Batch Bootstrapping II:

Bootstrapping in Polynomial Modulus Only Requires $\tilde{O}(1)$ FHE Multiplications in Amortization

Feng-Hao Liu[1], Han Wang (Corresponding Author)[2,3]

[1] Florida Atlantic University, Boca Raton, FL, USA. liuf@fau.edu.
[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Science, Beijing, China. {wanghan}@iie.ac.cn.
[3] School of Cyber Security, University of Chinese Academy of Science, Beijing, China.

Abstract. This work continues the exploration of the batch framework proposed in *Batch Bootstrapping I* (Liu and Wang, Eurocrypt 2023). By further designing novel batch homomorphic algorithms based on the batch framework, this work shows how to bootstrap λ LWE input ciphertexts within a polynomial modulus, using $\tilde{O}(\lambda)$ FHE multiplications. This implies an amortized complexity $\tilde{O}(1)$ FHE multiplications per input ciphertext, significantly improving our first work (whose amortized complexity is $\tilde{O}(\lambda^{0.75})$) and the theoretical state of the art MS18 (Micciancio and Sorrell, ICALP 2018), whose amortized complexity is $O(3^{1/\epsilon} \cdot \lambda^{\epsilon})$, for any arbitrary constant ϵ.
We believe that all our new homomorphic algorithms might be useful in general applications, and thus can be of independent interests.

1 Introduction

This work is the second work of the Batch Bootstrapping series, aiming to advance the frontier of the Fully homomorphic encryption (FHE). We continue the exploration of the algebraic batch bootstrapping framework of the first work [8], and our particular goal is to prove the following theorem:

Theorem 1.1 (Main Result of this Work, Informal) *Bootstrapping within a polynomial modulus requires $\tilde{O}(1)$ FHE multiplications in amortization.*

Contexts. FHE [6] is a powerful cryptographic tool that allows arbitrary computation over encrypted data, without the secret key. Currently, the only known way to achieve "fully"-HE is via the bootstrapping paradigm, which was originally perceived as *theoretical* only for its large computation overhead. After more than a decade of research and optimizations, there has been significant progress toward more efficient realizations. As major prior results (before the first work) have been summarized in the first work [8], curious readers can find relevant references there, so we do not repeat the presentation.

Below, we just go directly to the point, by staring with what is not achieved in [8], and a comparison with the current state of the art. In this way, the readers can easily identify the "delta", when reading the contributions of this work.

Challenges in the Prior Work. We first present a quick summary of the work [8], and state what was not solved. In Section 3, we give a more detailed review of the foundation, based on which, we develop various new homomorphic algorithms to further improve the frontier of the bootstrapping paradigm.

Briefly, the work [8] proposes a new batch framework, allowing single instruction multiple data (SIMD) operations that are compatible with FHEW-like (e.g., [4, 5]) bootstrapping methods. The framework allows SIMD computation over $r = O(\lambda^{0.25-o(1)})$ slots, where λ is the security parameter. Applying this to the AP14/FHEW/TFHE methods, we can bootstrap $r = O(\lambda^{0.25-o(1)})$ LWE ciphertexts within a polynomial modulus, using $\tilde{O}(\lambda)$ FHE multiplications, meaning $\tilde{O}(\lambda^{0.75})$ FHE multiplications in amortization. This is an improvement of a factor of $O(r)$ over the prior non-batch methods. We notice that all these methods only require workspace $O(1)$ FHE ciphertext for computation (excluding the input and the bootstrapping keys).

If more workspace for computation is available, the theoretical complexity of the above however, is not better than that of the existing method MS18 [12], whose amortized cost is $O(3^{1/\epsilon} \cdot \lambda^{\epsilon})$ FHE multiplications per input ciphertext, where $\epsilon > 0$ is an arbitrary constant. However, the dependency on ϵ posts an undesirable tradeoff between theory and practice – to achieve the best asymptotic complexity, ϵ should approach 0, e.g., 0.01, yet the constant would become prohibitively large, e.g., 3^{100}. Thus, it is not clear whether MS18 can lead to a practical method that matches their best theoretical indication.

Focus of this Work. An obvious open question is whether the tradeoff as stated above is inherent for the MS18 approach [12]. This work shows how to break the technical limitations, by developing various new batch homomorphic algorithms under the batch framework foundation [8]. Below we elaborate.

1.1 Our Contributions

The main result of this work is to prove Theorem 1.1. To achieve this, we first develop several new critical batch homomorphic algorithms based on the batch framework of [8]. These new algorithms play as important building blocks to improve the MS18 method, leading to our main result.

Recall that $r = O(\lambda^{0.25-o(1)})$ denotes the number of slots that the batch framework [8] can support. Using this foundation, we develop significant new batch homomorphic methods as stated below.

- We first propose a new batch vector-matrix multiplication algorithm, computing a vector of dimension w (in the clear) left multiplied by an encrypted matrix of dimension $h \times w$ for $h < r$, using $\tilde{O}(w + r)$ FHE multiplications. Thus, the amortized cost is $\tilde{O}(1 + w/r)$ FHE multiplications per dimension. As a ring multiplication can be expressed as the coefficient vector multiplied by the rotation matrix, our new batch algorithm immediately gives a batch algorithm for multiplying two ring elements of dimension $2d$ with amortized complexity $\tilde{O}(1 + 2d/r)$ FHE multiplications. Particularly, for $2d < r$, the amortized complexity would be $\tilde{O}(1)$ FHE multiplications per dimension. See Section 4 for details.

– Next we construct a new batch homomorphic (inverse) Discrete Fourier Transform (DFT) of dimension $2d < r$, with amortized complexity $\tilde{O}(1)$ FHE multiplications per dimension.
  To achieve this, we design three critical subroutines over packed ciphertexts: (1) homomorphic permutation, (2) homomorphic inverse over the exponents, and (3) batch homomorphic anti-cyclic rotation (via (1) + (2)). The batch homomorphic DFT/inverse-DFT can be achieved by using as a key building block the batch homomorphic anti-cyclic rotation. See Section 5 for details.
– We show that our batch homomorphic DFT/inverse-DFT is compatible with the recursive optimization of the Nussbaumer Transform. This plays a critical step to get rid of the dependency on ϵ as required by the MS18 framework. See Section 6 for details.

Putting these algorithms together, we are able to improve the overall MS18 bootstrapping method and achieve Theorem 1.1. See Section 7 for details of the final algorithm. We believe that all the new batch algorithms above can be of independent interests and might find applications in broader scoped of homomorphic computation. Below we present a table to compare results of this work with prior explicit methods (i.e., bootstrapping within a polynomial modulus).

Ref.	Amortized Complexity for Bootstrapping (# of FHE Multiplications per input LWE ciphertext)
[1, 4, 5]	$O(\lambda)$
[2]	$O(\lambda/\log\lambda)$
[12]	$O(3^{1/\epsilon} \cdot \lambda^{\epsilon})$
[8]	$\tilde{O}(\lambda^{0.75})$
This work	$\tilde{O}(1)$

Table 1. Comparison with prior work.

1.2 Technical Overview

We give a quick review of MS18 [12], and then present our new insights to break the technical limitation. We first recall the overall goal below.

The Goal. Let $\{\mathsf{ct}_i = (\boldsymbol{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q\}_{i\in[n]}$ be n LWE ciphertexts of dimension n, and each $b_i = \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e_i + q/2 \cdot m_i$, i.e., an encryption of the bit m_i. The goal is to compute bootstrapping of these n input ciphertexts (given appropriate bootstrapping keys), resulting in $\{\mathsf{ct}'_i = (\boldsymbol{a}'_i, b'_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q\}_{i\in[n]}$, where each output ct'_i encrypts the same underlying message m_i as ct_i.

The MS18 Framework. To achieve the goal, the MS18 framework does the following high-level steps:

1. First convert the input ciphertexts, i.e., $\{\mathsf{ct}_i\}_{i\in[n]}$, into a Ring-LWE ciphertext $(a, b) \in \mathcal{R}_q \times \mathcal{R}_q$ for ring $\mathcal{R}$ of degree n. Namely, $b = as + e + q/2 \cdot m$, where m is a ring element such that $\mathsf{coeffs}(m) = (m_1, \ldots, m_n)$, s is a ring element representing the secret key.

2. Let $z = b - as$. Given (a, b) in the clear and appropriate bootstrapping keys that encrypt the secret s, the next step computes n Ring-LWE ciphretexts $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n$, where each $\boldsymbol{d}_i \in \mathsf{RLWE.Enc}(X^{\mathsf{coeffs}(z)[i]})$. That is, the resulting ciphertexts encrypt the coefficients of z in the exponents.
3. Apply the sample-extraction procedure of [4, 5]. As a result, we have $\mathsf{ct}'_i \in \mathsf{LWE.Enc}(\mathsf{Round}(\mathsf{coeffs}(z)[i]))$, where Round is the Ring-LWE decryption rounding procedure.

It is easy to verify correctness of this approach. For complexity, the first and third steps are rather efficient as shown by [12]. The second step is the most computationally heavy one, and requires new techniques of homomorphic computation. The work MS18 [12] shows that this step can be achieved by $O(3^{1/\epsilon} \cdot \lambda^{1+\epsilon})$ FHE (particularly Ring-GSW) multiplications, and thus the amortized complexity is $O(3^{1/\epsilon} \cdot \lambda^{\epsilon})$ FHE multiplications per input LWE ciphertext (by setting $n = O(\lambda)$). This work shows how to further improve the efficiency of Step 2 by designing several new batch methods under the framework of [8].

In order to understand our insights, we need to delve into Step 2. Below we elaborate on this step, and some technical challenges that MS18 [12] faced. Then we present our new insight that breaks all these challenges.

More Details on Step 2. We notice that as long as we can homomorphically compute the coefficients of $w = -as$ in the exponents, i.e., $\tilde{\mathsf{ct}}_i \in \mathsf{RLWE.Enc}(X^{\mathsf{coeffs}(w)[i]})$, then this step can be achieved by additionally multiplying $X^{\mathsf{coeffs}(b)[i]}$ to $\tilde{\mathsf{ct}}_i$. Thus, we focus on how to homomorphically compute w given a in the clear and s encrypted under an appropriate form.

Naively, we can express $\mathsf{coeffs}(w) = \mathsf{Coeffs\text{-}Rot}(a) \cdot \mathsf{coeffs}(s)$ where $\mathsf{Coeffs\text{-}Rot}(a)$ is the anti-cyclic rotation matrix of a and $\mathsf{coeffs}(\cdot)$ denotes the coefficients of the input ring element. Then given bootstrapping keys $\mathsf{BK}_i = \mathsf{RGSW.Enc}(\mathsf{coeffs}(s)[i])$, we can achieve the task by using the FHEW-(like) method on every row of the rotation matrix $\mathsf{Coeffs\text{-}Rot}(a)$. However, this approach does not improve the amortized complexity at all, as it is basically the same as applying the straightforward method on individual input ciphertexts, separately.

To further improve the complexity, the work MS18 [12] explores nice recursive property from the algebraic ring in a novel way. Below we present some basic high level ideas, and the novel contributions of MS18.

We first recall that currently the most efficient way to compute ring multiplications is via the Fast Fourier Transform (FFT) technique, or its Number Theoretic Transform (NTT) variant as follow. To multiply ring elements a and s, we first convert a and s into the FFT/NTT form $(\tilde{a}_1, \ldots, \tilde{a}_n)$ and $(\tilde{s}_1, \ldots, \tilde{s}_n)$ respectively. Next we do a component-wise multiplication, and then convert the outcome back to the coefficient form using inverse FFT/NTT.

Following this idea, if we can adopt the idea to the homomorphic computation, then we can achieve the Step 2. However, there are several technical subtleties that a direct adoption would not work. Consider the following attempt: let $\mathsf{BK}_i = \mathsf{RGSW.Enc}(X^{\tilde{s}_i})$ be the bootstrapping key, encrypting the FFT/NTT coefficients in the exponents. Then we first homomorphically com-

pute $\mathsf{ct}'_i = \mathsf{RGSW.Enc}(X^{\tilde{s}_i \cdot \tilde{a}_i})$, which can be done via the method of [5]. Finally we compute the inverse-FFT/NTT for the final outcome. This idea seems promising, but would face the following technical barriers.

- The FFT representation needs to work with complex numbers, which is not compatible with the existing FHE schemes, especially for encrypting an element in the exponent.
- The NTT representation would require special property on the modulus q, i.e., $q\mathcal{R}$ fully splits. Such a modulus must be greater to n (and might be even much larger), and thus might not be compatible with FHEW, on which the MS18 framework is based.
- This subtle barrier is identified by the work MS18 [12] – the noise growth of the homomorphic computation based on FHEW would be $O(\lambda^\rho)$ where ρ is the recursive depth of the inverse FFT/NTT step. In order to bootstrap within a polynomial modulus, the recursive depth ρ can only be $O(1)$. As the complexity of (inverse)-FFT/NTT is better for larger recursive depth, this constraint seems to post an inherent barrier of efficiency of homomorphic (inverse)-FFT/NTT. In fact, this is also a major reason why MS18 has the dependency on ϵ.

To tackle the first two challenges as above, one novel technical insight of MS18 [12] is to (recursively) apply the Nussbaumer Transform over the FHEW framework [5], yet the third challenge still remains. This work shows that our new algorithms developed under the batch framework of [8] provide a novel way that solves the third challenge. We next elaborate on the idea of the Nussbaumer Transform, and then our new insights.

Nussbaumer Transform. We describe the high level concept using the algebraic language, which might look different from the description in MS18 [12] (and some other references), but what we state captures exactly the same algorithm. The algebraic presentation would be simpler for distilling its algorithmic ideas, assuming some algebraic number theory backgrounds.

Let $d > 2$ be a power of two, and $\mathbb{Z}[\xi_{2d}]$ be a subring of $\mathbb{Z}[\xi_{d^2}]$ where ξ_m is the m-th root of unity. To multiply $a, s \in \mathbb{Z}[\xi_{d^2}]$, the Nussbaumer Transform does essentially the following steps:

- Convert a, s into $2d$ points in the subring $\mathbb{Z}[\xi_{2d}]$, namely $(\tilde{a}_1, \ldots, \tilde{a}_{2d})$ and $(\tilde{s}_1, \ldots, \tilde{s}_{2d})$ via the Discrete Fourier Transform (DFT).
- Multiply the points in the subring coordinate-wisely, resulting in $(\tilde{z}_1, \ldots, \tilde{z}_{2d})$.
- Convert the result back to $z \in \mathbb{Z}[\xi_{d^2}]$ via the inverse DFT.

The DFT and inverse-DFT require the computation to support operations with the $2d$-th root of unity, i.e., ξ_{2d}, and its powers. Beautifully in the Nussbaumer Transform as above, we have $\xi_{2d} \in \mathbb{Z}[\xi_{2d}] \subset \mathbb{Z}[\xi_{d^2}]$, and thus, this required element and its powers naturally reside in the subring and the ring! This structure naturally supports the DFT/inverse-DFT, solving the first two challenges above.

This idea can be optimized in a recursive way. For example, consider the following tower of subrings: $\mathbb{Z}[\xi_{d^\rho}] \supset \mathbb{Z}[\xi_{d^{\rho-1}}] \supset \cdots \supset \mathbb{Z}[\xi_{d^2}] \supset \mathbb{Z}[\xi_{2d}]$ for $d > 2$ being some power of two. In order to compute ring multiplications over $\mathbb{Z}[\xi_{d^\rho}]$,

we can first recursively convert the elements into two vectors, each of $2^\rho \cdot d$ points in $\mathbb{Z}[\xi_{2d}]$. Then we compute the point-wise multiplication over $\mathbb{Z}[\xi_{2d}]$, and finally convert them back to an element in $\mathbb{Z}[\xi_{d^\rho}]$ by the inverse-DFT, recursively.

Now, let us describe the homomorphic version of the above idea, using as example the one-level recursion for simplicity of exposition, i.e., multiplying $a, s \in \mathbb{Z}[\xi_{d^2}]$ as above. The computation consists of the following three high level parts. (1) We can set the bootstrapping key as $\mathsf{BK}_{ij} = \mathsf{RGSW.Enc}(X^{\mathsf{coeffs}(\tilde{s}_i)[j]})$. (2) Then we homomorphically compute $\mathbf{C}_{ij} = \mathsf{RGSW.Enc}(X^{\mathsf{coeffs}(\tilde{z}_i)[j]})$, where $\tilde{z}_i = \tilde{a}_i \cdot \tilde{s}_i \in \mathbb{Z}[\xi_{2d}]$. (3) Finally we apply the homomorphic inverse DFT over these $\mathbf{C}_{ij}$'s as the MS18 method [12], resulting in what we want.

Limitations in MS18. To implement the above high level steps, MS18 however faces several technical challenges.

- First, to multiply elements in the bottom base field $\mathbb{Z}[\xi_{2d}]$, MS18 uses the textbook multiplication[4], whose amortized complexity is roughly $O(d)$ FHE multiplications per dimension.
- For the inverse DFT computation, MS18 also uses the straight-forward multiplication with the inverse-DFT matrix (of dimension $2d$), and similarly, the amortized complexity is roughly $O(d)$ multiplications per dimension.

Analyzing the recursion with the above facts, MS18 can compute multiplication over $\mathbb{Z}[\xi_{d^2}]$ roughly with amortized complexity $O(d)$ FHE multiplications per dimension. As MS18 observed, the recursive depth can be at most $\rho = O(1)$ to maintain a polynomial modulus, because the noise growth is roughly $O(\lambda^\rho)$. This would imply $d = O(\lambda^\epsilon)$, where $\epsilon = O(1/\rho) = O(1)$. Applying the argument recursively, their overall algorithm can achieve the amortized complexity $O(\lambda^\epsilon)$ FHE multiplications per input LWE ciphertext.

Our New Insights. Here we observe – as long as we can improve the amortized complexity of the ring multiplications over $\mathbb{Z}[\xi_{2d}]$ and inverse-DFT of dimension $2d$, we can improve the overall algorithm. Due to the noise growth, we cannot set $d = O(1)$ as it would require a large recursive depth, i.e., $\rho = O(\log \lambda)$. To handle this barrier, we next observe that the batch framework of our first work [8] is exactly the technical tool we need. Even though it can only batch $r = O(\lambda^{0.25-o(1)})$ slots, we can set $r > 2d$ such that the amortized complexity of the sub-ring multiplication over $\mathbb{Z}[\xi_{2d}]$ is small. Similarly, this idea can be applied to the inverse-DFT as well. Particularly, under the batch framework of [8], we develope the following new methods.

- We design a new homomorphic ring multiplication over $\mathbb{Z}[\xi_{2d}]$, using $\tilde{O}(d + d^2/r)$ FHE multiplications. Thus, the amortized complexity is $\tilde{O}(1)$ FHE multiplications per dimension.
- We design a new homomorphic inverse-DFT with dimension $2d$, with amortized complexity $\tilde{O}(1)$ FHE multiplications per dimension.

[4] As this is the bottom base field, no further recursive acceleration can be applied (e.g., Karatsuba or Toom-Cook).

Using a similar analysis of MS18 [12], we can then prove that the overall amortized complexity is $\tilde{O}(1)$ FHE multiplications, to bootstrap one input LWE ciphertext, achieving our main result. We notice that to implement the above high level picture requires substantial new design ideas over the batch framework [8]. We elaborate on the details of each piece in the coming sections.

2 Preliminaries

In this section, we present the preliminaries of this work. We note that this work shares a lot of common background with the first work of the series [8], so many basic materials are described verbatim as those in the first work.

Notations. Denote the set of integers by $\mathbb{Z}$, the set of rational numbers by $\mathbb{Q}$, real numbers by $\mathbb{R}$, and complex numbers by $\mathbb{C}$. Notation log refers to the base-2 logarithm. For a positive $k \in \mathbb{Z}$, let $[k]$ be the set of integers $\{1, ..., k\}$. We denote $[a, b]$ as the set $[a, b] \cap \mathbb{Z}$ for any integers $a \leq b$.

In this work, a vector is always a column vector by default and is denoted by a bold lower-case letter, e.g., $\boldsymbol{x}$. We use $\|\boldsymbol{x}\|_2$ denotes the l_2-norm and $\|\boldsymbol{x}\|_\infty$ denotes the l_∞-norm of $\boldsymbol{x}$. We use bold capital letters to denote matrices. For a matrix $\mathbf{X}$, $\mathbf{X}^\top$ denotes the transpose of $\mathbf{X}$. Given some set S, $S^{m\times n}$ denotes the set of all $m \times n$ matrices with entries in S. For matrices $\mathbf{X} \in S^{m\times n_1}$ and $\mathbf{Y} \in S^{m\times n_2}$ over some set S, $[\mathbf{X}\|\mathbf{Y}](\in S^{m\times(n_1+n_2)})$ denotes the concatenation of $\mathbf{X}$ with $\mathbf{Y}$. Let $\mathbf{X}$ be a matrix with even columns a matrix, and denote $\mathbf{X} = (\mathbf{X}^{(1)}\|\mathbf{X}^{(2)})$, where $\mathbf{X}^{(1)}$ is the left half sub-matrix of $\mathbf{X}$ and $\mathbf{X}^{(2)}$ is the right half sub-matrix.

For a set A and a probability distribution $\mathcal{P}$, we use $a \leftarrow A$ to denote that a is uniformly chosen from A and $a \leftarrow \mathcal{P}$ to denote that a is chosen according to the distribution $\mathcal{P}$.

Vector/Matrix Indexing. For vector $\boldsymbol{a}$, we use $\boldsymbol{a}[i]$ to describe the i-th element. Similarly, for matrix $\mathbf{X}$, we use $\mathbf{X}[i, j]$ to index the element in i-th row and j-th column. For an n-dimensional vector $\boldsymbol{a}$, we usually start the index from 1, i.e., $\boldsymbol{a} = (\boldsymbol{a}[1], \ldots, \boldsymbol{a}[n])$, and set $\boldsymbol{a}[0] = \boldsymbol{a}[n]$. Similarly, for an $n \times m$ matrix $\mathbf{X}$, we usually start the index from $\mathbf{X}[1, 1]$ to $\mathbf{X}[n, m]$, and set $\mathbf{X}[0, j] = \mathbf{X}[n, j]$ and $\mathbf{X}[i, 0] = \mathbf{X}[i, m]$ for general i, j's. For RGSW scheme, we use a bold and upper case variable, e.g., $\mathbf{C}$ to describe a ciphertext as it is a ring matrix. We use $\overrightarrow{\mathbf{C}}$ to describe a vector of ciphertexts, and $\overrightarrow{\mathbf{C}}[j]$ to index the j-th ciphertext. Similar to the previous case, $\overrightarrow{\mathbf{C}}[0] = \overrightarrow{\mathbf{C}}[n]$ indexes the n-th ciphertext where n is the vector dimension.

2.1 Lattices and sub-Gaussian Random Variables.

Lattices. An n-dimension (full-rank) lattice $\Lambda \subseteq \mathbb{R}^n$ is the set of all integer linear combinations of some set of independent basis vectors $\mathbf{B} = \{\mathsf{b}_1, \ldots, \mathsf{b}_n\} \subseteq \mathbb{R}^n$, $\Lambda = \mathcal{L}(\mathbf{B}) = \{\sum_{i=1}^n z_i \mathsf{b}_i : z_i \in \mathbb{Z}\}$.

Sub-Gaussian. As discussed in [1, 5], it is convenient to use the notion of sub-Gaussian to analyze the error growth in the FHE constructions. A sub-gaussian variable X with parameter $\alpha > 0$ satisfies $E[e^{2\pi tX}] \leq e^{\pi\alpha^2/t^2}$, for all $t \in \mathbb{R}$.

– Boundedness: If X is a sub-Gaussian variable with parameter $r > 0$, then $\Pr[|X| \geq t] \leq 2\exp(-\pi t^2/r^2)$.
– Homogeneity: If X is a sub-Gaussian variable with parameter $r > 0$, then cX is sub-gaussian with parameter $c \cdot r$ for any constant $c \geq 0$.
– Pythagorean additivity: If X_1 and X_2 are two sub-Gaussian variables with parameter r_1 and r_2 respectively, then $X_1 + X_2$ is sub-Gaussian with parameter $r_1 + r_2$, or $\sqrt{r_1^2 + r_2^2}$ if the two random variables are independent.

$\boldsymbol{g}^{-1}$ algorithm. This algorithm is used heavily in the research of FHE as we summarize in the following lemma.

Lemma 2.1 *For a given integer q, let $\ell = \lceil \log q \rceil$ and $\boldsymbol{g} = (1, 2, .., 2^{\ell-1})$. Then there is a randomized, efficiently computable algorithm denoted as $\boldsymbol{g}^{-1} : \mathbb{Z}_q \to \mathbb{Z}^\ell$ such that the output of the function, $\boldsymbol{x} \leftarrow \boldsymbol{g}^{-1}(a)$ is sub-gaussian with parameter $O(1)$, satisfying $\langle \boldsymbol{g}, \boldsymbol{x} \rangle = a \mod q$.*

We can extend $\boldsymbol{g}^{-1}$ to the matrix case (using the notation $\mathbf{G}^{-1}(\cdot)$) by applying $\boldsymbol{g}^{-1}(\cdot)$ to each entry of the matrix.

2.2 Algebraic Number Theory Background

We present some necessary background of algebraic number theory. This work heavily uses number fields and their rings of integers, and particularly, we represent a ring element as an algebraic number, instead of a polynomial. This representation gives more algebraic insights for our designs and analyses. Due to space limit, we defer some basic concepts in the full version of this work, and note that more details can be found in the work [10].

Number Fields. This work focuses on number fields as field extension that can be expressed as $K = \mathbb{Q}(\alpha)$, by adjoining some α to $\mathbb{Q}$ where α is a root of some irreducible polynomial $f(x) \in \mathbb{Z}[x]$. Let ξ_m be the m-th root of unity, and $\mathbb{Q}(\xi_m)$ is known as the m-th cyclotomic field. Suppose $\Phi_m(x)$ is the m-th cyclotomic polynomial, then the $\mathbb{Z}$-ring homomorphism Υ induces an isomorphism of $\mathbb{Z}[x]/\Phi_m(x) \cong \mathbb{Z}[\xi_m]$ as:

$$\Upsilon : \mathbb{Z}[x] \to \mathbb{Z}[\xi_m] \text{ such that } x \mapsto \xi_m.$$

We also use the concept of tensor fields, whose preliminaries are presented in the full version. Below we present a useful decomposition property.

Lemma 2.2 *[10] Let $m = \prod_\ell m_\ell$ be the prime-power factorization. Then $K = \mathbb{Q}(\xi_m)$ is isomorphic to the tensor product $\otimes_\ell \mathbb{Q}(\xi_{m_\ell})$, via the bijection $\prod_\ell a_\ell \mapsto \otimes_\ell(a_\ell)$, where each a_ℓ in K_ℓ can be naturally embedded in the field K.*

Geometry of Number Fields. Throughout this work, we use the canonical embedding to define norms for algebraic numbers. As argued in [10], this definition is independent of the representation of the algebraic number and can give us better bounds in the setting of general cyclotomic fields. Due to space limit, we defer the details to the full version of this work.

Trace, Ring of Integers, and Duality. The first work of this series [8] developed the batch homomorphic computation based heavily on the concepts of the algebraic trace, tensor rings, and their duals. This work builds upon the prior results in a black-box way, so our new results can still be accessible without the mathematical details.

2.3 Learning with Errors Assumption

Our schemes and analyses are based on the learning with errors (LWE) and the ring version RLWE (in general cyclotomic rings) as introduced by [9, 17]. We assume that the readers are familiar with these problems, and defer more details to the full version.

2.4 RLWE/RGSW in General Cyclotomic Rings

We present the schemes RLWE [9, 10] and RGSW [1, 7] in the setting of general cyclotomic rings. As the first work [8] showed, the noise behavior of the homomorphic operations in general cyclotomic rings is similar to that in the setting of power-of-two's, under the analysis of the canonical embedding [9, 10]. Below, we describe these schemes with a lemma that summarizes the noise growth.

Below we describe the parameters of the RLWE and RGSW schemes.

- λ: the security parameter.
- $\mathcal{R}$: the m-th cyclotomic ring with degree $N = \phi(m)$.
- Q: the modulus.
- $\mathcal{R}_Q$: the quotient ring $\mathcal{R}/Q\mathcal{R}$.
- $\mathcal{D}$: some error distribution over $\mathcal{R}$.
- ℓ: set $\ell = \lceil \log Q \rceil$ (with respect to some log base).

RLWE Scheme. We describe the basic symmetric RLWE encryption scheme (in the primal form for simplicity). The scheme contains the following algorithms.

- $\mathbf{KeyGen}(1^\lambda)$: Choose randomly $s \leftarrow \mathcal{R}_Q$ and output $\mathsf{sk} := (1, -s)^\top \in \mathcal{R}_Q^2$.
- $\mathbf{Enc}(\mathsf{sk}, \mu \in \mathcal{R}_t)$: Sample a uniform ring element $a \leftarrow \mathcal{R}_Q$ and a noise $e \leftarrow \mathcal{D}$. The output ciphertext is set as $\boldsymbol{c} := \begin{pmatrix} sa + e \\ a \end{pmatrix} + \begin{pmatrix} \lfloor \frac{Q}{t} \rceil \mu \\ 0 \end{pmatrix} \in \mathcal{R}_Q^2$.
  We call $\left\lfloor \frac{Q}{t} \right\rceil \mu$ the encoded message of $\boldsymbol{c}$ and μ the encrypted message of $\boldsymbol{c}$.
- $\mathbf{Dec}(\boldsymbol{c}, \mathsf{sk})$: The algorithm outputs an element μ in $\mathcal{R}_t$ as follow:

$$\mu = \lfloor \langle (1, -s), \boldsymbol{c} \rangle \rceil_t := \lfloor t \langle (1, -s), \boldsymbol{c} \rangle / Q \rceil \mod t.$$

We use $\mathsf{RLWE}_s^{t/Q}(\mu)$ to denote the set of all RLWE ciphertexts of encoded message μ under secret s with ciphertext modulus Q and plaintext modulus t. Sometimes, we use $\mathsf{RLWE}_s^{Q}(\left\lfloor \frac{Q}{t} \right\rceil \mu)$ to denote the same set. The latter notion drops the t in the super-script, but presents the whole encoded message in the parentheses.

RGSW Scheme. Now we present the RGSW scheme. We notice that this work suffices to use the symmetric-key version of RGSW, so we just present this for simplicity. The public-key version works analogously.

Denote the fixed gadget vector as $\boldsymbol{g}^\top = (1, 2, ..., 2^{\ell-1})$, and the gadget matrix as $\mathbf{G} = \boldsymbol{g}^\top \otimes \mathbf{I}_2$. As demonstrated by [1,11], the gadget vector/matrix play a vital role in the homomorphic computation methods. Similar to the RLWE scheme above, we present the primal version of RGSW.

- $\mathbf{KeyGen}(1^\lambda)$: Choose randomly $s \leftarrow \mathcal{R}_Q$ and set $\mathsf{sk} := (1, -s)^\top \in \mathcal{R}_Q^2$.
- $\mathbf{Enc}(\mathsf{sk}, \mu \in \mathcal{R}_2)$: Sample a uniform vector $\boldsymbol{a} \leftarrow \mathcal{R}_Q^{2\ell}$ and a noise vector $\boldsymbol{e} \leftarrow \mathcal{D}^{2\ell}$. The ciphertext is set as $\mathbf{C} := \begin{pmatrix} s\boldsymbol{a}^\top + \boldsymbol{e}^\top \\ \boldsymbol{a}^\top \end{pmatrix} + \mu\mathbf{G} \in \mathcal{R}_Q^{2\times 2\ell}$.
- $\mathbf{Dec}(\mathbf{C}, \mathsf{sk})$: The algorithm outputs an element μ in $\mathcal{R}_t$ as follow:

$$\mu = \left\lfloor \langle (1, -s)^\top, \boldsymbol{c}_{(\ell-1)} \rangle \right\rceil \mod 2,$$

  where $\boldsymbol{c}_{(\ell-1)}$ is the $(\ell-1)$-th column of $\mathbf{C}$.
- **Homomorphic Addition** $\mathbf{C}_1 \boxplus \mathbf{C}_2$: It takes as inputs two RGSW ciphertexts $\mathbf{C}_1$, $\mathbf{C}_2$ under the same secret key sk and outputs $\mathbf{C}_1 \boxplus \mathbf{C}_2 := \mathbf{C}_1 + \mathbf{C}_2$.
- **Homomorphic Multiplication** $\mathbf{C}_1 \boxdot \mathbf{C}_2$: It takes as inputs two RGSW ciphertexts $\mathbf{C}_1$, $\mathbf{C}_2$ under the same secret key sk and outputs the following as the result of homomorphic multiplication: $\mathbf{C}_1 \boxdot \mathbf{C}_2 \leftarrow \mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$. Here $\mathbf{G}^{-1}(\cdot)$ can be either deterministic or randomized. As argued by [1], a randomized instantiation can yield tighter parameters of the noise growth than those derived from the deterministic version. We notice that in the ring setting, a basis needs to be specified when computing $\mathbf{G}^{-1}$.
- **External Product** $\mathbf{C}_1 \boxtimes \boldsymbol{c}_2$: It takes as inputs a RGSW ciphertexts $\mathbf{C}_1$ and a RLWE ciphertext $\boldsymbol{c}_2$ under the same secret key sk and outputs the following RLWE ciphertext as the result of external product: $\mathbf{C}_1 \boxtimes \boldsymbol{c}_2 \leftarrow \mathbf{C}_1 \cdot \boldsymbol{g}^{-1}(\boldsymbol{c}_2)$.

The IND-CPA security of the above RGSW scheme (for general cyclotomic rings) follows from the RLWE assumption, using the same argument of [1,7]. Below we present some notations for the noise analysis.

Definition 2.3 *Adapt the notations from the above. Given a ciphertext* $\mathbf{C}$ *that encrypts message* μ *under a secret key* $\mathsf{sk} = (1, -s)^\top$*, we can express as the following relation* $\mathsf{sk}^\top \cdot \mathbf{C} = \mu \cdot \mathsf{sk}^\top \cdot \mathbf{G} + \boldsymbol{e}^\top \in \mathcal{R}_Q^m$*, for some error vector* $\boldsymbol{e}$*. Then define* $\mathsf{Err}_\mu(\mathbf{C}) := \boldsymbol{e}^\top = \mathsf{sk}^\top \cdot \mathbf{C} - \mu \cdot \mathsf{sk}^\top \cdot \mathbf{G}$*. When the context is clear, we may drop the index* μ*.*

We use $\mathsf{RGSW}_s^Q(\mu)$ *to denote the set of all the* RGSW *ciphertexts that encrypt* μ *under secret* s *in the modulo* Q *space. If the parameters* Q *are clear from the context, we would use the abbreviation* $\mathsf{RGSW}_s(\mu)$ *for simplicity.*

Note. The above error function can be defined for RLWE ciphertexts analogously. We do not present another definition to avoid repetition.

The following analysis was developed by the prior work of the series [8].

Lemma 2.4 ([8]) *For any* RGSW *ciphertexts* $\mathbf{C}_1, \mathbf{C}_2$ *that encrypt* μ_1, μ_2 *with the error terms* $\boldsymbol{e}_1, \boldsymbol{e}_2$ *respectively, then we have the following.*

- $\mathsf{Err}(\mathbf{C}_1 \boxplus \mathbf{C}_2) = \boldsymbol{e}_1^\top + \boldsymbol{e}_2^\top$.
- $\mathsf{Err}(\mathbf{C}_1 \boxdot \mathbf{C}_2) = \boldsymbol{e}_1^\top \cdot \mathbf{G}^{-1}(\mathbf{C}_2) + \mu_1 \cdot \boldsymbol{e}_2^\top$.

Furthermore, suppose $\mathbf{G}^{-1}$ *is sampled with respect to some* $\mathbb{Z}$*-basis of* $\mathcal{R}$*, i.e.,* $\mathbf{B} = \{\mathsf{b}_1, ..., \mathsf{b}_n\}$*, such that for all* $i \in [n]$ $\|\sigma(\mathsf{b}_i)\|_\infty \leq 1$*. Then the following holds.*

- *Denote* $\boldsymbol{e}_1^\top \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$ *as* $\boldsymbol{e}^\top = (e_1, ..., e_{2\ell})$*. Then each entry of* $\boldsymbol{e}$ *is an independent random variable.*
- $||\sigma(\boldsymbol{e})||_\infty$ *is upper bounded by a sub-Gaussian variable with parameter* $O(r)$*, for some real positive* $r \leq \sqrt{N \cdot \log Q} \cdot \|\sigma(\boldsymbol{e}_1)\|_\infty$.

Encrypted Elements in the Exponents. Next we define a notation for RGSW ciphertexts, encrypting integers of a vector in the exponents. This notation will be convenient for the presentation of our new homomorphic algorithms.

Definition 2.5 *Let* ξ_p *be the* p*-th root of unity which is included in the message space of* RGSW*. Given an integer vector* $\boldsymbol{a} = (a_0, a_1, \cdots, a_{n-1}) \in \mathbb{Z}^n$*, we denote* $\mathsf{RGSW.EncVec\text{-}Exp}(\boldsymbol{a})$ *as a vector of ciphertexts, each entry of which is a* RGSW *ciphertext encrypting* $\xi_q^{a_i}$*. Namely,* $\overrightarrow{\mathbf{C}} = (\mathbf{C}_0, \cdots, \mathbf{C}_{n-1}) \in \mathsf{RGSW.EncVec\text{-}Exp}(\boldsymbol{a})$*, where each* $\mathbf{C}_i \in \mathsf{RGSW}(\xi_q^{a_i})$.

The parameter ξ_q will be specified in each algorithm that uses $\mathsf{RGSW.EncVec\text{-}Exp}$. Moreover, there exists a homomorphic anti-rotation algorithm $\mathsf{Anti\text{-}Rot}(\cdot, \cdot)$ that on input $\overrightarrow{\mathbf{C}} \in \mathsf{RGSW.EncVec\text{-}Exp}(\boldsymbol{a})$ and $z \in \mathbb{Z}$ outputs a rotated ciphertext $\overrightarrow{\mathbf{C}}' \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{Anti\text{-}Rot}(\boldsymbol{a}, z))$, where $\mathsf{Anti\text{-}Rot}(\boldsymbol{a}, z)$ is the anti-cyclic rotation of z positions in the plaintext. The error growth is only increased by an additive term $\boldsymbol{e}'$ that is independent of the input ciphertext.

3 Foundation Developed in Batch Bootstrapping I

In this section, we present the framework of batch homomorphic computation of the work [8]. To be rigorous, our presentation uses the math concepts of tensor rings and dual basis. To make it more friendly to the general, we abstract the required homomorphic methods and analyses in a modular way, so that how to apply the framework can be accessible without going into the math details. The main results and algorithms of this work will be presented using the modular abstraction of the homomorphic methods.

Math Background and Notations. Let $K = K_1 \otimes K_2 \otimes K_3$ be a tensor field of three linearly disjoint fields, and $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$ be their rings of integers, respectively. It follows that the ring of integers of K (denoted as $\mathcal{R}$) is isomorphic to $\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$. Furthermore, we present some useful facts and notations.

- K_{12} and K_{13} denote $K_1 \otimes K_2$ and $K_1 \otimes K_3$, respectively.

- $\mathcal{R}, \mathcal{R}_{12}$ and $\mathcal{R}_{13}$ denote the rings of integers of K, K_{12}, and K_{13}, respectively. It is known that $\mathcal{R} \cong \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$, $\mathcal{R}_{12} \cong \mathcal{R}_1 \otimes \mathcal{R}_2$, and $\mathcal{R}_{13} \cong \mathcal{R}_1 \otimes \mathcal{R}_3$.
- Let $(\mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_\rho)$ and $(\mathsf{w}_1, \mathsf{w}_2, \ldots, \mathsf{w}_\tau)$ be some $\mathbb{Z}$-bases of $\mathcal{R}_2$ and $\mathcal{R}_3$, respectively, where ρ and τ are the degrees of the rings $\mathcal{R}_2$ and $\mathcal{R}_3$.
- Denote $(\mathsf{v}_1^\vee, \mathsf{v}_2^\vee, \ldots, \mathsf{v}_\rho^\vee)$ and $(\mathsf{w}_1^\vee, \mathsf{w}_2^\vee, \ldots, \mathsf{w}_\tau^\vee)$ as the corresponding $\mathbb{Z}$-bases of the dual spaces $\mathcal{R}_2^\vee$ and $\mathcal{R}_3^\vee$, respectively.
- Let $r = \min(\rho, \tau)$, the maximal number of slots our method can pack.
- Denote the trace functions (with respect to different underlying subfields) as

$$\mathsf{Tr}_{K/K_{12}} : K \to K_{12} \text{ and } \mathsf{Tr}_{K/K_{13}} : K \to K_{13}$$

In our instantiation, we set $K := \mathbb{Q}[\xi_{q\rho'\tau'}] \cong \mathbb{Q}[\xi_q] \otimes \mathbb{Q}[\xi_{\rho'}] \otimes \mathbb{Q}[\xi_{\tau'}] := K_1 \otimes K_2 \otimes K_3$, where q is equal to the modulus of input (Ring)-LWE being bootstrapped, ρ' and τ' are powers of some prime numbers of size $O(1)$. Moreover, we have $\rho = \phi(\rho')$ and $\tau = \phi(\tau')$.

3.1 The Framework of Batch Homomorphic Computation

By using the tensor of three rings, the work [8] showed how to batch homomorphic computation as we summarize below.

Message Packing and Operations. First, the message space is the first ring, i.e., $\mathcal{R}_1$, and the other two rings, i.e., $\mathcal{R}_2, \mathcal{R}_3$ are the work rings for computation. Particularly, there are features as following:

1. There are four modes of packing, i.e., $\mathsf{mode} \in \{$"$\mathcal{R}''_{12}$, "$\mathcal{R}''_{13}$, "$\mathcal{R}_{12} \to \mathcal{R}''_{13}$, "$\mathcal{R}_{13} \to \mathcal{R}''_{12}\}$, where a vector of messages can be packed with respect to.
2. There is an algorithm Pack that on input a vector (of messages) $\boldsymbol{x} = (x_1, \ldots, x_r) \in \mathcal{R}_1^r$ and mode outputs a packed message x. Particularly, if $\mathsf{mode} =$ "$\mathcal{R}''_{12}$, then $x \in \mathcal{R}_1 \otimes \mathcal{R}_2$; $\mathsf{mode} =$ "$\mathcal{R}''_{13}$, $x \in \mathcal{R}_1 \otimes \mathcal{R}_3$; $\mathsf{mode} =$ "$\mathcal{R}_{12} \to \mathcal{R}''_{13}$, $x \in \mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3$; $\mathsf{mode} =$ "$\mathcal{R}_{13} \to \mathcal{R}''_{12}$, $x \in \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3^\vee$.
3. For any two packed messages (x, mode) and (y, mode) where $x = \mathsf{Pack}(\boldsymbol{x}, \mathsf{mode})$ and $y = \mathsf{Pack}(\boldsymbol{y}, \mathsf{mode})$, $(x + y, \mathsf{mode}) = \mathsf{Pack}((\boldsymbol{x} + \boldsymbol{y}), \mathsf{mode})$.
4. There is a multiplication method that on input two packed messages (x, mode_1) and (y, mode_2), outputs a packed message (z, mode_3) with the following. If $\mathsf{mode}_1 =$ "$\mathcal{R}''_{12}$, $\mathsf{mode}_2 =$ "$\mathcal{R}_{12} \to \mathcal{R}''_{13}$ or vice versa, then $\mathsf{mode}_3 =$ "$\mathcal{R}''_{13}$. If $\mathsf{mode}_1 =$ "$\mathcal{R}''_{13}$, $\mathsf{mode}_2 =$ "$\mathcal{R}_{13} \to \mathcal{R}''_{12}$ or vice versa, then $\mathsf{mode}_3 =$ "$\mathcal{R}''_{12}$. For all the other cases, $\mathsf{mode}_3 = \perp$.
   Moreover, $(z, \mathsf{mode}_3) = \mathsf{Pack}(\boldsymbol{z}, \mathsf{mode}_3)$ where $\boldsymbol{z} = (x_1 y_1, \ldots, x_r y_r)$.

Ciphertext Packing and Operations. For RGSW instantiated over the tensor ring $\mathcal{R}$, the work [8] realizes homomorphic methods for the above plaintext packing and operations. Particularly, we have the following.

1. There is an algorithm $\mathsf{RGSW\text{-}Pack}$ that on input mode and $\mathbf{C}_1, \ldots, \mathbf{C}_r$ where each $\mathbf{C}_i \in \mathsf{RGSW}(x_i) \in \mathcal{R}^{2\times 2\ell}$ and $x_i \in \mathcal{R}_1$, outputs a packed ciphertext $(\mathbf{C}, \mathsf{mode})$. The ciphertext $\mathbf{C} \in \mathcal{R}^{2\times 2\ell}$ or the dual ring (omitting the modulus), depending on mode the same way as Item 2 of the plaintext packing. Importantly, the size of $\mathbf{C}$ is the same as that of each $\mathbf{C}_i$, meaning that the packing method is non-trivial.

2. Let $(\mathbf{C}_x, \mathsf{mode})$ and $(\mathbf{C}_y, \mathsf{mode})$ be two packed ciphertexts of the message vectors $\boldsymbol{x} = (x_1, \ldots, x_r) \in \mathcal{R}_1^r$, $\boldsymbol{y} = (y_1, \ldots, y_r) \in \mathcal{R}_1^r$. Then $(\mathbf{C}_x + \mathbf{C}_y, \mathsf{mode})$ is a packed ciphertext of the message vector $\boldsymbol{x} + \boldsymbol{y}$.
3. Continue from the above notation. There is a non-trivial[5] batch homomorphic algorithm $\mathsf{Batch\text{-}Mult}$ that on input $(\mathbf{C}_x, \mathsf{mode}_1)$ and $(\mathbf{C}_y, \mathsf{mode}_2)$ outputs $(\mathbf{C}_z, \mathsf{mode}_3)$ where the modes $\mathsf{mode}_1, \mathsf{mode}_2, \mathsf{mode}_3$ follow the relation as described in item 4 of the above plaintext packing. Moreover, $\mathbf{C}_z$ is a packed ciphertext that corresponds to the vector of messages $(x_1 y_1, \ldots, x_r y_r)$.
4. There is an algorithm UnPack that on input a packed ciphertext $(\mathbf{C}_x, \mathsf{mode})$ outputs $\mathbf{C}_1, \ldots, \mathbf{C}_r$ where each $\mathbf{C}_i \in \mathsf{RGSW}(x_i)$ for $x_i \in \mathcal{R}_1$.

Remark 3.1 *In the framework, only two ciphertexts/plaintexts of the modes ("$\mathcal{R}''_{12}$ and "$\mathcal{R}_{12} \to \mathcal{R}''_{13}$) or ("$\mathcal{R}_{13}$ and "$\mathcal{R}_{13} \to \mathcal{R}''_{12}$) can be homomorphically multiplied. All the other combinations do not support the multiplication, e.g., if $\mathbf{C}_1$ is mode "$\mathcal{R}''_{12}$ and $\mathbf{C}_2$ is mode "$\mathcal{R}''_{12}$, then they cannot be multiplied.*

Parameters and Computational Efficiency. The first work [8] showed that the following parameters and computational complexity are feasible. First, we set the first ring as the q-th cyclotomic ring, i.e., $\mathcal{R}_1 = \mathbb{Z}(\xi_q)$, where q the modulus of the input LWE being bootstrapped. Then we set the maximal number of slots as $r = \deg(\mathcal{R}_2/\mathbb{Q}) \approx \deg(\mathcal{R}_3/\mathbb{Q}) = O(\sqrt{N/q})$. Asymptotically, this would be $q = \tilde{O}(\sqrt{\lambda})$, meaning that $r = O(\lambda^{0.25-o(1)})$.

For the (homomorphic) efficiency, the following can be achieved.

- $\mathsf{RGSW\text{-}Pack}$ requires $O(r)$ RGSW additions.
- Efficiency of the packed addition is the same as that of the RGSW addition.
- $\mathsf{Batch\text{-}Mult}$ for packed ciphertexts takes $O(\log \lambda)$ number of calls to RGSW multiplications, i.e., $\boxdot$, by setting $\mathcal{R}_2$ and $\mathcal{R}_3$ as cyclotomic rings with a proper tower structure.
  Note: this satisfies the non-trivial requirement as $O(\log \lambda)$ calls of RGSW multiplications are much less than the trivial non-batch method, which requires $r = O(\lambda^{0.25-o(1)})$ RGSW multiplications.
- UnPack takes $O(r)$ RGSW multiplications.

Noise Growth. The noise growth depends on how we choose the basis for $\mathbf{G}^{-1}$ to which the RGSW multiplication is with respect (see Lemma 2.4 for reference). For the case of general cyclotomic rings, the work [8] showed a way to instantiate a short basis (with infinity norms 1 for all elements in the basis) and all the necessary components, leading to the following results:

Theorem 3.2 ([8]) *Let $\mathbf{C}$ be a RGSW ciphertext that encrypts $m \in \mathcal{R}$, and denote $\mathsf{Err}(\mathbf{C}) := (\mathsf{Err}_1(\mathbf{C}) \| \mathsf{Err}_2(\mathbf{C}))$, where $\mathsf{Err}_1(\mathbf{C})$ is the first half of the error vector, and $\mathsf{Err}_2(\mathbf{C})$ is the other half. There exists a homomorphic method $\mathsf{Eval\text{-}Tr}_{K/K_{12}}$ that on input $\mathbf{C}$ outputs $\mathbf{C}' \in \mathsf{RGSW}(\mathsf{Tr}_{K/K_{13}}(m))$, satisfying $\mathsf{Err}_1(\mathbf{C}') = \mathsf{Tr}_{K/K_{12}}(\mathsf{Err}_1(\mathbf{C})) + \boldsymbol{e}'$ for some $\boldsymbol{e}'$ that is independent of $\mathbf{C}$. Moreover, $\mathsf{Err}_2(\mathbf{C}') = s \cdot \mathsf{Tr}_{K/K_{12}}(\mathsf{Err}_1(\mathbf{C})) + \boldsymbol{e}''$ for some $\boldsymbol{e}''$ that is independent of $\mathbf{C}$.*

[5] The term *non-trivial* requires $\mathsf{Batch\text{-}Mult}$ to be much more efficient than the trivial non-batch computation, i.e., computing r RGSW multiplications separately and then packing the outcomes into one ciphertext.

Theorem 3.3 ([8]) *Let* $\mathbf{C}_1, \ldots, \mathbf{C}_r$ *be* RGSW *ciphertexts with error terms* $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_r$, *messages* $\mu_1, \ldots, \mu_r \in \mathcal{R}_1$ *and* $\mathbf{C}'_1, \ldots, \mathbf{C}'_r$ *be* RGSW *ciphertexts with error terms* $\boldsymbol{e}'_1, \ldots, \boldsymbol{e}'_r$, *messages* $\mu'_1, \ldots, \mu'_r \in \mathcal{R}_1$. *Denote*

- $\mathsf{RGSW\text{-}Pack}(\mathbf{C}_1, \ldots, \mathbf{C}_r, \text{“}\mathcal{R}_{12}\text{”})$ *as* $\mathbf{D}$,
- $\mathsf{RGSW\text{-}Pack}(\mathbf{C}'_1, \ldots, \mathbf{C}'_r, \text{“}\mathcal{R}_{12} \to \mathcal{R}_{13}\text{”})$ *as* $\mathbf{D}'$,
- $\mathsf{Batch\text{-}Mult}(\mathbf{D}', \mathbf{D})$ *as* $\mathbf{F}$,
- *the encrypted messages of the packed ciphertexts* $\mathbf{D}$ *as* $\mu_{\mathbf{D}}$,
- *the encrypted messages of the packed ciphertexts* $\mathbf{D}'$ *as* $\mu_{\mathbf{D}'}$.

Then, $\mu_{\mathbf{D}} = \sum_{i=1}^{r} \mu_i \cdot \mathsf{v}_i$, $\mu_{\mathbf{D}'} = \sum_{i=1}^{r} \mu'_i \cdot \mathsf{v}_i^{\vee}\mathsf{w}_i$ *and* $\mathbf{F}$ *is a packed* RGSW *ciphertext encrypting* $\mathsf{Tr}_{K/K_{13}}(\mu_{\mathbf{D}} \cdot \mu_{\mathbf{D}'})$ *with mode* $\mathcal{R}_{13}$.

Combing Algorithm 3.2, then we have $\mathsf{Err}_1(\mathbf{F}) = \mathsf{Tr}_{K/K_{13}}(\sum_i \boldsymbol{e}'_i \mathsf{v}_i^{\vee}\mathsf{w}_i \mathbf{G}^{-1}(\mathbf{D}^{(1)}) + \mu_{\mathbf{D}'}(\boldsymbol{e}_i^{(1)}\mathsf{v}_i)) + \boldsymbol{e}'$ *and* $\mathsf{Err}_2(\mathbf{F}) = s \cdot \mathsf{Tr}_{K/K_{13}}(\sum_i \boldsymbol{e}'_i \mathsf{v}_i^{\vee}\mathsf{w}_i \mathbf{G}^{-1}(\mathbf{D}^{(1)}) + \mu_{\mathbf{D}'}(\boldsymbol{e}_i^{(1)}\mathsf{v}_i)) + \boldsymbol{e}''$, *where* $\boldsymbol{e}'$ *and* $\boldsymbol{e}''$ *are independent of* $\mathbf{C}$.

Corollary 3.4 ([8]) *Adapt the notations of Theorems 3.3. If the errors of the key-switch keys is upper bounded by* E, *and* $\boldsymbol{g}^{-1}$ *is with respect to the basis* $\mathcal{B}_1 \otimes \mathcal{B}_2 \otimes \mathcal{B}_3$. *Denote the error of the output by* $\mathsf{Err}(\mathbf{F}) = (\boldsymbol{e}_1^{\top} || \boldsymbol{e}_2^{\top})$, *where* $\boldsymbol{e}_1$ *and* $\boldsymbol{e}_2$ *are both* ℓ*-entry vectors. Then* $\|\mathsf{Err}_1(\mathbf{F})\|_{\infty}$ *is upper bounded by*

$$\frac{2\rho(p_1-1)\sqrt{N\log Q}}{\rho'} \sum \|\boldsymbol{e}'_i\|_{\infty} + \rho\|\mu_{\mathbf{D}'}\| \sum \|\boldsymbol{e}_i\|_{\infty} + \|e'\|_{\infty},$$

where $\|e'\|_{\infty}$ *is a sub-Gaussian with parameter upper bounded by* $3\rho'\sqrt{N\log Q}E$.

$\|\mathsf{Err}_2(\mathbf{F})\|_{\infty}$ *is upper bounded by*

$$\frac{2\rho(p_1-1)\sqrt{N\log Q}\|s\|_{\infty}}{\rho'} \sum \|\boldsymbol{e}'_i\|_{\infty} + \rho\|s\|_{\infty}\|\mu_{\mathbf{D}'}\| \sum \|\boldsymbol{e}_i\|_{\infty} + \|e''\|_{\infty},$$

where $\|e''\|_{\infty}$ *is a sub-Gaussian with parameter upper bounded by* $(3\rho'\|s\|_{\infty} + 2)\sqrt{N\log Q}E$.

4 New Batch Homomorphic Algorithms

Here we present some critical batch homomorphic algorithms, which will be used as building blocks to improve the MS18 method [12]. As discussed in the introduction, an important goal is to design a batch algorithm that gives a better amortized efficiency to compute ring multiplications of the sub-ring $\mathbb{Z}[\xi_{2d}]$.

To achieve this, we first consider a more general setting of batch vector-matrix multiplication of the following form. The input contains:

1. v vectors $\boldsymbol{a}_1 \ldots \boldsymbol{a}_v$ where for $k \in [v]$, $\boldsymbol{a}_k \in \{0,1\}^w$;
2. v matrices of ciphertexts $\{\mathbf{C}_{k,(i,j)}\}_{i\in[h],j\in[w],k\in[v]}$, where each $\mathbf{C}_{k,(i,j)} \in \mathsf{RGSW}(\xi_q^{\mathbf{M}_k[i,j]})$ and for each $k \in [v]$, $\mathbf{M}_k$ is a matrix in the domain $\mathbb{Z}_q^{h\times w}$.

Let $\boldsymbol{z}_k = \mathbf{M}_k \cdot \boldsymbol{a}_k \in \mathbb{Z}_q^h$ for $k \in [v]$. The goal is to compute a vector of ciphertext $\overrightarrow{\mathbf{C}} \in \mathsf{RGSW.EncVec\text{-}Exp}(\boldsymbol{z}_1 \| \boldsymbol{z}_2 \| \ldots \| \boldsymbol{z}_v)$, where $\|$ denotes the concatenation.

Even though each input vector $\boldsymbol{a}_i \in \{0,1\}^w$ is just a bit vector, this still suffices to capture general vector-matrix multiplication in $\mathbb{Z}_q$ by using the technique of bit-decomposition and power-of-2. Particularly, any $\mathbf{X} \cdot \boldsymbol{y}$ is equivalent to $\mathbf{X}' \cdot \boldsymbol{y}'$ where $\mathbf{X}'$ is the power-of-2 matrix, i.e., $= \boldsymbol{g}^\top \otimes \mathbf{X}$, and $\boldsymbol{y}' = \mathbf{G}^{-1}(\boldsymbol{y})$ is the bit-decomposition vector. Therefore, this form of homomorphic computation would be sufficient for our later algorithms.

For the naive un-batch homomorphic computation, this would require $v \cdot h \cdot w$ RGSW multiplications. In the next section, we show that suppose the input ciphertext is packed under the batch framework [8], then we can improve the efficiency by using roughly $O(v \cdot h \cdot w/r)$ RGSW multiplications. By using this batch algorithm, we can derive more efficient homomorphic ring multiplications of the sub-ring $\mathbb{Z}[\xi_{2d}]$ and other critical procedures as we will present next.

Note: our further presentation heavily uses indices to vectors and matrices, and thus we would recommend the readers to quickly recall the indexing rules of this work as described in the preliminary (Section 2). Consider an example with an n-dimensional vector $\boldsymbol{a}$. We represent it as $(\boldsymbol{a}[1], \boldsymbol{a}[2], \ldots, \boldsymbol{a}[n])$, and for convenience we use $\boldsymbol{a}[0]$ as a reference to $\boldsymbol{a}[n]$ – namely, they are the same variable holding the same value.

4.1 Batch "Vector"-"Encrypted Matrix" Multiplication

Let $\{\mathbf{M}_k\}_{k\in[v]}$ be matrices, each belonging to $\mathbb{Z}_q^{h\times w}$, and $\{\mathbf{C}_{k,(i,j)}\}_{i\in[h],j\in[w],k\in[v]}$ be RGSW ciphertexts as specified as above. Now we consider the following pre-processing of the ciphertexts. In our applications, we assume w to be even, which is without loss of generality. Importantly, for the best amortized efficiency, we require the constraint $hv \le r$, where r (which can be set to $O(\lambda^{0.25-o(1)})$) is the number of slots supports by the framework [8]. Intuitively, this means that we have a sufficient number of slots to pack the inputs.

Let $\overrightarrow{\mathbf{C}_{ki}} := (\mathbf{C}_{k,(1,i)}, \mathbf{C}_{k,(2,i)}, \cdots, \mathbf{C}_{k,(h,i)})^\top$ as the i-th column vector of $\{\mathbf{C}_{k,(i,j)}\}_{i\in[h],j\in[w]}$. Then the pre-processing step pre-computes the following packed ciphertexts.

Pre-computing $\mathbf{B}_{ki}$'s. We pack the column vectors into mode "$\mathcal{R}_{12} \to \mathcal{R}_{13}$" and mode "$\mathcal{R}_{12} \to \mathcal{R}_{13}$", alternately. Let $\boldsymbol{\eta}_k \in \mathbb{Z}^v$ be the vector with only one 1 in the k-th entry and 0 elsewhere, i.e., $(0, 0, \ldots, 1, 0, \ldots, 0)^\top$. Then we compute:

$$\begin{aligned}
\mathbf{B}_{k1} &= \mathsf{RGSW\text{-}Pack}(\overrightarrow{\mathbf{C}_{k1}} \otimes \boldsymbol{\eta}_k, \text{"}\mathcal{R}_{12} \to \mathcal{R}_{13}\text{"}) \\
\mathbf{B}_{k2} &= \mathsf{RGSW\text{-}Pack}(\overrightarrow{\mathbf{C}_{k2}} \otimes \boldsymbol{\eta}_k, \text{"}\mathcal{R}_{13} \to \mathcal{R}_{12}\text{"}) \\
&\cdots \\
\mathbf{B}_{kw} &= \mathsf{RGSW\text{-}Pack}(\overrightarrow{\mathbf{C}_{kw}} \otimes \boldsymbol{\eta}_k, \text{"}\mathcal{R}_{13} \to \mathcal{R}_{12}\text{"})
\end{aligned}$$

Moreover, we set

$$\begin{aligned}
\mathbf{G}_0 &= \mathsf{RGSW\text{-}Pack}(\mathbf{G}, \mathbf{G}, \cdots, \mathbf{G}, \text{"}\mathcal{R}_{13} \to \mathcal{R}_{12}\text{"}) \\
\mathbf{G}_1 &= \mathsf{RGSW\text{-}Pack}(\mathbf{G}, \mathbf{G}, \cdots, \mathbf{G}, \text{"}\mathcal{R}_{12} \to \mathcal{R}_{13}\text{"})
\end{aligned}$$

Algorithm 4.1: $\mathsf{VecMatMult}(\cdot,\cdot)$

Input :
- $\boldsymbol{a}_k \in \{0,1\}^w$, for $k \in [v]$
- a vector of (pre-processed) packed RGSW ciphertext $\{\mathbf{B}_{ki}\}_{i\in[w],k\in[v]}$.

Output : a ciphertext vector $\overrightarrow{\mathbf{C}} \in \mathsf{RGSW.EncVec\text{-}Exp}(\boldsymbol{z}_1||\cdots||\boldsymbol{z}_v)$, where $\boldsymbol{z}_k = \mathbf{M}_k \cdot \boldsymbol{a}_k$.

$\mathsf{ACC}_0 = \mathsf{RGSW\text{-}Pack}(\mathbf{G}, \mathbf{G}, \cdots, \mathbf{G}, \text{“}\mathcal{R}_{12}\text{”})$;
for $i = 1$ **to** w **do**
  $\mathbf{B}_i = \sum_{k\in[v]} \big(\boldsymbol{a}_k[i] \cdot \mathbf{B}_{ki} + (1 - \boldsymbol{a}_k[i]) \cdot \mathbf{G}_{(i \mod 2)}\big)$;
  ACC_i=$\mathsf{Batch\text{-}Mult}(\mathsf{ACC}_{i-1}, \mathbf{B}_i)$;
$\overrightarrow{\mathbf{C}} = \mathsf{UnPack}(\mathsf{ACC}_w)$;
Return: $\overrightarrow{\mathbf{C}}$;

Next we present Algorithm 4.1 and Theorem 4.1 to capture the the correctness and error growth. As the proof technique is similar to that of [8], due to the space limit, we defer the proof to the full version.

Theorem 4.1 *Algorithm 4.1 satisfies the correctness as required by the input/output specification. Moreover, let s be the secret of the* RGSW *scheme and E be the upper bound (infinity norm of the canonical embedding) of errors in all evaluation keys and the packed* RGSW *ciphertexts in $\{\mathbf{B}_{ki}\}$. Then $\max \|\mathsf{Err}(\overrightarrow{\mathbf{C}}[i])\|_\infty$ is bounded by a sub-Gaussian variable with parameter $O(\gamma)$ such that $\gamma \le wvr^2\sqrt{N \log Q} \cdot \|s\|_\infty \cdot E$.*

Complexity. The preprocessing step takes wv RGSW-Pack packing, and thus requires $O(wvr)$ RGSW additions. For the online computation, we have $O(wv)$ RGSW additions and w Batch-Mult's in the for loop. Then we compute UnPack(), which is roughly vh Batch-Mult's. We notice that one Batch-Mult is roughly $O(\log \lambda)$ RGSW multiplications. Thus in total, we have $O(wv)$ RGSW additions and $O((w + vh)\log \lambda)$ RGSW multiplications. In amortization, this would be $O(wv/(vh))$ RGSW additions and $O((w+vh)\log \lambda/(vh))$ RGSW multiplications, per dimension (over h) per vector-matrix pair (over v).

4.2 Multiplications over Small(er) Rings

Now we show how to achieve a batch homomorphic multiplication over $\mathbb{Z}[\xi_{2d}]$ for $2dv < r$ with good amortized complexity, by using the homomorphic method as we developed above. Particularly, let d be a power of two such that $2dv < r$, and $\{a_k\}_{k\in[v]}, \{x_k\}_{k\in[v]}$ be ring elements over $\mathbb{Z}[\xi_{2d}]$. We consider the task of homomorphic computation of $\{a_k x_k\}_{k\in[v]}$ where each $a_k \in \mathbb{Z}[\xi_{2d}]$ is in the clear and $x_k \in \mathbb{Z}[\xi_{2d}]$ is encrypted in some form, as we formalize below.

Task Specifications. Let $\mathbf{X}_k = \mathsf{Coeffs\text{-}Rot}(x_k) \in \mathbb{Z}_q^{2d\times 2d}$ be the anti-cyclic rotation matrix of x_k for $k \in [v]$. Set the power-of-two matrix, i.e., $\mathbf{M}_k = \boldsymbol{g}^\top \otimes \mathbf{X}_k \in \mathbb{Z}_q^{2d\times 2d\log q}$, and generate RGSW ciphertexts $\{\mathbf{C}_{k,(i,j)}\}_{i\in[2d],j\in[2d\log q],k\in[v]}$,

each of which encrypts the corresponding entry $\mathbf{M}_k[i,j]$ of $\mathbf{M}_k$. Finally, let $\{\mathbf{B}_{kj}\}_{j\in[2d\log q],k\in[v]}$ be the packed ciphertext as computed in the pre-processing of Section 4.1. Now we formally present the task statement:

- **Input**: Let $a_1,\ldots,a_v \in \mathbb{Z}_q[\xi_{2d}]$ and $\{\mathbf{B}_{kj}\}_{j\in[2d\log q],k\in[v]}$ be the packed ciphertexts that represent the pre-processed ciphertext of $x_k \in \mathbb{Z}_q[\xi_{2d}]$.
- **Output**: $(\overrightarrow{\mathbf{C}}'_1, \overrightarrow{\mathbf{C}}'_2, \ldots, \overrightarrow{\mathbf{C}}'_v)$ such that for each $k\in[v]$, $\overrightarrow{\mathbf{C}}'_k[i] \in \mathsf{RGSW}(m_k[i])$, $m_k[i] = \xi_q^{\boldsymbol{z}_k[i]}$ and $\boldsymbol{z}_k = \mathsf{coeffs}(a_k \cdot x_k)$.

This task can be achieved easily given Algorithm 4.1 as we present below.

Algorithm 4.2: Multiplications over Small(er) Rings

Input : $a_1,\ldots,a_v \in \mathbb{Z}_q[\xi_{2d}]$ and $\{\mathbf{B}_{kj}\}_{j\in[2d\log q],k\in[v]}$ (as specified above).
Output : $(\overrightarrow{\mathbf{C}}'_1,\ldots,\overrightarrow{\mathbf{C}}'_v)$ (as specified above).

1 $\boldsymbol{a}'_k = \boldsymbol{g}^{-1}(\mathsf{coeffs}(a_k))$, for $k\in[v]$;
2 **Return:** $\mathsf{VecMatMult}\left(\{\boldsymbol{a}'_k\}, \{\mathbf{B}_{kj}\}_{k\in[v],j\in[2d\log q]}\right)$ (setting $h=2d$, $w=2d\log q$ in Algorithm 4.1).

Theorem 4.2 *The above algorithm satisfies the correctness as required by the input/output specification.*

This theorem simply follows from Theorem 4.1.

Complexity. The complexity of this algorithm follows essentially the same as that of Algorithm 4.1, by setting $h=2d$, $w=2d\log q$. Assuming that $d=\lambda^{O(1)}$, $v=\lambda^{O(1)}$, $q=\mathsf{poly}(\lambda)$, then the amortized complexity of the online computation would be $O(\log\lambda)$ RGSW additions, and $O(\log\lambda)$ RGSW multiplications. This can be simplified as $O(\log\lambda)=\tilde{O}(1)$ RGSW multiplications, per dimension (over $2d$) per ring multiplication (over v).

5 Homomorphic DFT/inverse-DFT

In this section, we consider another form of batch vector-matrix multiplication where the vector is encrypted and the matrix is in the clear yet of some special form, where each entry is a power of a root of unity. By setting the matrix to the DFT (or respectively DFT^{-1}) matrix, this task would immediately give a batch homomorphic DFT/inverse-DFT, which is another important building block of the bootstrapping framework of MS18 [12]. Here we present an efficient batch DFT/inverse-DFT with dimension $2d<r$ where $r=O(\lambda^{0.25-o(1)})$ is the number of slots that the batch framework can support. Then in Section 6, we show that a recursive optimization can be further applied for larger dimensions, e.g., $O(\lambda)$ as required by the bootstrapping. Below we present a detailed formulation.

Setting. Let $m>d$ be two numbers of **powers of two**. Clearly, we have $2d|m$, and thus $\mathbb{Z}[\xi_{2d}]$ is a sub-ring of $\mathbb{Z}[\xi_m]$. Let $\mathbf{M}\in\mathbb{Z}_q[\xi_{2d}]^{2d\times 2d}$ be a matrix where each entry is some power of the $2d$-th root of unity, i.e., each $\mathbf{M}[i,j]=\xi_{2d}^{\delta_{ij}}$ for $\delta_{ij}\in\mathbb{Z}_{2d}$, and let $\boldsymbol{a}\in\mathbb{Z}_q[\xi_m]^{2d}$ be a vector of elements in the ring of the extension field. This task is to homomorphically compute $\mathbf{M}\cdot\boldsymbol{a}$, where $\mathbf{M}$ is in the clear and $\boldsymbol{a}$ is encrypted in some form as specified next.

Basic Facts. We first describe some useful facts in the algebraic number theory. We know that $d' = m/(2d)$ is the degree of field extension $\mathbb{Q}(\xi_m)/\mathbb{Q}(\xi_{2d})$. Then for any $x \in \mathbb{Z}[\xi_m]$, we can uniquely represent x as d' $\mathbb{Z}[\xi_{2d}]$-coefficients (say, $x_0, x_1, \ldots, x_{d'-1} \in \mathbb{Z}[\xi_{2d}]^{d'}$) over some $\mathbb{Q}(\xi_{2d})$-basis of $\mathbb{Q}(\xi_m)$, e.g., $\{1, \xi_m, \xi_m^2, \ldots, \xi_m^{d'-1}\}$, meaning that $x = \sum_{0 \le i < d'} x_i \xi_m^i$. The coefficients can be viewed as a vector space with coefficients in $\mathbb{Z}[\xi_{2d}]$, i.e., for any $x' \in \mathbb{Z}[\xi_{2d}]$, we have $x' \cdot x = \sum_{0 \le i < d'} (x' \cdot x_i)\xi_m^i$, whose $\mathbb{Z}[\xi_{2d}]$-coefficients are $(x'x_0, \ldots, x'x_{d'-1})$.

We notice that the matrix $\mathbf{M}$ in the setting suffices to capture the case of DFT/inverse-DFT, as the DFT matrix (of dimension $2d$) can be expressed as

$$\mathbf{M}_{\mathsf{DFT}} = \begin{pmatrix} \xi_{2d}^{1\cdot 1} & \xi_{2d}^{1\cdot 1} & \cdots & \xi_{2d}^{1\cdot 2d} \\ \xi_{2d}^{2\cdot 1} & \xi_{2d}^{2\cdot 1} & \cdots & \xi_{2d}^{2\cdot 2d} \\ \cdots & & & \\ \xi_{2d}^{2d\cdot 1} & \xi_{2d}^{2d\cdot 2} & \cdots & \xi_{2d}^{2d\cdot 2d} \end{pmatrix}.$$

The inverse DFT matrix can be expressed as $\mathbf{M}_{\mathsf{DFT}^{-1}} = (2d)^{-1} \cdot \mathbf{M}_{\mathsf{DFT}}^*$ where $*$ denotes the conjugate. We notice that for the homomorphic DFT^{-1} over the exponent (over $\mathcal{R}_1 = \mathbb{Z}[\xi_q]$), we need that $2d$ to be relatively prime to q, and thus $(2d)^{-1}$ exists when taking modulo q. In our setting, this is not a problem as we can set q to be a prime. For the work [12], they use power-of-two q as required by the FHEW framework [5]. In this case, they would need to change the degree of DFT into 3's powers.

Next we present the details of the task.

Task Specifications. Now we specify how $\boldsymbol{a} = (a_1, \ldots, a_{2d}) \in \mathbb{Z}_q[\xi_m]^{2d}$ is encrypted. For each $i \in [2d]$, we represent $a_i = \sum_{0 \le j < d'} a_{ij} \xi_m^j$ where each $a_{ij} \in \mathbb{Z}[\xi_{2d}]$. Similar to the indexing principle as we used for vectors/matrices, we let $a_{(2d)(j)} = a_{0j}$ and $a_{(i)(d')} = a_{i0}$ for general i, j's. Then we denote $\overrightarrow{\mathbf{C}}_{ij} \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{coeffs}(a_{ij}))$ for $i \in [2d], j \in [d']$. Given these ciphertexts, we formally describe the task statement:

- **Input**: (1) $\mathbf{M} \in \mathbb{Z}_q[\xi_{2d}]^{2d \times 2d}$ such that each $\mathbf{M}[i, j] = \xi_{2d}^{\delta_{ij}}$ for $\delta_{ij} \in \mathbb{Z}_{2d}$. (2) $\left\{ \overrightarrow{\mathbf{C}}_{ij} \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{coeffs}(a_{ij})) \right\}_{i \in [2d], j \in [d']}$
- **Output**: $\{\overrightarrow{\mathbf{C}}'_{ij}\}_{i \in [2d], j \in [d']}$ such that $\overrightarrow{\mathbf{C}}'_{ij} \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{coeffs}(z_{ij}))$ for each $i \in [2d], j \in [d']$ with the following conditions. Let each $z_i \in \mathbb{Z}_q[\xi_m]$ be the i-th element entry of $\mathbf{M} \cdot \boldsymbol{a}$, i.e., $z_i = \mathbf{M} \cdot \boldsymbol{a}[i] \in \mathbb{Z}_q[\xi_m]$, and $(z_{i0}, \ldots, z_{i(d'-1)}) \in \mathbb{Z}_q[\xi_{2d}]^{d'}$ be its unique coefficient representation with respect to the power basis, i.e., $z_i = \sum_{0 \le j < d'} z_{ij} \xi_{2d}^j$. Note: for convenience, the following two variables are equivalent $z_{id'} := z_{i0}$, as the indexing rule of vectors/matrices.

(*) Importantly, in this section we assume $2d < r$, where r is the number of slots the batch framework supports. Intuitively, this allows us to encrypt all the coefficients of $a_{ij} \in \mathbb{Z}[\xi_{2d}]$ in one packed RGSW cipherext.

5.1 First Attempt

At a first sight, the main task can be achieved by using the prior batch vector-matrix multiplication (Algorithm 4.1). Following this intuition, below we present an attempt that would almost achieve our task, yet would require too many homomorphic additions. Even though unsatisfactory, this attempt still gives insights that lead to our further improvements in the next section. Thus, we still present this algorithm here as a good warm up for the readers.

Recall the following procedure from Section 2.4: there is an efficient homomorphic procedure $\mathsf{Anti\text{-}Rot}(\cdot,\cdot)$ that given as inputs $\overrightarrow{\mathbf{C}} \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{coeffs}(z))$ for $z \in \mathbb{Z}[\xi_{2d}]$, and $\delta \in \mathbb{Z}_{2d}$, outputs $\overrightarrow{\mathbf{C}}' \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{coeffs}(z \cdot \xi_{2d}^{\delta}))$. We notice that in the setting power-of-two, $\mathsf{coeffs}(z \cdot \xi_{2d})$ is an anti-cyclic rotation of $\mathsf{coeffs}(z)$, and this procedure can be computed homomorphically. Next we present the algorithm below.

Algorithm 5.1: Batch Vector-Matrix Mult for Special Matrices

Input :
- $\mathbf{M} \in \mathbb{Z}_q[\xi_{2d}]^{2d \times 2d}$, where each entry is a power of ξ_{2d};
- $\left\{\overrightarrow{\mathbf{C}}_{ij} \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{coeffs}(a_{ij}))\right\}_{i\in[2d],j\in[d']}$, as specified above.

Output : $\{\overrightarrow{\mathbf{C}}'_{ij}\}_{i\in[2d],j\in[d']}$ as specified above.

Let $\boldsymbol{v} = (1, 1, \ldots, 1) \in \{0,1\}^{2d}$ be the all-one vector;
for $i = 1$ **to** $2d$ **do**
  for $j = 1$ **to** d' **do**
    for $k = 1$ **to** $2d$ **do**
      set $\overrightarrow{\mathbf{R}}_{kj} = \mathsf{Anti\text{-}Rot}(\overrightarrow{\mathbf{C}}_{kj}, \mathbf{M}[i,k])$;
      set $\mathbf{D}_{kj} = \mathsf{RGSW\text{-}Pack}(\overrightarrow{\mathbf{R}}_{kj})$ (to the appropriate mode, either "$\mathcal{R}_{12} \to \mathcal{R}_{13}$" or "$\mathcal{R}_{13} \to \mathcal{R}_{12}$") ;
    Set $\overrightarrow{\mathbf{C}}'_{ij} = \mathsf{VecMatMult}(\boldsymbol{v}, \{\mathbf{D}_{kj}\}_{i\in[2d]})$;
Return: $\{\overrightarrow{\mathbf{C}}'_{ij}\}_{i\in[2d],j\in[d']}$.

The correctness can be easily verified so we do not present the details. Below we just analyze the complexity and point out a technical subtlety that this algorithm does not satisfy our efficiency requirement.

Complexity Analysis. From the computation, at least the algorithm needs $2dd'$ $\mathsf{VecMatMult}()$, each of which is roughly $2d$ $\mathsf{Batch\text{-}Mult}$, and $4d^2d'$ $\mathsf{RGSW\text{-}Pack}$. As we analyzed, each $\mathsf{Batch\text{-}Mult}$ is roughly $O(d \log \lambda)$ RGSW multiplications, and each $\mathsf{RGSW\text{-}Pack}$ is roughly $O(d)$ RGSW additions. Thus in total this would be $O(d^2d')$ RGSW multiplications and $O(d^3d')$ RGSW additions. In amortization (per ring dimension m and per inverse-DFT dimension $2d$), this would be $\tilde{O}(1)$ RGSW multiplications and $O(d)$ RGSW additions. At first it seems we can neglect the RGSW additions, yet in our parameter setting later, we will require $d = \lambda^{O(1)}$. As a RGSW multiplication is roughly equal to $O(\log \lambda)$ RGSW

additions, then the overall amortized complexity will be dominated by the $O(d)$ RGSW additions. This will prevent us from getting the desired efficiency, i.e., overall $\tilde{O}(1)$ RGSW multiplications for bootstrapping, per input ciphertext.

This drawback comes from Step 6, where the above algorithm needs too many calls to RGSW-Pack. At a high level, we need to perform anti-cyclic rotations on the ciphertexts, and then perform the vector-matrix multiplication. The input matrices $\{\mathbf{D}_{kj}\}$ to VecMatMult() need to be packed in the mode of either "$\mathcal{R}_{12} \to \mathcal{R}_{13}$" or "$\mathcal{R}_{13} \to \mathcal{R}_{12}$", but we do not know how to perform homomorphic anti-cyclic rotations over packed ciphertexts of these modes. Therefore, the above method can only perform anti-cyclic rotations on un-packed ciphertexts (Step 5) and then compute different packed ciphertexts (Step 6) for each call of the vector-matrix multiplication (Step 7).

5.2 New Building Blocks

In this section, we present some useful batch homomorphic algorithms, which will be used as major building blocks for our improved method. Particularly, we identify a new batch homomorphic method to compute anti-cyclic rotations for packed ciphertexts of modes "$\mathcal{R}''_{12}$ and "$\mathcal{R}''_{13}$. Even though this does not solve the challenge described in the prior section[6], later on we will show a new homomorphic method that can incorporate the new homomorphic anti-cyclic algorithm, resulting in the overall improvement.

We now present the task for our new homomorphic method. Let $\boldsymbol{x}$ be some vector, and $\boldsymbol{y}$ be its anti-cyclic rotation, i.e., $\boldsymbol{y} = \mathsf{Anti\text{-}Rot}(\boldsymbol{x})$. Given input a packed ciphertext $\mathbf{C} \in \mathsf{RGSW}(\sum_{i\in[r]} x_i \mathsf{v}_i)$ of mode "$\mathcal{R}''_{12}$ (or "$\mathcal{R}''_{13}$ respectively), our goal is to compute a packed ciphertext $\mathbf{C} \in \mathsf{RGSW}(\sum_{i\in[r]} y_i \mathsf{v}_i)$.

To achieve this, we first consider the following two sub-tasks:

Sub-Task I: Batch Permutation: Given input (1) a permutation $\pi : [r] \to [r] \in S_r$ where S_r is the symmetric group of degree r, and (2) a packed ciphertext $\mathbf{C} \in \mathsf{RGSW}(\sum_{i\in[r]} x_i \mathsf{v}_i)$ of mode "$\mathcal{R}''_{12}$ (or "$\mathcal{R}''_{13}$ respectively), the goal is to compute a packed ciphertext $\mathbf{C}' \in \mathsf{RGSW}(\sum_{i\in[r]} x_{\pi(i)} \mathsf{w}_i)$ (or the other mode respectively). This can be achieved by the Algorithm 5.2.

Algorithm 5.2: Batch-Permute$(\cdot,\cdot)$

Input :
- $\mathbf{C}$, a packed RGSW ciphertext encrypting $\sum_{i\in[r]} x_i \mathsf{v}_i$;
- π, a permutation in the symmetric group S_r.

Output : $\mathbf{C}'$, a packed RGSW ciphertext encrypting $\sum x_i \mathsf{w}_{\pi(i)}$.

1 Let $\mathbf{C}_\pi = \sum_{i\in[r]} \mathsf{v}_i^\vee \mathsf{w}_{\pi(i)}$, i.e., a packed ciphertext of mode $\mathcal{R}_{12} \to \mathcal{R}_{13}$;
2 **Return:** $\mathbf{C}' = \mathsf{Batch\text{-}Mult}(\mathbf{C}, \mathbf{C}_\pi)$.

[6] Recall that the challenge is to homomorphically rotates batch ciphertexts of modes "$\mathcal{R}_{12} \to \mathcal{R}''_{13}$ or "$\mathcal{R}_{13} \to \mathcal{R}''_{12}$.

Sub-Task II: Batch Inverse Automorphism: Given input a packed ciphertext $\mathbf{C} \in \mathsf{RGSW}(\sum_{i\in[r]} \xi_q^{a_i}\mathsf{v}_i)$ of mode "$\mathcal{R}''_{12}$ (or "$\mathcal{R}''_{13}$ respectively), the goal is to compute a packed ciphertext $\mathbf{C}' \in \mathsf{RGSW}(\sum_{i\in[r]} \xi_q^{-a_i}\mathsf{v}_i)$ (or the other mode respectively). In another word, this is to homomorphically *conjugate* the plaintexts (the $\mathcal{R}_1$ part) while keeping the basis $\{\mathsf{v}_i\}$ intact. The can be achieved by Algorithm 5.3.

Algorithm 5.3: $\mathsf{Inv\text{-}Auto}(\cdot)$

Input : $\mathbf{C}$, a packed RGSW ciphertext encrypting $\sum_{i\in[r]} \xi_q^{a_i}\mathsf{v}_i$, i.e., plaintext $\mathsf{Pack}(\xi_q^{a_1}, \cdots, \xi_q^{a_r})$ in mode $\mathcal{R}_{12}$;

Output : $\mathbf{C}'$, a packed RGSW ciphertext encrypting $\sum_{i\in[r]} \xi_q^{-a_i}\mathsf{v}_i$, i.e., $\mathsf{Pack}(\xi_q^{-a_1}, \cdots, \xi_q^{-a_r})$ in mode $\mathcal{R}_{12}$;

1 Let σ be the automorphism of $\mathcal{R}$, satisfying $\xi_q \mapsto \xi_q^{-1}$, $\xi_{\rho'} \mapsto \xi_{\rho'}$ and $\xi_{\tau'} \mapsto \xi_{\tau'}$;
2 Apply σ to each entry of $\mathbf{C}$ and get $\overline{\mathbf{C}}$;
3 **Return:** $\mathsf{RGSW\text{-}KS}(\overline{\mathbf{C}}, \mathsf{evk}^{(\sigma^{-1})})$

Combining the above two algorithms, we can homomorphically evaluate the homomorphic anti-cyclic rotation over packed ciphertexts as Algorithm 5.4.

Algorithm 5.4: $\mathsf{Batch\text{-}Anti\text{-}Rot}(\cdot,\cdot)$

Input :
- $\mathbf{C}$, a packed RGSW ciphertext encrypting $\sum_{i\in[r]} \xi_q^{a_i}\mathsf{v}_i$, i.e., plaintext $\mathsf{Pack}(\xi_q^{a_1}, \cdots, \xi_q^{a_r})$ in mode $\mathcal{R}_{12}$;
- a monomial ξ_q^{δ}.

Output : $\mathbf{C}'$, a packed RGSW ciphertext encrypting $\sum_{i\in[\delta]} \xi_q^{-a_{r-\delta+i}}\mathsf{w}_i + \sum_{i\in[r-\delta]} \xi_q^{a_i}\mathsf{w}_{i+\delta}$, namely, plaintext is $\mathsf{Pack}(\xi_q^{-a_{r-\delta+1}}, \cdots, \xi_q^{-a_r}, \xi_q^{a_1}, \cdots, \xi_q^{a_{r-\delta}})$, the anti-cyclic rotation of the input in mode $\mathcal{R}_{12}$;

1 Let π_δ be the *cyclic* rotation that shifts δ;
2 ACC=$\mathsf{Batch\text{-}Permute}(\mathbf{C}, \pi_\delta)$;
3 ACC'=$\mathsf{Inv\text{-}Auto}(\mathsf{ACC})$;
4 $\mathsf{ACC}_+ = \mathsf{Batch\text{-}Mult}(\mathsf{ACC}, \sum_{i=\delta+1}^{r} \mathsf{v}_i \cdot \mathsf{w}_i^{\vee})$;
5 $\mathsf{ACC}_- = \mathsf{Batch\text{-}Mult}(\mathsf{ACC}', \sum_{i=1}^{\delta} \mathsf{v}_i \cdot \mathsf{w}_i^{\vee})$;
6 $\mathbf{C}' = \mathsf{ACC}_+ + \mathsf{ACC}_-$;
7 **Return:** $\mathbf{C}'$

Theorem 5.1 *The above algorithm satisfies the correctness as required by the input/output specification.*

The correctness can be easily verified. We do not analyze the noise here. Instead, we analyze the overall noise behavior in our Algorithm 5.5, the improved homomorphic anti-cyclic rotation.

5.3 Our Improved Method

Now we describe our new improved algorithm to achieve the main task of this section, by using the new algorithms in Section 5.2 as critical building blocks. We first present some basic ideas for the plaintext computation, and then the homomorphic method.

We use the following (simplified) example to illustrate our core idea. Given $\boldsymbol{b} = (b_1, \ldots, b_{2d}) \in \mathbb{Z}_q[\xi_{2d}]^{2d}$ and $\boldsymbol{x} = (\xi_{2d}^{\delta_1}, \ldots, \xi_{2d}^{\delta_{2d}})$, the task is to compute the inner product $\langle \boldsymbol{b}, \boldsymbol{x} \rangle$. In the homomorphic computation, $\boldsymbol{b}$ is encrypted and $\boldsymbol{x}$ is in the clear. At a high level, Algorithm 5.1 performs the computation as: $\sum_{i \in [2d]} \mathsf{coeffs}\left(b_i \cdot \xi_{2d}^{\delta_i}\right)$. Even though the term $\mathsf{coeffs}\left(b_i \cdot \xi_{2d}^{\delta_i}\right)$ can be (homomorphically) computed by using $\mathsf{Anti\text{-}Rot}$, i.e., permuting the coefficients and negating some of them properly, the homomorphic algorithm requires to pack the coefficients in every $\mathsf{coeffs}\left(b_i \cdot \xi_{2d}^{\delta_i}\right)$ in mode $\mathcal{R}_{12} \to \mathcal{R}_{13}$ or $\mathcal{R}_{13} \to \mathcal{R}_{12}$, resulting in the undesired $O(d^3 d')$ homomorphic additions. It is unclear whether there is an efficient homomorphic method transforming a ciphertext of $\mathsf{Pack}(\mathsf{coeffs}(b_i))$ into another of $\mathsf{Pack}(\mathsf{Anti\text{-}Rot}(\mathsf{coeffs}(b_i), \xi_{2d}^{\delta_i}))$, under the modes $\mathcal{R}_{12} \to \mathcal{R}_{13}$ or $\mathcal{R}_{13} \to \mathcal{R}_{12}$. It is important to notice that our algorithm in the above section (Section 5.2) can compute the anti-cyclic rotation for ciphertexts of mode $\mathcal{R}_{12}$ or $\mathcal{R}_{13}$ but not $\mathcal{R}_{12} \to \mathcal{R}_{13}$ or $\mathcal{R}_{13} \to \mathcal{R}_{12}$.

To tackle the above challenge, we consider another way of computation. First we observe that the inner product $\langle \boldsymbol{b}, \boldsymbol{x} \rangle$ can be re-expressed as computing g_{2d} recursively as follow: $g_1 = b_1 \cdot \xi_{2d}^{\delta_1 - \delta_2}$, for $1 < j < 2d$, $g_j = (g_{j-1} + b_j) \cdot \xi_{2d}^{\delta_j - \delta_{j+1}}$, $g_{2d} = (g_{2d-1} + b_{2d}) \cdot \xi_{2d}^{\delta_{2d}}$. It is not hard to verify that these two computation methods are equivalent, producing the same value for any $\boldsymbol{b}$ and $\boldsymbol{x}$.

Now, we can homomorphically compute the above sequence using an ACC storing each g_j in mode either $\mathcal{R}_{12}$ or $\mathcal{R}_{13}$. Let consider an example where g_j is stored in mode $\mathcal{R}_{12}$. Now suppose the ciphertexts that encrypt the coefficients of b_{j+1} are packed into mode $\mathcal{R}_{12} \to \mathcal{R}_{13}$. Then we can compute a ciphertext $\mathbf{D}$ of g_{j+1} in mode $\mathcal{R}_{13}$, and then apply the homomorphic $\mathsf{Anti\text{-}Rot}$ on $\mathbf{D}$ to update the ACC, which works because $\mathbf{D}$ is in mode $\mathcal{R}_{13}$. Proceeding in this way, the final ACC would be a ciphertext that encrypts $g_{2d} = \langle \boldsymbol{b}, \boldsymbol{x} \rangle$. We formalize the idea into Algorithm 5.5.

Theorem 5.2 summarizes the correctness and error growth. As the proof is similar to that of Theorem 4.1 (though much tedious), due to space limit we defer the proof to the full version.

Theorem 5.2 *The above algorithm satisfies the correctness as required by the main task specification in this section .*

Moreover, let s be the secret of the RGSW *scheme and E be the upper bound (infinity norm of the canonical embedding) of errors in all evaluation keys and the packed* RGSW *ciphertexts in $\left\{\overrightarrow{\mathbf{C}}_{ij}\right\}_{i \in [2d], j \in [d']}$. Then the error of each* RGSW *ciphertext in $\left\{\overrightarrow{\mathbf{C}}'_{ij}\right\}_{i \in [2d], j \in [d']}$ is bounded by a sub-Gaussian variable with parameter $O(\gamma)$ such that $\gamma \leq dr^3 \|s\| \sqrt{N \log Q} E$.*

Algorithm 5.5: $\mathsf{RGSW.EncVec\text{-}MatMult}(\cdot,\cdot)$

Input :
1. $\mathbf{M} \in \mathbb{Z}_q[\xi_{2d}]^{2d\times 2d}$ where each entry is a power of ξ_{2d}.
2. $\left\{\overrightarrow{\mathbf{C}}_{kj} \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{coeffs}(a_{kj}))\right\}_{k\in[2d],j\in[d']}$, as above in the task specifications.

Output : $\{\overrightarrow{\mathbf{C}}'_{ij}\}_{i\in[2d],j\in[d']}$ as required above.

for $j = 1$ **to** d', $k = 1$ **to** $2d$ **do**
  $\mathbf{C}_{kj}$=RGSW-Pack($\overrightarrow{\mathbf{C}}_{kj}$, "$\mathcal{R}_{1,(3-(k \bmod 2))} \to \mathcal{R}_{1,(2+(k \bmod 2))}$");
Initialize ACC_0=RGSW-Pack($\mathbf{G},\cdots,\mathbf{G}$,"$\mathcal{R}_{12}$");
for $i = 1$ **to** $2d$ **do**
  for $j = 1$ **to** d' **do**
    ACC_R=ACC_0;
    for $k = 1$ **to** $2d-1$ **do**
      $\mathsf{ACC}_M = \mathsf{Batch\text{-}Mult}(\mathsf{ACC}_R, \mathbf{C}_{kj})$;
      $\mathsf{ACC}_R = \mathsf{Batch\text{-}Anti\text{-}Rot}(\mathsf{ACC}_M, \mathbf{M}[i,k]/\mathbf{M}[i,k+1])$;
    $\mathsf{ACC}_M = \mathsf{Batch\text{-}Mult}(\mathsf{ACC}_R, \mathbf{C}_{(2d)j})$;
    $\mathsf{ACC}_R = \mathsf{Batch\text{-}Anti\text{-}Rot}(\mathsf{ACC}_M, \mathbf{M}[i,2d])$;
    Set $\overrightarrow{\mathbf{C}}'_{ij} = \mathsf{UnPack}(\mathsf{ACC}_R)$;
Return: $\{\overrightarrow{\mathbf{C}}'_{ij}\}_{i\in[2d],j\in[d']}$.

Efficiency. The algorithm makes $O(d'd)$ calls to RGSW-Pack, $O(d^2d')$ calls to Batch-Mult, and $O(dd')$ calls to UnPack. Similar to the prior analysis, this would be upper bounded by $O(d^2d')$ RGSW additions and $\tilde{O}(d^2d')$ RGSW multiplications, which is dominated by $O(d^2d')$ RGSW multiplications. Thus, in amortization (per ring dimension $m = 2dd'$ per inverse-DFT dimension $2d$) the cost would be $\tilde{O}(1)$ RGSW multiplications.

6 Homomorphic DFT^{-1}, Recursively

The multiplication of the DFT/inverse-DFT matrix is a critical step for realizing the recursive DFT/inverse-DFT. In this section, we show how to achieve a homomorphic DFT/inverse-DFT by applying the method in Section 5 recursively, via the Nussbaumer Transform as identified by MS18 [12]. First we present the Nussbaumer Transform using the language of algebraic extension, which might give a better intuition than the polynomial representation as used in the work [12].

6.1 Nussbaumer Transform

Let $K \supseteq E \supseteq \mathbb{Q}$ be towers of field extensions, where $K = \mathbb{Q}(\xi_n)$, $E = \mathbb{Q}(\xi_{n'})$ for $n > n'$. Denote $d = n/n'$ be the degree of the field extension K/E, and we assume that $2d \mid n'$, which is required by the DFT framework as we further elaborate later. At a high level, Nussbaumer Transform shows that the multiplication operation (of two elements) in K can be reduced to $2 \cdot d$ multiplications of elements in E in the following way.

First we observe that K is a field extension over E that can be expressed in a polynomial quotient ring with coefficients in E, i.e., $K \cong E[X]/(X^d - \xi_{n'})$,

where $\xi_{n'} \in E$ is the n'-th root of unity. Then any two ring elements $a, b \in K$ can be expressed as $a \cong a(X) = a_0 + a_1 X + \cdots + a_{d-1} X^{d-1}$ and $b \cong b(X) = b_0 + b_1 X + \cdots + b_{d-1} X^{d-1}$, where all the coefficients are in E. In this way, the multiplication of $a \cdot b \in K$ can be computed equivalently as $a(X) \cdot b(X) \mod (X^d - \xi_{n'}) \in E[X]/(X^d - \xi_{n'})$.

To compute $a(X) \cdot b(X)$ in the DFT manner, we notice that the $2d$-th root of unity, denoted as $\omega = \xi_{2d}$, would be used. If the underlying coefficients are in $\mathbb{Q}$, then we inherently need to work with complex numbers, which are not compatible with the FHEW-based computation. Interestingly, if the coefficients are in E, then we do have $\omega = \xi_{2d} \in E = \mathbb{Q}(\xi_{n'})$ as long as $2d \mid n'$. Now we can compute DFT-based multiplication with integral coefficients. This is compatible with FHEW as demonstrated by [12], and can be further batched by our framework as we will show in the next section. Now we present the DFT-based multiplication in details.

- Convert $a(X)$ in the DFT representation as $(a(\omega^0), \ldots, a(\omega^{2d-1})) \in E^{2d}$, and similarly $b(X)$ as $(b(\omega^0), \ldots, b(\omega^{2d-1})) \in E^{2d}$.
- Multiply the vectors component-wisely, and obtain $(c(\omega^0), \ldots, c(\omega^{2d-1}))$.
- Convert the resulting vector back to the polynomial $c(X)$ using inverse DFT, and then output $c(X) \mod (X^d - \xi_{n'})$.

We notice that $a(X) \cdot b(X)$ is a polynomial of degree at most $2(d-1) < 2d$, and therefore, can be uniquely interpolated from the DFT representation of $2d$ points. Thus, $c(X) \mod X^d - \xi_{n'}$ would give the correct answer. We notice that there is a natural mapping from $c(X)$ to the ring element in K, namely $c(X) \mapsto c(\xi_n)$, which can be thought as plugging in X with ξ_n.

Recursive Optimizations. This process can be further optimized by recursively computing multiplications in E, as long as there is another intermediate field $E \supset F \supset \mathbb{Q}$. Under this observation, a tower structure would give the best performance for recursion. Particularly, we assume that $n = 2d^\rho$ for some integer $\rho > 1$, and n, d are both two's power, i.e., $d = 2^\delta$, and $n = 2^{\delta\rho+1}$. This allows us to present the recursive optimization in a clean way.

Let DFT_{2d} be an algorithm parameterized by $2d$ and DFT_{2d}^{-1} be the inverse-DFT algorithm parameterized by $2d$. We present these algorithms in Algorithms 6.1 and 6.2. The correctness and efficiency can be easily verified and analyzed. Here our description can help readers get better intuitions of the homomorphic version of the inverse-DFT.

Importantly, these two algorithms are defined for general (recursive) inputs, which can be of varying input lengths. In the first level, DFT_{2d} takes input $a \in \mathbb{Z}[\xi_n]$, and DFT_{2d}^{-1} takes input $\boldsymbol{a} \in \mathbb{Z}[\xi_{2d}]^{(2d)^{\rho-1}}$.

6.2 Homomorphic Evaluation

Now we present the homomorphic algorithm for the recursive inverse-DFT as Aglrotihm 6.3. Here we describe the specifications.

- **Input**: The input contains ciphertext vectors that encrypts $\boldsymbol{a} \in \mathbb{Z}[\xi_{n'}]^{(2d)^{\rho'-1}}$, where $\boldsymbol{a}$ is the input of the plaintext-based algorithm DFT^{-1} as Algorithm 6.2.

Algorithm 6.1: DFT_{2d}

Input : $a \in \mathbb{Q}(\xi_{n'})$ for $n' = 2d^{\rho'}$
Output : $\boldsymbol{a} \in \mathbb{Q}(\xi_{2d})^{(2d)^{\rho'-1}}$

if $n' = 2d$ **then**
  return a
else
  represent $a \cong a(X) = a_0 + a_1 X + \cdots + a_{d-1} X^{d-1}$, where each $a_i \in \mathbb{Q}(\xi_{n'/d})$;
  let $\omega = \xi_{2d} \in \mathbb{Q}(\xi_{n'/d})$;
  compute $(a'_0, \ldots, a'_{2d-1}) := (a_0, \ldots, a_{d-1}, 0, \ldots, 0) \cdot \mathbf{M}_{\mathsf{DFT}}$; // $a'_i = a(\omega^i)$
  return $\Big(\mathsf{DFT}_{2d}(a'_1), \ldots, \mathsf{DFT}_{2d}(a'_{2d})\Big)$.

Algorithm 6.2: DFT_{2d}^{-1}

Input : $\boldsymbol{a} \in \mathbb{Q}(\xi_{n'})^{(2d)^{\rho'-1}}$
Output : $a \in \mathbb{Q}(\xi_{n' d^{\rho'-1}})$

if $\rho' = 1$ **then**
  return: $\boldsymbol{a}$;
else
  parse $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{2d})$ where each $\boldsymbol{a}_i \in \mathbb{Q}(\xi_{n'})^{(2d)^{\rho'-2}}$;
  compute $a'_i = \mathsf{DFT}_{2d}^{-1}(\boldsymbol{a}_i)$ for $i \in [2d]$;
  compute $(a_1, \ldots, a_{2d}) = (a'_1, \ldots, a'_{2d}) \cdot \mathbf{M}_{\mathsf{DFT}^{-1}}$;
  compute $a(X) = a_0 + a_1 X + \cdots + a_{2d-1} X^{2d-1} \mod X^d - \xi_{n' d^{\rho'-2}}$, where we set $a_0 = a_{2d}$;
  return $a \cong a(X)$ as an element in the extension field $\mathbb{Q}(\xi_{n' d^{\rho'-1}})$.

Specifically, let ρ', n' be two parameters as implicit inputs, indicating which recursive level the algorithm is at. The (explicit) input contains $(2d)^{\rho'-1}$ ciphertext vectors $\left\{\overrightarrow{\mathbf{C}}_i\right\}_{i \in [(2d)^{\rho'-1}]}$, where each $\overrightarrow{\mathbf{C}}_i \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{coeffs}(\boldsymbol{a}[i]))$, meaning that it encrypts all the coefficients of $\boldsymbol{a}[i]$ in the exponents.

– **Output**: Let $a = \mathsf{DFT}^{-1}(\boldsymbol{a}) \in \mathbb{Z}[\xi_{n' d^{\rho'-1}}]$. The output contains a ciphertext vector $\overrightarrow{\mathbf{C}}' \in \mathsf{RGSW.EncVec\text{-}Exp}(\mathsf{coeffs}(a))$, meaning that it encrypts all the coefficients of a in the exponents.

Before presenting our homomorphic algorithm, we first introduce some useful facts for "change of representation". Let $m \geq 2d$ be two numbers of two's powers. Below, we consider two particular ways to represent a ring element.

- Given ring element $a \in \mathbb{Z}[\xi_m]$, we can express it as a $\mathbb{Z}$-coefficient vector of dimension $m/2$, denoted as $\mathsf{coeffs}(a)$ with respect to the power basis, namely $a \mapsto (a_1, \ldots, a_{m/2}) \in \mathbb{Z}^{m/2}$ such that $a = a_0 + \sum_{i \in [m/2-1]} a_i \xi_m^i$. Using our indexing rule for convenience, we set $a_{m/2}$ as an alias variable of a_0.

- On the other hand, we can also represent a as a $\mathbb{Z}[\xi_{2d}]$-coefficient vector of dimension $m/(2d)$, namely $a \mapsto (a'_1, \ldots, a'_{m/(2d)}) \in \mathbb{Z}[\xi_{2d}]^{m/(2d)}$ such that $a = a'_0 + \sum_{i\in[m/(2d)-1]} a'_i \xi_m^i$. Similarly, we set $a'_{m/(2d)}$ as an alias variable of a'_0. Of course each a'_i can be further expanded by the $\mathbb{Z}$-coefficient representation, namely $a'_i \mapsto (a'_{i1}, \ldots, a'_{id})$ denoted as $\mathsf{coeffs}(a'_i)$.

These two representations are equivalent as they both represent the same ring element. Moreover, the two representations can be mutually converted from one to the other, just by permuting/rearranging the indices. Thus, this also gives an efficient homomorphic method for converting a ciphertext vector $\overrightarrow{\mathbf{C}}$ that encrypts $(a_1, \ldots, a_{m/2})$, into ciphertext vectors $\{\overrightarrow{\mathbf{C}}_i\}_{i\in[m/(2d)]}$ where each $\overrightarrow{\mathbf{C}}_i$ encrypts $\mathsf{coeffs}(a'_i)$, and vice versa. As it just requires to permute the indices, no heavy homomorphic method (even addition) is required.

We can formalize the conversions as the following two algorithms, and their homomorphic variants work analogously.

- Rearr: on input $m \to 2d$, $\mathsf{coeffs}(a) \in \mathbb{Z}^{m/2}$ representing the $\mathbb{Z}$ coefficients of $a \in \mathbb{Z}[\xi_m]$, outputs $\Big(\mathsf{coeffs}(a'_1), \ldots, \mathsf{coeffs}(a'_{m/(2d)})\Big)$;
- $\mathsf{Rev\text{-}Rearr}$: on input $2d \to m$, $\Big(\mathsf{coeffs}(a'_1), \ldots, \mathsf{coeffs}(a'_{m/(2d)})\Big)$ representing $(a'_1, \ldots, a'_{m/(2d)}) \in \mathbb{Z}[\xi_{2d}]^{m/(2d)}$, outputs $\mathsf{coeffs}(a) \in \mathbb{Z}^{m/2}$ where $a \in \mathbb{Z}[\xi_m]$.

We notice that it is natural to present our homomorphic inverse-DFT algorithm using the first representation. However, it is more natural to use the second representation for the improved batch homomorphic multiplication of the inverse-DFT matrix as we designed in Algorithm 5.5. Thus, we introduce the above two efficient conversions to glue these two parts together. Additionally, we also define the following notation for convenience.

Definition 6.1 *Let $a|b$ be integers. Define sets $S_1, \ldots, S_a$ that equally partition $[b]$ as: $S_1 = [b/a]$, $S_2 = [b/a+1, 2b/a], \ldots,$ and $S_a = [(a-1)b/a+1, b]$.*

Now we present our new method in Algorithm 6.3 and summarize the result in Theorem 6.2. Due to space limit, we defer the proof to the full version.

Theorem 6.2 *The above algorithm satisfies the correctness as required by the task specification in this section .*

Moreover, let s be the secret of the RGSW *scheme and E be the upper bound of errors in all evaluation keys and in all* RGSW *ciphertexts in $\left\{\overrightarrow{\mathbf{C}}_{ij}\right\}_{i\in[(2d)^{\rho'-1}], j\in[d']}$. Then the error of each* RGSW *ciphertext in output is bounded by a sub-Gaussian variable with parameter $O(\gamma)$ such that $\gamma \le (dr^3\|s\|N\log Q)^{\rho'}E$.*

Complexity. We can show that the amortized complexity of the recursive version is still $\tilde{O}(1)$ RGSW multiplications. Intuitively, if the multiplication of inverse-DFT matrix has $\tilde{O}(1)$ amortized complexity (which is true for our Algorithm 5.5), then we can achieve $\tilde{O}(\rho)$ amortized complexity where ρ is the

Algorithm 6.3: Hom-DFT^{-1}

Input : Integers ρ', n' and ciphertext vectors $\left\{\overrightarrow{\mathbf{C}}_i\right\}_{i\in[(2d)^{\rho'-1}]}$

Output : A ciphertext vector $\overrightarrow{\mathbf{C}}'$ as specified above.

if $\rho' = 1$ **then**
  Return: $\overrightarrow{\mathbf{C}}$;
else
  Let $S_1, \ldots, S_{2d}$ be the sets that equally partition $[(2d)^{\rho'-1}]$;
  For $i \in [2d]$, compute $\overrightarrow{\mathbf{C}}'_i = \text{Hom-}\mathsf{DFT}^{-1}\left(\{\overrightarrow{\mathbf{C}}_j\}_{j\in S_i}\right)$;
  For $i \in [2d]$, compute $\{\overrightarrow{\mathbf{C}'_{ij}}\}_{j\in[d'']} = \mathsf{Rearr}((2d)^{\rho'-2} \to 2d, \overrightarrow{\mathbf{C}}'_i)$, where $d'' = n'd^{\rho'-2}/(2d)$;
  $\{\overrightarrow{\mathbf{C}}''_{ij}\}_{i\in[2d],j\in[d'']} = \mathsf{RGSW.EncVec\text{-}MatMult}(\mathbf{M}_{\mathsf{DFT}^{-1}}, \{\overrightarrow{\mathbf{C}'_{ij}}\}_{i\in[2d],j\in[d'']})$;
  For $i \in [2d]$, compute $\overrightarrow{\mathbf{C}}''_i = \mathsf{Rev\text{-}Rearr}(2d \to (2d)^{\rho'-2}, \{\overrightarrow{\mathbf{C}}''_{ij}\}_{j\in[d'']})$;
  For $i \in [d+1, 2d]$, $\overrightarrow{\mathbf{C}}''_i = \mathsf{Anti\text{-}Rot}(\overrightarrow{\mathbf{C}}''_i, \xi_{n'd^{\rho'-2}})$;
  for $i = 1$ **to** d **do**
    for $j = 1$ **to** $n'd^{\rho'-2}$ **do**
      $\overrightarrow{\mathbf{C}}''_i[j] = \overrightarrow{\mathbf{C}}''_i[k] \boxdot \overrightarrow{\mathbf{C}}''_{i+d}[k]$;
  $\overrightarrow{\mathbf{C}}' = \mathsf{Rev\text{-}Rearr}((2d)^{\rho'-2} \to (2d)^{\rho'-1}, \{\overrightarrow{\mathbf{C}}''_i\}_{i\in[d]})$;
  Return: $\overrightarrow{\mathbf{C}}'$;

recursive depth. We will set parameters such that $\rho = O(1)$, and thus the overall amortized complexity matches what we claimed. Below we elaborate.

Let $n = (2d)^\rho$ be the final output (ring) dimension. Similar to the analysis of [12], we first identify that that at level i of the recursion, the algorithm makes $(2d)^i$ calls to the $\mathsf{RGSW.EncVec\text{-}MatMult}$ algorithm, with dimension $m^i = (2d)^{\rho-i-1}$. Thus at this level, each call would be $\tilde{O}(m^i \times 2d) = \tilde{O}((2d)^{\rho-i})$ RGSW multiplications, resulting in a total complexity (at this level) $\tilde{O}((2d)^\rho)$ as the setting of parameters. Thus, in total there would be $\tilde{O}(\rho(2d)^\rho)$ RGSW multiplications. For the case where $\rho = O(1)$, this would be $\tilde{O}((2d)^\rho)$, implying the amortized complexity $\tilde{O}(1)$ RGSW multiplications per dimension.

7 Putting Things Together – Faster Bootstrapping

Now we present how to use our new batch algorithms in Sections 4 and 6 to improve MS18 [12], resulting an overall more efficient bootstrapping method.

7.1 MSB Extract and LWE Packing

We recall several building blocks from the literature.

- From [12], there is a conversion algorithm that takes input n LWE ciphertexts (under the same secret key), and outputs $(a, b) \in \mathsf{RLWE}$ with secret $z \in \mathcal{R}$, of dimension n, such that $b - az = \Delta m + e$ where $\mathsf{coeffs}(m)[i]$ corresponds to the i-th message, Δ is some scaling number, e.g., $q/2$, and q is the modulus.

– From [4,5,8,12], there is an algorithm $\mathsf{msbExtract}$ that on input $\mathbf{C} \in \mathsf{RGSW}(\xi_q^m)$ outputs an LWE ciphertext $\boldsymbol{c} \in \mathsf{LWE}(f(m))$, where $f(m)$ denotes the most significant bit. Assuming β is the error bound of the input ciphertext, then the error in the resulting ciphertext is bounded by $O(q\beta)$.

7.2 Our Batch Bootstrapping Method

Parameters. Here are the parameters used in our overall bootstrapping.

– N: the ring dimension of RGSW (the bootstrapping keys).
– n: the dimension of input LWE ciphertexts, and number of input ciphertexts. Here we set n to be a power of two.
– q: the input LWE modulus. In the batch framework, $\mathcal{R}_1$ is set to be $\mathbb{Z}[\xi_q]$.
– $2d$: the parameter of $\mathsf{DFT}/\mathsf{DFT}^{-1}$. We set d to be a power of two.
– ρ: the depth of the recursive algorithm. We set $n = 2d^\rho$.
– r: the maximal number of slots we can packed in the batch framework of [8].
– v: the number of inputs in Algorithm 4.2. We require $r > 2dv$.

The Bootstrapping Algorithm. We first present how the bootstrapping keys are constructed. Let z be the secret of the RLWE ciphertext derived from packing n input LWE ciphertexts. Let $(z_1, \ldots, z_{(2d)^{\rho-1}}) = \mathsf{DFT}(z)$, $\mathbf{Z}_k = \mathsf{Coeffs\text{-}Rot}(z_k) \in \mathbb{Z}_q^{2d\times 2d}$ be the anti-cyclic rotation matrix of z_k, and the corresponding power-of-two matrix, i.e., $\mathbf{M}_k = \boldsymbol{g}^\top \otimes \mathbf{X}_k \in \mathbb{Z}_q^{2d\times 2d\log q}$. Then we generate RGSW ciphertexts $\{\mathbf{C}_{k,(i,j)}\}_{i\in[2d],j\in[2d\log q],k\in[(2d)^{\rho-1}]}$, each of which encrypts the corresponding entry $\mathbf{M}_k[i,j]$ in the exponents.

Now we equally partition $[(2d)^{\rho-1}]$ into v' sets (ref. Definition 6.1), namely, $U_1, \ldots, U_{v'}$ where $v' = (2d)^{\rho-1}/v$. For each $w \in [v']$, we pack the ciphertexts $\{\mathbf{C}_{k,(i,j)}\}_{i\in[2d],j\in[2d\log q],k\in U_w}$ according to the pre-processing step of Section 4.1, obtaining the resulting packed ciphertext as $\{\mathbf{B}_{kj}^{(w)}\}_{w\in[v'],j\in[2d\log q],k\in U_w}$.

Note. The above step uses many indices, which might look overwhelming. Here we remind readers the high level ideas, which would be helpful in understanding what we are doing. Basically, we first encrypt the (power-or-two) rotation matrices of $(z_1, \ldots, z_{(2d)^{\rho-1}})$ in the exponents as $\{\mathbf{C}_{k,(i,j)}\}_{i\in[2d],j\in[2d\log q],k\in[(2d)^{\rho-1}]}$. To compute multiplications over the sub-ring $\mathbb{Z}[\xi_{2d}]$, we would need to pack these ciphertexts according to Algorithm 4.2, particularly the preprocessing steps in Section 4.1. Given these packed ciphertexts and ring elements $(x_1, \ldots, x_{(2d)^{\rho-1}}) \in \mathbb{Z}[\xi_{2d}]^{(2d)^{\rho-1}}$, we can homomorphically compute the coefficients $(x_1z_1, \ldots, x_{(2d)^{\rho-1}}z_{(2d)^{\rho-1}})$ in the exponents, in a batch way using Algorithm 4.2.

Now we present our batch bootstrapping algorithm in Algorithm 7.1, and Theorem 7.1 to summarize the correctness and noise growth. The proof follows from Theorems 4.1 and 6.2 in a straight-forward way.

Theorem 7.1 *Adapt the notations above. If each $(b_i, \boldsymbol{a}_i)$ in the input is an LWE ciphertext encrypting μ_i, then the output are LWE ciphertexts encrypting μ_i respectively.*

Moreover, let s be the secret of the RGSW scheme, E be the upper bound of errors in all evaluation keys and in all RGSW ciphertexts in $\{\mathbf{B}_{kj}^{(i)}\}_{i\in[v'],j\in[2d\log q],k\in U_i}$. Then the error of each LWE ciphertext in output is bounded by a sub-Gaussian variable with parameter $O(\gamma)$ such that $\gamma \le \sqrt{n}\log q d^\rho r^{3\rho+3}\|s\|^{\rho+1}(N\log Q)^{\rho+1/2}E$.

Algorithm 7.1: Batch Ring Bootstrapping

Input :
- n LWE ciphertexts
- Bootstrapping keys: $\{\mathbf{B}_{kj}^{(i)}\}_{i\in[v'],j\in[2d\log q],k\in U_i}$.

Output : n bootstrapped LWE ciphertexts.

1 Convert n LWE ciphertexts into one RLWE ciphertext $(a(\xi_n), b(\xi_n))$ under some secret $z(\xi_n)$;
2 $(\overline{a_i})_{i\in[(2d)^{\rho-1}]} \leftarrow \mathsf{DFT}(a)$;
3 **for** $i = 1$ **to** v' **do**
4 $\quad (\vec{\mathbf{C}}_{i1}, \cdots, \vec{\mathbf{C}}_{iv}) = \mathsf{VecMatMult}((\overline{a_i})_{i\in U_i}, \{\mathbf{B}_{kj}^{(i)}\}_{i\in[v'],j\in[2d\log q],k\in U_i})$;
5 Set $\vec{\mathbf{C}}'_{(i-1)v+j} = \vec{\mathbf{C}}_{ij}$, for $i \in [v']$ and $j \in [v]$;
6 $\vec{\mathbf{C}}'' = \text{Hom-}\mathsf{DFT}^{-1}(\vec{\mathbf{C}}'_1, \cdots, \vec{\mathbf{C}}'_{(2d)^{\rho-1}})$;
7 For $i \in [n]$, $\vec{\mathbf{C}}''[i] = \vec{\mathbf{C}}''[i] \cdot \xi_q^{b_i}$;
8 For $i \in [n]$, $(b'_i, \boldsymbol{a}'_i) = \mathsf{msbExtract}(\vec{\mathbf{C}}''[i])$;
9 **Return:** $\{(b'_i, \boldsymbol{a}'_i)\}_{i\in[n]}$.

7.3 Efficiency

To analyze the asymptotic efficiency, we first determine all the parameters in terms of the security parameter λ. Similar to our first work [8], we set $n = O(\lambda)$, $q = \tilde{O}(\sqrt{n})$, $N = O(n)$. In this way, the batch parameter can be set as $r \approx O(\sqrt{N/q}) = O(\lambda^{1/4-o(1)})$. Then we set $d = O(\lambda^{0.2})$, $v = O(\lambda^{0.04})$, satisfying $2dv < r$. Finally, we can set $\rho = 5$, which is $O(1)$, meaning that the noise growth in Theorem 7.1 can be bounded by a fixed polynomial. Moreover, we have $n = 2d^\rho = O(\lambda)$.

By plugging these parameters to the above analysis, now we analyze the efficiency of the overall Algorithm 7.1. It requires $\tilde{O}(2dvv') = \tilde{O}(n)$ RGSW multiplications in the first for loop (Lines 3 - 4), as analyzed in Section 4. The Hom-DFT^{-1} would require $\tilde{O}((2d)^\rho) = \tilde{O}(n)$ RGSW multiplications as analyzed in Section 6. The other steps are dominated by these two modules. Thus, the overall complexity is $\tilde{O}(n)$ RGSW multiplications to bootstrap $n = O(\lambda)$ LWE input ciphertexts. In amortization, this would be $\tilde{O}(1)$ RGSW multiplications per input LWE ciphertext as claimed.

Acknowledgement. The authors would like to thank anonymous reviewers for their insightful comments that significantly help improve the presentation. Feng-Hao Liu is supported by NSF CNS-1942400. Han Wang is supported by the National Key R&D Program of China under Grant 2020YFA0712303 and State Key Laboratory of Information Security under Grant TC20221013042.

References

1. J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314. Springer, Heidelberg, Aug. 2014.

2. G. Bonnoron, L. Ducas, and M. Fillinger. Large FHE gates from tensored homomorphic accumulator. In A. Joux, A. Nitaj, and T. Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 217–251. Springer, Heidelberg, May 2018.
3. Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.
4. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Heidelberg, Dec. 2016.
5. L. Ducas and D. Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Heidelberg, Apr. 2015.
6. C. Gentry. Fully homomorphic encryption using ideal lattices. In Mitzenmacher [13], pages 169–178.
7. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, Aug. 2013.
8. F.-H. Liu and H. Wang. Batch bootstrapping I: A new framework for simd bootstrapping in polynomial modulus. In Eurocrypt 2023.
9. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.
10. V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-LWE cryptography. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Heidelberg, May 2013.
11. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, Apr. 2012.
12. D. Micciancio and J. Sorrell. Ring packing and amortized FHEW bootstrapping. In I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 100:1–100:14. Schloss Dagstuhl, July 2018.
13. M. Mitzenmacher, editor. *41st ACM STOC*. ACM Press, May / June 2009.
14. C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Mitzenmacher [13], pages 333–342.
15. C. Peikert. How (not) to instantiate ring-LWE. In V. Zikas and R. De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 411–430. Springer, Heidelberg, Aug. / Sept. 2016.
16. C. Peikert, O. Regev, and N. Stephens-Davidowitz. Pseudorandomness of ring-LWE for any ring and modulus. In H. Hatami, P. McKenzie, and V. King, editors, *49th ACM STOC*, pages 461–473. ACM Press, June 2017.
17. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow and R. Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.