension k', we may write $\mathbf{s} = (\check{\mathbf{s}}_0, \check{\mathbf{s}}_1, \cdots, \check{\mathbf{s}}_{K/k'-1})$ with $\check{\mathbf{s}}_j \in \mathcal{R}^{k'}_{q,N/k}$ for each j. Let $\mathbf{s}_j = \iota^{k/k'}_{q,N/k'}(\check{\mathbf{s}}_j) \in \mathcal{R}^{k'}_{q,N/k'}$, $\mathcal{S}_j = (\pi^{k'}_{q,N})^{-1}(\mathbf{s}_j)$, $\mathcal{S}' = (\pi^{k'}_{q,N})^{-1}(\mathbf{s}')$ and $\mathsf{swk}_j = \mathsf{RLWE}_{qp,N}.\mathsf{Enc}_{\mathcal{S}'}(p \cdot \mathcal{S}_j)$ for $0 \leq j < K/k'$. Given some $\mathsf{MLWE}^K_{q,N/k}$ ciphertexts $(\mathbf{c}_j)_{0\leq j<k/k'}$, where $\mathbf{c}_j$ decrypts to (m_j) under $\mathbf{s}$, $\mathsf{ModPack}^{K,k,k'}_N$ (Algorithm 1) consists of the following steps:

1. $\mathbf{C} \leftarrow \mathsf{Combine}^K_{N/k,k/k'}((\mathbf{c}_j)_{0\leq j<k/k'})$, then $\mathbf{C}$ is an $\mathsf{MLWE}^K_{N/k'}$ ciphertext encrypting
$$m' = (\pi^{k/k'}_{q,N/k'})^{-1}(m_0, \cdots, m_{k/k'-1}) \in \mathcal{R}_{q,N/k'}.$$
2. Write $\mathbf{C} = (\mathbf{C}[t])_{0\leq t\leq K}$ and divide $\mathbf{C}$ into $(\mathbf{C}_j)_{0\leq j<K/k'} \in (\mathcal{R}^{k'+1}_{q,N/k'})^{K/k'}$, where
$$\mathbf{C}_j = (0, \mathbf{C}[jk'+1], \mathbf{C}[jk'+2], \cdots, \mathbf{C}[jk'+k'])$$

for $j \neq 0$ and $\mathbf{C}_0 = (\mathbf{C}[0], \mathbf{C}[1], \cdots, \mathbf{C}[k'])$. Note that each $\mathbf{C}_j$ does not encrypt a valid data, while the decryption of their sum is m' if we assume that each $\mathbf{C}_j$ is encrypted with $\mathbf{s}_j$.

3. Use hoisted MLWE key switching to switch keys, i.e.,

$$\mathbf{C}' = \mathsf{Extract}^{k'}_{q,N} \circ \mathsf{KS}\left(\mathsf{Embed}^{k'}_{q,N}(\mathbf{C}_j)_{0\leq j<K/k'}; (\mathsf{swk}_j)_{0\leq j<K/k'}\right).$$

The output $\mathbf{C}'$ is an MLWE encryption of m' of rank k' and degree N/k', under secret key $\mathbf{s}'$. The algorithm is illustrated in Fig. 2.

Theorem 2 (ModPack). *Let* $\mathbf{s} = (\check{\mathbf{s}}_0, \check{\mathbf{s}}_1, \cdots, \check{\mathbf{s}}_{K/k'-1}) \in \mathcal{R}^K_{q,N/k}$ *with* $\check{\mathbf{s}}_j \in \mathcal{R}^{k'}_{q,N/k}$ *for each* j, *and* $\mathbf{s}' \in \mathcal{R}^{k'}_{q,N/k'}$. *Define* $\mathbf{s}_j = \iota^{k/k'}_{q,N/k'}(\check{\mathbf{s}}_j) \in \mathcal{R}^{k'}_{q,N/k'}$ *and* $\mathcal{S}_j = (\pi^{k'}_{q,N})^{-1}(\mathbf{s}_j)$ *for each* j, *and* $\mathcal{S}' = (\pi^{k'}_{q,N})^{-1}(\mathbf{s}')$. *Given as input* k/k' *MLWE ciphertexts* $\mathbf{c}_j = \mathsf{MLWE}^K_{q,N/k}.\mathsf{Enc}_{\mathbf{s}}(m_j)$ *for some* m_j *and* K/k' *switching keys* $\mathsf{swk}_j = \mathsf{RLWE}_{qp,N}.\mathsf{Enc}_{\mathcal{S}'}(p \cdot \mathcal{S}_j)$, ModPack *returns* $\mathbf{C}'$ *satisfying*

$$\mathbf{C}' = \mathsf{MLWE}^{k'}_{q,N/k'}.\mathsf{Enc}_{\mathbf{s}'}(m'),$$

where

$$m' = (\pi^{k/k'}_{q,N/k'})^{-1}(m_0, \cdots, m_{k/k'-1}) \in \mathcal{R}_{q,N/k'}.$$

Proof. Consider $\mathbf{C} = \mathsf{Combine}^K_{N/k,k/k'}((\mathbf{c}_j)_{0\leq j<k/k'})$ and define $(\mathbf{C}_j)_{0\leq j<K/k'} \in (\mathcal{R}^{k'+1}_{q,N/k'})^{K/k'}$ as in Equation (2) for $j \neq 0$ and $\mathbf{C}_0 = (\mathbf{C}[0], \mathbf{C}[1], \cdots, \mathbf{C}[k'])$. Write $\mathbf{C} = (\mathbf{B}, \mathbf{A})$, $\mathbf{C}_j = (\mathbf{B}_j, \mathbf{A}_j)$, $\mathsf{Embed}^{k'}_{q,N}(\mathbf{C}_j) = (\mathcal{B}_j, \mathcal{A}_j)$, $\mathcal{B} = \sum_{0\leq j<K/k'} \mathcal{B}_j$ and $\hat{\mathcal{A}}_j = \mathsf{ModUp}(\mathcal{A}_j)$ for each j. Then we have

$$\epsilon^{k'}_{q,N}\Big(\sum_{0\leq j<K/k'} \mathcal{A}_j\mathcal{S}_j + \mathcal{B}\Big) = \sum_{0\leq j<K/k'} \langle \mathbf{A}_j, \mathbf{s}_j\rangle + \mathbf{B} = \langle \mathbf{A}, \mathbf{s}\rangle + \mathbf{B} = m'. \quad (4)$$

Also, we have

$$\begin{aligned} &\mathsf{KS}\Big(\mathsf{Embed}^{k'}_{q,N}(\mathbf{C}_j)_{0\leq j<K/k'}; (\mathsf{swk}_j)_{0\leq j<K/k'}\Big) \\ &= \mathsf{ModDown}\Big(\sum_{0\leq j<K/k'} \mathsf{MultSwk}(\hat{\mathcal{A}}_j, \mathsf{swk}_j)\Big) + (\mathcal{B}, 0) \\ &= \mathsf{RLWE}_{q,N}.\mathsf{Enc}_{\mathcal{S}'}\Big(\sum_{0\leq j<K/k'} \mathcal{A}_j\mathcal{S}_j + \mathcal{B}\Big). \end{aligned} \quad (5)$$

From Equations (4), (5) and (2), we obtain

$$\begin{aligned} \mathbf{C}' &= \mathsf{Extract}^{k'}_{q,N}\Big(\mathsf{RLWE}_{q,N}.\mathsf{Enc}_{\mathcal{S}'}\Big(\sum_{0\leq j<K/k'} \mathcal{A}_j\mathcal{S}_j + \mathcal{B}\Big)\Big) \\ &= \mathsf{MLWE}^{k'}_{q,N/k'}.\mathsf{Enc}_{\mathbf{s}'}(m'), \end{aligned}$$

as desired. □

Algorithm 1: $\mathsf{ModPack}_N^{K,k,k'}: k/k' \cdot \mathsf{MLWE}_{q,N/k}^K \to \mathsf{MLWE}_{q,N/k'}^{k'}$

Input : $(\mathbf{c}_j)_{0\le j<k/k'}$
$(\mathsf{swk}_j)_{0\le j<K/k'}$
Output: $\mathbf{C}'$
$\mathbf{C} \leftarrow \mathsf{Combine}_{N/k,k/k'}^K((\mathbf{c}_j)_{0\le j<k/k'})$
$\mathbf{C}_0 \leftarrow (\mathbf{C}[0], \mathbf{C}[1], \cdots, \mathbf{C}[k'])$
for $j = 1, \cdots, K/k'$ **do**
  $\mathbf{C}_j \leftarrow (0, \mathbf{C}[jk'+1], \mathbf{C}[jk'+2], \cdots, \mathbf{C}[jk'+k'])$
end for
$\mathbf{C}' \leftarrow \mathsf{Extract}_{q,N}^{k'} \circ \mathsf{KS}(\mathsf{Embed}_{q,N}^{k'}(\mathbf{C}_j)_{0\le j<K/k'}; (\mathsf{swk}_j)_{0\le j<K/k'})$
return $\mathbf{C}'$

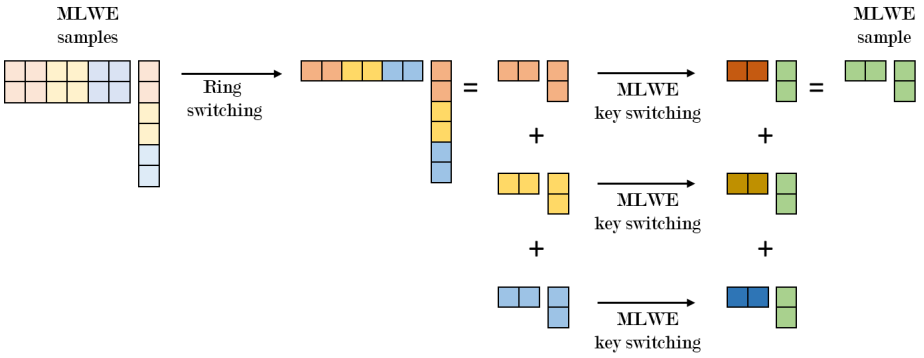

Fig. 2. Visualization of ModPack, which packs multiple MLWE ciphertexts into a single MLWE ciphertext with the same dimension. The entries of each vector belong to quotient polynomial rings. A light color means a ring with a smaller degree while a dark color indicates a ring with a larger degree.

$\mathsf{ModPack}_N^{K,k,k'}$ consists of K/k' executions of ModUp, 1 execution of ModDown and $2K/k'$ Hadamard multiplications. The overall key consists of K/k' elementary switching keys. Each ModUp and ModDown involves $O(N \log N)$ integer operations since degree-N NTT dominates, and a Hadamard multiplication has $O(N)$ complexity. The size of a switching key is $2N \log(qp)$ since it is a degree-N RLWE ciphertext of modulus qp.

Note that $\mathsf{ModPack}_N^{N,N,1}$ is exactly the column method, and hence ModPack can be seen as a generalization of the column method.

4.4 BaseHERMES: a base ring packing with MLWE midpoints

Below, we describe how to reduce the switching key size by using ModPack (Algorithm 1), introducing BaseHERMES, a composition of ModPacks. To compose several ModPacks to form a base ring packing, we insert MLWE midpoints with intermediate degrees to split the ring packing procedure into several steps. Then,

we apply ModPack at each step. We set the rank of the input LWE ciphertexts as $K \geq N_{\mathsf{Enc}}$ and set the number of input LWE ciphertexts as $N \geq N_{\mathsf{KS}}$. The dimension of switching key and the degree of the output RLWE ciphertexts are set to N. Base ring switching and hence $\mathsf{BaseHERMES}_N^{K,\kappa}$ have the following signature:

$$N \cdot \mathsf{LWE}_{Q_{\mathsf{Enc}}}^{K} \to 1 \cdot \mathsf{RLWE}_{Q_{\mathsf{Enc}},N} \tag{6}$$

Beyond the input LWE dimension K and output RLWE degree N, BaseHERMES is also parametrized by some $\kappa = \{k_1 > \cdots > k_t\}$. It refers to the set of ranks of the MLWE midpoints: t is the number of midpoints and k_j is the rank of jth midpoint for all j. We assume that $k_0 = N > k_j > 1 = k_{t+1}$ for all j. In case of $\kappa = \emptyset$, $\mathsf{BaseHERMES}_N^{K,\emptyset}$ is exactly $\mathsf{ModPack}_N^{K,k_0,k_1} = \mathsf{ModPack}_N^{K,N,1}$, which is the column method.

Single MLWE midpoint. $\mathsf{BaseHERMES}_N^{K,\{k\}}$ consists in inserting an intermediate state "$k \cdot \mathsf{MLWE}_{Q_{\mathsf{Enc}},N/k}^{k}$" in (6), i.e., in splitting the base ring packing procedure into two steps:

$$N \cdot \mathsf{LWE}_{Q_{\mathsf{Enc}}}^{K} \xrightarrow{\text{Step 1}} k \cdot \mathsf{MLWE}_{Q_{\mathsf{Enc}},N/k}^{k} \xrightarrow{\text{Step 2}} 1 \cdot \mathsf{RLWE}_{Q_{\mathsf{Enc}},N}. \tag{7}$$

Step 1 is performed with k parallel executions of $\mathsf{ModPack}_N^{K,N,k}$, and Step 2 is itself implemented by calling $\mathsf{ModPack}_N^{k,k,1}$. More compactly, we may write:

$$\mathsf{BaseHERMES}_N^{K,\{k\}} = \mathsf{ModPack}_N^{k,k,1} \circ (k \cdot \mathsf{ModPack}_N^{K,N,k}).$$

The algorithm is illustrated in Fig. 3. Note that once the states are determined, there is a unique ModPack parametrization corresponding to it. The input of the first step consists of the N LWE ciphertexts of dimension K, which may be grouped and ordered arbitrarily, affecting the order of the plaintext slots of the output.

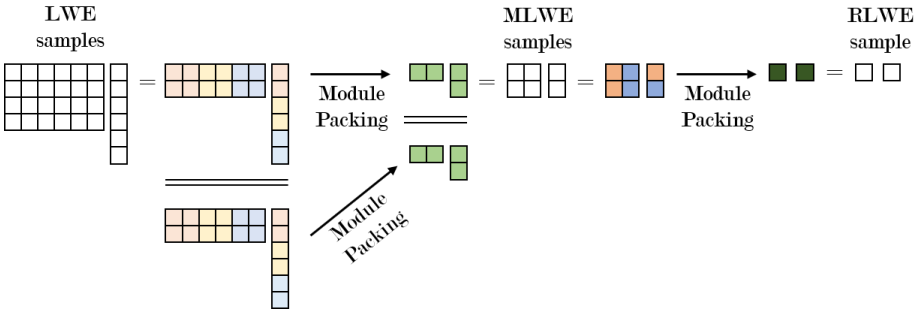

Fig. 3. Visualization of BaseHERMES with a single middle point.

Table 3 compares the costs and sizes of switching keys of the $\mathsf{BaseHERMES}_N^{K,\emptyset}$ (the column method) and $\mathsf{BaseHERMES}_N^{K,\{k\}}$ algorithms. It displays the number

of executions of ModUp and ModDown (each of which has consumes $O(N \log N)$ arithmetic operations), the number of Hadamard multiplications and the number of elementary switching keys (an elementary switching key is a degree-N RLWE ciphertext of modulus Q_{KS} and thus has $N \log(Q_{\mathsf{KS}})$ bit-size). For $K \geq \sqrt{N}$, which is the typical scenario, the value of k minimizing the number of switching keys is $\sqrt{K}$. It is therefore interesting to set k as the nearest power of two.

Method		ModUp $O(N \log N)$	ModDown $O(N \log N)$	HadamardMult $O(N)$	SwitchingKey $O(N \log(Q_{\mathsf{KS}}))$
$\mathsf{BaseHERMES}_N^{K,\emptyset}$ (column method)		K	1	$2K$	K
$\mathsf{BaseHERMES}_N^{K,\{k\}}$ (single midpoint)	Step 1	K	k	$2K$	K/k
	Step 2	k	1	$2k$	k
	Total	$K+k$	$k+1$	$2(K+k)$	$K/k+k$

Table 3. Comparison between $\mathsf{BaseHERMES}_N^{K,\emptyset}$ and $\mathsf{BaseHERMES}_N^{K,\{k\}}$. Each cell provides the number of ModUp/ModDown/HadamardMult/SwitchingKey in each method. For time complexity, we mostly focus on ModUp and ModDown since they have higher cost $O(N \log N)$ than Hadamard multiplication's $O(N)$ cost.

To see the effectiveness of the new method, we consider the case of $K = N$ with $k = \sqrt{N}$ (see Table 4). In this case, the computational complexity difference is negligible compared to the main cost N executions of ModUp. However, the number of elementary switching keys significantly reduces from N for the column method to $2\sqrt{N}$ for the single midpoint method.

Method		ModUp $O(N \log N)$	ModDown $O(N \log N)$	HadamardMult $O(N)$	SwitchingKey $O(N \log(Q_{\mathsf{KS}}))$
$\mathsf{BaseHERMES}_N^{N,\emptyset}$ (column method)		N	1	$2N$	N
$\mathsf{BaseHERMES}_N^{N,\{\sqrt{N}\}}$ (single midpoint)	Step 1	N	$\sqrt{N}$	$2N$	$\sqrt{N}$
	Step 2	$\sqrt{N}$	1	$2\sqrt{N}$	$\sqrt{N}$
	Total	$N+\sqrt{N}$	$\sqrt{N}+1$	$2(N+\sqrt{N})$	$2\sqrt{N}$

Table 4. Comparison between $\mathsf{BaseHERMES}_N^{N,\emptyset}$ and $\mathsf{BaseHERMES}_N^{N,\{\sqrt{N}\}}$.

Generalization. By taking more MLWE midpoints, one can reduce the number of elementary switching keys for a limited slowdown in time performance. In general, we can insert t midpoints of rank k_j satisfying $N = k_0 > k_1 > \cdots > k_t > k_{t+1} = 1$ to build $\mathsf{BaseHERMES}_N^{K,\kappa}$ with $\kappa = \{k_1 > \cdots > k_t\}$. The state diagram of $\mathsf{BaseHERMES}_N^{K,\kappa}$ is as follows.

$$N \cdot \mathsf{LWE}_{Q_{\mathsf{Enc}}}^{K} \xrightarrow{\text{Step 1}} k_1 \cdot \mathsf{MLWE}_{Q_{\mathsf{Enc}},N/k_1}^{k_1} \longrightarrow \cdots$$
$$\longrightarrow k_t \cdot \mathsf{MLWE}_{Q_{\mathsf{Enc}},N/k_t}^{k_t} \xrightarrow{\text{Step } t+1} 1 \cdot \mathsf{RLWE}_{Q_{\mathsf{Enc}},N}.$$

Step 1 is performed with k_1 parallel executions of $\mathsf{ModPack}_N^{K,k_0,k_1}$, whereas Step j for $j > 1$ consists in k_j parallel calls to $\mathsf{ModPack}_N^{k_{j-1},k_{j-1},k_j}$. To sum up, we have

$$\begin{aligned}\mathsf{BaseHERMES}_N^{K,\kappa} =&(k_{t+1} \cdot \mathsf{ModPack}_N^{k_t,k_t,k_{t+1}}) \circ (k_t \cdot \mathsf{ModPack}_N^{k_{t-1},k_{t-1},k_t}) \circ \cdots \\ &\circ (k_2 \cdot \mathsf{ModPack}_N^{k_1,k_1,k_2}) \circ (k_1 \cdot \mathsf{ModPack}_N^{K,k_0,k_1}).\end{aligned}$$

Table 5 gives the costs and the number of elementary switching keys.

Method		ModUp $O(N \log N)$	ModDown $O(N \log N)$	HadamardMult $O(N)$	SwitchingKey $O(N \log(Q_{\mathsf{KS}}))$
$\mathsf{BaseHERMES}_N^{K,\emptyset}$ (column method)		K	1	$2K$	K
$\mathsf{BaseHERMES}_N^{K,\kappa}$ (t midpoints)	Step 1	K	k_1	$2K$	K/k_1
	Step 2	k_1	k_2	$2k_1$	k_1/k_2
	$\cdots$	-	-	-	-
	Step $t+1$	k_t	1	$2k_t$	k_t
	Total	$K + \sum_{1 \le j \le t} k_j$	$\sum_{1 \le j \le t+1} k_j$	$2(K + \sum_{1 \le j \le t} k_j)$	$K/k_1 + \sum_{1 \le j \le t} k_j/k_{j+1}$

Table 5. Comparison between $\mathsf{BaseHERMES}_N^{K,\emptyset}$ and $\mathsf{BaseHERMES}_N^{K,\kappa}$.

Increasing the number of mid-points t allows to significantly reduce key-size. For example, set $k_i = K^{1-i/(t+1)}$. Then the key consists of $(t+1) \cdot K^{1/(t+1)}$ elementary switching keys. The total number of ModUp's and ModDown's is $K + 2\sum_{i=1}^{t} K^{i/(t+1)} + 1$, which is no more than three times larger than that for $\mathsf{BaseHERMES}_N^{K,\emptyset}$. Note that $K + 2\sum_{i=1}^{t} K^{i/(t+1)} + 1 = K + 2\frac{K - K^{1/(t+1)}}{K^{1/(t+1)} - 1} + 1 \le 3K + 1$. Therefore, key size decreases fast with t, whereas the cost at most tripled.

In practice, as seen in Table 10, the single midpoint method already enjoys a very small key compared to the bootstrapping key, and reducing it further does not seem interesting in the FHE ring packing scenario.

4.5 HERMES

With ring switching, HalfBTS and BaseHERMES at hand, we can now describe HERMES. HERMES ring-packs N_{BTS} $\mathsf{LWE}_{Q_{\mathsf{Enc}},N_{\mathsf{Enc}}}^{K}$ ciphertexts into an $\mathsf{RLWE}_{Q_{\mathtt{comp}}}$ ciphertext. It consists of three steps:

1. Group the given N_{BTS} LWE ciphertexts into $N_{\mathsf{BTS}}/N_{\mathsf{KS}}$ groups, each one with N_{KS} LWE ciphertexts; for each group, run $\mathsf{BaseHERMES}_{N_{\mathsf{KS}}}^{K,\kappa}$, for some parameter κ, to obtain $N_{\mathsf{BTS}}/N_{\mathsf{KS}}$ $\mathsf{RLWE}_{Q_{\mathsf{Enc}},N_{\mathsf{KS}}}$ ciphertexts.
2. Ring switch the RLWE ciphertexts to form a single RLWE ciphertext.
3. Run HalfBTS on the RLWE ciphertext to raise the modulus to $Q_{\mathtt{comp}}$.

The algorithm is illustrated in Fig. 4.

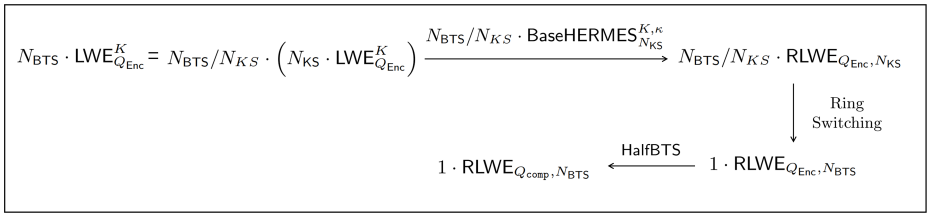

Fig. 4. Visualization of HERMES, which ring-packs N_{BTS} $\mathsf{LWE}^K_{Q_{\mathsf{Enc}}}$ ciphertexts into an $\mathsf{RLWE}_{Q_{\mathsf{comp}},N_{\mathsf{BTS}}}$ ciphertext.

5 Implementation

We provide some proof-of-concept implementation results for our method. We used the C++ **HEaaN** library for the development. All experiments are measured on AMD® Ryzen 7 3700x 8-core processor with a single-threaded CPU.

In our (base) ring packing implementations, N denotes the degree of the output RLWE ciphertext, ℓ the number of input LWE ciphertexts, and K the dimension of the input LWE ciphertexts. The mean (resp. worst) precision is $-\log_2(\mathbb{E}(\|e\|_1/m))$ (resp. $-\log_2 \max \|e\|_\infty$), where $e \in \mathbb{R}^m$ denotes the overall (base) ring packing error; expectation and maximum are taken over 100 iterations for each implementation.

5.1 HERMES Implementation

We first describe our ring packing implementation and parameters. We start with 2^{16} LWE ciphertexts of dimension 2^{11} with scaling factor of $\Delta = 2^{42}$. Our method consists of the following procedures.

1. We either use the column method ($\mathsf{BaseHERMES}^{2^{11},\emptyset}_{2^{12}}$) or the single midpoint method ($\mathsf{BaseHERMES}^{2^{11},\{2^6\}}_{2^{12}}$) and obtain 2^4 RLWE ciphertexts of degree 2^{12}.
2. We use ring switching to combine 2^4 RLWE ciphertexts of degree 2^{12} into a single RLWE ciphertext of degree 2^{16}.
3. We perform HalfBTS, in order to bootstrap the ciphertext to modulus Q_{comp}.

We used the bootstrapping parameter set from Table 2. We let HERMES^0 denote the method based on the column method and HERMES^1 the method based on the single midpoint method. The latency of HERMES^0 and HERMES^1 are 29.1s and 30.7s, respectively, while their total key sizes including ring packing keys and BTS keys are 782MB and 673MB, respectively. We provide detailed information in Table 6.

5.2 Comparison to the state of the art

The only previous work that provides implementation results for ring packing to RLWE slots at Q_{comp} is Pegasus [LHH+21]. We provide a comparison between

	Latency (s)	Key size (MB)	Precision (mean-worst)
HERMES^0	29.1	782	$(23.8, 21.1)$
HERMES^1	30.7	673	$(24.9, 21.8)$

Table 6. HERMES implementation results. The latency includes base ring packing and HalfBTS. The key size includes both ring packing keys and BTS keys. Let $\ell = 2^{16}$ be the number of input LWE ciphertexts.

our method and [LHH+21] in Table 8. The value of $\log_2(Q_{\mathsf{comp}})$ in [LHH+21] is not given explicitly but we estimate from the data in [LHH+21, Section VI] that it is equal to $45 \cdot 6 = 270$. For fair comparison, we constructed a CKKS parameter with $\log_2(Q_{\mathsf{comp}}) = 270$, see Table 7. We use a smaller ring degree N than PEGASUS by using the technique of [BTPH22] and choosing appropriate Hamming weights. The PEGASUS figures in Table 8 are borrowed from [LHH+21, Table V].

N	$(h, \tilde{h})$	$\log_2(QP)$	$\log_2(Q_{\mathsf{Enc}})$	$\log_2(Q_{\mathsf{comp}})$	$\log_2(Q_{\mathsf{top}})$	Key size
2^{15}	$(256, 32)$	820	45	270	765	542MB

Table 7. Data on the parameter set constructed for comparison with PEGASUS. Here h and $\tilde{h}$ denote the Hamming weights of the dense and sparse secret keys, respectively, for the sparse-secret encapsulation technique from [BTPH22]. The column '$\log_2(QP)$' denotes the size of the largest ciphertext modulus that can be used with the parameters while maintaining the desired security. The key size refers to the total size of the keys needed to HalfBTS.

	N	$\log_2(Q_{\mathsf{comp}})$	ℓ	Latency (s)	Amortized time (ms/slot)	Key size (MB)
[LHH+21]	2^{16}	270	2^8	23.4	93.0	1870
			2^{10}	51.6	50.4	3540
			2^{12}	51.7	12.6	
HERMES^1	2^{16}	562	2^8	19.4	75.8	1010
			2^{10}	20.4	19.9	1050
			2^{12}	21.9	5.35	927
			2^{16}	30.7	0.468	673
	2^{15}	270	2^8	6.15	24.0	812
			2^{10}	6.23	6.08	832
			2^{12}	6.68	1.63	718
			2^{15}	10.2	0.311	547

Table 8. Comparison between HERMES and PEGASUS. The key size includes both ring packing keys and BTS keys.

We compare HERMES^1 with $\log_2(Q_{\mathsf{comp}}) = 270$ and [LHH+21]. HERMES^1 is 3.8 to 7.7 times faster for the same value of ℓ. Our throughput may be optimized by choosing $\ell = 2^{15}$, which leads to a factor 41 improvement compared to [LHH+21] with $\ell = 2^{12}$. For $\ell < N$, we used the bootstrapping algorithm for sparsely packed ciphertexts from [CHK+18, Section 5.2].

All the other previous works including [CDKS21] and [CGGI17] should show similar or even slower latency than [LHH+21] because they homomorphically perform a linear transformation with moduli larger than Q_{comp}.

5.3 Impact of our ingredients

We introduced several optimizations including ring switching and modulus optimizations in Section 3. These optimizations are general ones that can be applied to all the base ring packing methods to improve performance. In Section 4, we proposed a method using multiple MLWE midpoints to reduce the key size of [CGGI17] while retaining a similar ring packing time. In this section, we provide some implementation results explaining how effective our ingredients are.

Optimizations. For comparison, we report in Table 9 experimental results for the improved versions of the row method [CDKS21] and the column method [CGGI17]. We first performed 2^4 many base ring packings for each method at dimension 2^{12}, gathered the ciphertexts into dimension 2^{16} via ring switching, and then performed $\mathsf{HalfBTS}$. The improved row and column methods have a latency of 54.3s and 29.1s, respectively, which are fairly fast compared to the implementation of [LHH+21], meaning that our optimizations are indeed effective.

	N	K	T_{RP}	Base ring packing key size	BTS key size
The row method + ring switching + modulus optimization	2^{16}	2^{12}	54.3s	670KB	667MB
The column method + ring switching + modulus optimization (HERMES^0)		2^{11}	29.1s	114MB	

Table 9. Comparison between the improved row and column methods. Here T_{RP} denotes the total ring packing time including $\mathsf{HalfBTS}$.

The improved row method gives a lower key size but is slower, and the column method is faster but has a significantly large key size. The key size of the row method is negligible compared to the BTS key size while it is not for the column method. Note that we used the BTS key reduction algorithm in [HS18], and BTS key size could be reduced even further with [KLK+22]. The trade-off between row and column methods gives a motivation for finding a hybrid method between the two.

MLWE midpoints. In Table 10, we provide a comparison between different base ring packing methods, in terms of time and memory. Our single midpoint method is almost as fast as the improved column method but has a fairly small ring packing key size compared to the improved row method. The latency difference between the row method and the others comes from hoisting and the use of a smaller LWE dimension K. Since the row method with the same $\log_2 Q$ as the other methods gives significantly lower precision, we provide higher precision implementation results as well.

	N	K	$\log_2 Q$	T_{baseRP}	Key size	Precision (mean-worst)
Improved row method	2^{12}	2^{12}	54	1.78s	664KB	(22.5, 20.0)
			72	3.34s	1.33MB	(40.5, 37.9)
$\mathsf{BaseHERMES}_N^{K,\emptyset}$ (improved column method)		2^{11}	54	231ms	113MB	(30.9, 28.4)
$\mathsf{BaseHERMES}_N^{K,\{2^{\lceil \log_2 K/2 \rceil}\}}$ (single midpoint)			54	374ms	5.31MB	(32.8, 30.3)

Table 10. Comparison of different base ring packing methods. Here T_{baseRP} denotes the base ring packing time. The bit size difference between output scaling factor and base modulus is set to 12 bits for fair comparison.

5.4 Transciphering using HERMES

The fundamental goal of the transciphering framework is to decrease the computational and communication costs for a client that sends FHE ciphertexts (e.g., CKKS ciphertexts) to a computing server. Beyond their size, another important drawback of CKKS ciphertexts in this context is their low granularity. They contain thousands of plaintext slots and the client has to wait and send a very large ciphertext if it wants to fully exploit the number of slots to decrease the expansion factor. To handle this, several works have focused on transciphering from a symmetric encryption scheme to CKKS. As CKKS encrypts approximations to real numbers with moderate precision rather than bits or finite field elements, some care is required to obtain efficient transciphering. For this purpose, HERA [HKL+22] and Rubato [CHK+21] rely on the so-called Real-to-Finite-field framework (RtF), using symmetric ciphers with plaintexts of bit-sizes slightly above the CKKS plaintext bit-sizes. The symmetric cipher is homomorphically decrypted with the discrete-plaintext BFV/BGV FHE schemes and the result is then mapped from BFV/BGV to CKKS.

HERMES provides a more direct solution. A client sends the encrypted data in LWE or small-degree MLWE format to achieve high granularity, and it uses an extendable output-format function (XOF) on a public seed to implicitly represent most of the ciphertext and achieve low ciphertext expansion. The server uses HERMES to construct FHE ciphertexts. Our approach directly encrypts approx-

imations to real numbers in the LWE symmetric ciphers, hence allowing to bypass the costly BFV/BGV step in RtF framework. Thus, the efficiency of HERMES significantly reduces the server-side computational overhead even compared to the state-of-the-art transciphering approaches for CKKS [HKL$^+$22, CHK$^+$21]. Figure 5 (inspired from [CHK$^+$21, Figure 2]) illustrates the offline and online phases of the client and server.

Using a XOF on a public seed makes much room for precomputation. Concretely, the $\boldsymbol{a}$ part in (M)LWE ciphertexts $((b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + m, \boldsymbol{a}))$ is computed as $\boldsymbol{a} = \mathrm{XOF}(\mathrm{seed})$ for a public seed. The client can precompute $\boldsymbol{a}$ and $\widehat{b} = \langle \boldsymbol{a}, \boldsymbol{s} \rangle$ using the secret key $\boldsymbol{s}$ and seed, and it computes $b = \widehat{b} + m$ in the online phase. To reduce the communication cost due to b, the client can cut its least significant bits. Concretely, the client sends $\lfloor (2^{\hat{p}}/\Delta) \cdot b \rceil = \lfloor (2^{\hat{p}}/\Delta) \cdot (\widehat{b} + m) \rceil \approx (2^{\hat{p}}/\Delta) \cdot \widehat{b} + \lfloor (2^{\hat{p}}/\Delta) \cdot m \rceil$ instead of b, where $\hat{p}$ is the input precision. The server can scale up the received $\lfloor (2^{\hat{p}}/\Delta) \cdot b \rceil$ by a factor $\Delta/2^{\hat{p}}$, yielding an LWE encryption of m with sufficient precision $\approx \hat{p}$.

The server can precompute the $\boldsymbol{a}$ part of the (M)LWE ciphertexts using seed. It then pre-performs the base ring packing of HERMES for the ephemeral LWE ciphertexts $(0, \boldsymbol{a})$'s, and obtains a precomputed RLWE ciphertext $(\widehat{B}(X), A(X))$. In the online phase, the server first appropriately rearranges the received b parts of input (M)LWE, yielding $\widetilde{B}(X)$. It then completes the ring packing by running HalfBTS on $(\widehat{B}(X) + \widetilde{B}(X), A(X))$. Sending part of the HERMES computation to the offline phase works since through HERMES, the $A(X)$ part of the output RLWE ciphertext depends only on the $\boldsymbol{a}$ parts of input LWE ciphertexts.

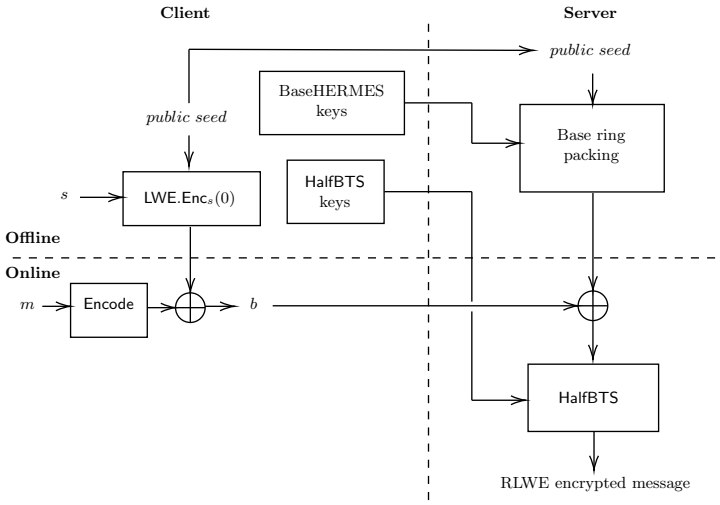

Fig. 5. Visualization of transciphering using HERMES.

Experimental results. In Table 11, we compare HERMES transciphering with HERA [CHK+21, Table 5, `Par-128a`] and Rubato [HKL+22, Table 6], which are state-of-the-art for transciphering to CKKS. All parameter sets in the table achieve security of ≈ 128 bits.

As in [CHK+21] and [HKL+22], we compute the expansion ratio as the ratio between the ciphertext bit-size and the plaintext precision multiplied by the number of slots. It is equal to $(\log q)/(p+1)$, where $\log q$ is the bit size of the ciphertext modulus and p is the precision. Note that this excludes the seed used for $\boldsymbol{a} = \mathrm{XOF}(\mathrm{seed})$, though its bit size is negligible compared to the ciphertext bit size when amortized over many ciphertexts (by using a counter, only one seed needs to be sent). We have $\log_2(Q_0) = 58$, $\Delta = 2^{42}$ and $\hat{p} = 22$, implying that $\log_2(q) = 58 - 42 + 22 = 38$.

Our method allows some flexibility to choose n. For instance, we can choose $n = 1$ to achieve the highest granularity, i.e., to send ciphertexts corresponding to individual plaintexts. Rubato also considered several granularity levels, but for optimizing client throughput. For a consistent comparison, we started from MLWE ciphertexts of degrees n close to Rubato's block sizes. We use HERMES[1] with midpoint ring degree of $2^{\lceil \log_2(\sqrt{N_B n}) \rceil}$ for each n, where N_B is a ring dimension for BaseHERMES.

Our latency is 5.5 times smaller than HERA's, and at least 2.8 times smaller than Rubato's, while achieving similar precision and similar number of levels remaining after transciphering. Our expansion ratio is larger than those achieved by HERA and Rubato, but might be improved by using different parameter sets and/or the technique from [BCC+22] to increase the precision.

Scheme	N	n	Latency (s)	Expansion ratio	Mean precision
RtF-HERA [CHK+21]	2^{16}	16	142	1.24	19.1
RtF-Rubato [HKL+22]		64	106	1.26	18.9
		36	88.4	1.26	18.9
		16	71.1	1.31	18.8
Ours		64	25.8	1.58	23.0
		32	26.1	1.58	23.0
		16	25.7	1.58	23.0
		1	30.9	1.58	23.0

Table 11. Comparison between different transciphering schemes. Here n denotes the block size which corresponds to the MLWE degree in our method, and latency denotes the total transciphering time including online and offline phases of the client and server. Note that there is a difference in precision: it is due to the difference in bootstrapping algorithms used.

Acknowledgment. The research corresponding to this work was conducted while the fourth author was visiting CryptoLab Inc. as an intern student.

References

[BCC+22] Y. Bae, J. H. Cheon, W. Cho, J. Kim, and T. Kim. META-BTS: Bootstrapping precision beyond the limit. In *CCS*, 2022.

[BGGJ20] C. Boura, N. Gama, M. Georgieva, and D. Jetchev. CHIMERA: combining ring-LWE-based fully homomorphic encryption schemes. *J. Math. Cryptol.*, 2020.

[BGV14] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 2014.

[Bra12] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapSVP. In *CRYPTO*, 2012.

[BTPH22] J.-P. Bossuat, J. Troncoso-Pastoriza, and J.-P. Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In *ACNS*, 2022.

[CDKS21] H. Chen, W. Dai, M. Kim, and Y. Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In *ACNS*, 2021.

[CGGI16] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT*, 2016.

[CGGI17] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *ASIACRYPT*, 2017.

[CHK+18] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT*, 2018.

[CHK+21] J. Cho, J. Ha, S. Kim, B. Lee, J. Lee, J. Lee, D. Moon, and H. Yoon. Transciphering framework for approximate homomorphic encryption. In *ASIACRYPT*, 2021.

[CKKS17] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, 2017.

[Cry] CryptoLab.inc. HEaaN private AI: Homomorphic encryption library.

[DM15] L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT*, 2015.

[FV12] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. 2012. Available at `http://eprint.iacr.org/2012/144`.

[GHPS13] C. Gentry, S. Halevi, C. Peikert, and N. P. Smart. Field switching in BGV-style homomorphic encryption. *Journal of Computer Security*, 2013.

[HK20] K. Han and D. Ki. Better bootstrapping for approximate homomorphic encryption. In *CT-RSA*, 2020.

[HKL+22] J. Ha, S. Kim, B. Lee, J. Lee, and M. Son. Rubato: Noisy ciphers for approximate homomorphic encryption. In *EUROCRYPT*, 2022.

[HS14] S. Halevi and V. Shoup. Algorithms in HElib. In *CRYPTO*, 2014.

[HS18] S. Halevi and V. Shoup. Faster homomorphic linear transformations in HElib. In *CRYPTO*, 2018.

[HS21] S. Halevi and V. Shoup. Bootstrapping for HElib. *J. Cryptol.*, 2021.

[KLK+22] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. Ahn. ARK: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse. In *MICRO*, 2022.

[LHH+21] W.-J. Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu. PEGASUS: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *S&P*, 2021.

[LPR10] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, 2010.

[LS15] A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 2015.

[MS18] D. Micciancio and J. Sorrell. Ring packing and amortized FHEW bootstrapping. In *ICALP*, 2018.

[NLV11] M. Naehrig, K. E. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW*, 2011.

[Reg09] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 2009.

[SSTX09] D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, 2009.

A List of Functions on Module Elements

Name	Symbol	Domain	Range	Introduced in
Module decomposition	$\pi^k_{q,N}$	$\mathcal{R}_{q,N}$	$(\mathcal{R}_{q,N/k})^k$	Sec. 3.2
Ring embedding	$\iota^k_{q,N}$	$\mathcal{R}_{q,N/k}$	$\mathcal{R}_{q,N}$	Sec. 3.2
Combine	$\mathsf{Combine}^k_{n,\ell}$	$(\mathcal{R}_{q,n})^{(k+1)\ell}$	$(\mathcal{R}_{q,n\ell})^{k+1}$	Sec. 3.2 & 4.2
Split	Split	$(\mathcal{R}_{q,N})^2$	$(\mathcal{R}_{q,N/\ell})^{2\ell}$	Sec. 3.2
Extract the first coefficient	$\epsilon^k_{q,N}$	$\mathcal{R}_{q,N}$	$\mathcal{R}_{q,N/k}$	Sec. 4.2
Twist	tw	$(\mathcal{R}_{q,N/k})^k$	$(\mathcal{R}_{q,N/k})^k$	Sec. 4.2
Embed	$\mathsf{Embed}^k_{q,N}$	$(\mathcal{R}_{q,N/k})^{k+1}$	$(\mathcal{R}_{q,N})^2$	Sec. 4.2
Extract	$\mathsf{Extract}^k_{q,N}$	$(\mathcal{R}_{q,N})^2$	$(\mathcal{R}_{q,N/k})^{k+1}$	Sec. 4.2
ModPack	$\mathsf{ModPack}^{K,k,k'}_N$	$(\mathcal{R}_{q,N/k})^{(K+1)k/k'}$	$(\mathcal{R}_{q,N/k'})^{k'+1}$	Sec. 4.3

Table 12. List of functions used to manipulate module elements.