Batch Bootstrapping I:

A New Framework for SIMD Bootstrapping in Polynomial Modulus

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. In this series of work, we aim at improving the bootstrapping paradigm for fully homomorphic encryption (FHE). Our main goal is to show that the amortized cost of bootstrapping within a polynomial modulus only requires $\tilde{O}(1)$ FHE multiplications.

To achieve this, we develop substantial algebraic techniques in two papers. Particularly, the first one (this work) proposes a new mathematical framework for batch homomorphic computation that is compatible with the existing bootstrapping methods of AP14/FHEW/TFHE. To show that our overall method requires only a polynomial modulus, we develop a critical algebraic analysis over noise growth, which might be of independent interest. Overall, the framework yields an amortized complexity $\tilde{O}(\lambda^{0.75})$ FHE multiplications, where λ is the security parameter. This improves the prior methods of AP14/FHEW/TFHE, which required $O(\lambda)$ FHE multiplications in amortization.

Developing many substantial new techniques based on the foundation of this work, the sequel (*Bootstrapping II*, Eurocrypt 2023) shows how to further improve the recursive bootstrapping method of MS18 (Micciancio and Sorrell, ICALP 2018), yielding a substantial theoretical improvement that can potentially lead to more practical methods.

1 Introduction

Fully homomorphic encryption (FHE) allows arbitrary computation over ciphertexts without the need to first decrypt it. The concept was first proposed by [35] back to 1978, and soon numerous applications were noticed, albeit no plausible scheme was known. Thirty years later, Gentry [18] proposed the first plausible scheme that supports general homomorphic computation[4], inspiring many follow-up works, (see [37] for a comprehensive listing), with a wide array of optimizations and as well new applications, such as outsourcing computation, multiparty computation, and many others.

FHE was initially considered as *theoretical only* as the homomorphic operations were prohibitively expensive. During the past years, many exciting new methods were proposed, e.g., [3,4,12,15,17,21], making substantial steps towards practical realizations. A particularly important progress is the improvement of

[4] Homomorphic computation refers to the ability to compute on ciphertexts (encrypted data). A fully homomorphic encryption supports general homomorphic computation, i.e., computation for any arbitrary function.

Gentry's bootstrapping technique, which is currently the only known method to achieve *fully* homomorphic encryption. Originally bootstrapping was extremely impractical, yet after years of efforts, now we can achieve the task within subseconds (amortized), e.g., [15,21,28] by even a simple personal computing system. Thus, FHE with bootstrapping can be practical in some applications [21].

Limitations of Current Bootstrapping Techniques. Despite significant progress, there are some fundamental questions unanswered. Below we summarize the two main approaches in the state of the art, and then their deficiencies.

- Bootstrapping BGV. This line was used (and implemented) by the work [3, 19, 21]. An advantage of this approach is the support of single instruction multiple data (SIMD) operations, and thus can achieve batch computation that bootstraps multiple ciphertexts per operation. However, the method inherently incurs a super-polynomial error, and thus would require a super-polynomial size modulus (e.g., concretely a 400-bit integer [21]), resulting in large bootstrapping keys and thus large storage to perform the homomorphic computation. Moreover, this would require a stronger assumption (i.e., a super-polynomial modulus-to-noise ratio) of the underlying (Ring)-LWE.
- The AP14/FHEW approach. Bootstrapping within a polynomial size modulus was first achieved by [9], and later improved by AP14 [4], and the ring variant FHEW [17] (with other novel optimizations). With further optimizations [6, 13, 15, 23, 28], now bootstrapping a single ciphertext can be computed within 100 milliseconds, with significantly smaller bootstrapping key and memory (compared with the above approach). The methods in this line are modular and thus conceptually simpler, and moreover, can be used to bootstrap *all* currently known (Ring) LWE-based FHE schemes.

  However, all exiting practical methods (in the current libraries) can only bootstrap one single ciphertext per operation, and thus the amortized efficiency does not outperform the above. Particularly, the existing methods [4, 6, 13, 15, 17, 23, 28] require $O(\lambda)$ FHE multiplications to bootstrap one single (LWE) ciphertext, where λ is the security parameter. Some later works [5,29] tried to mitigate this by new designs built on top of the FHEW, but their techniques have several inherent drawbacks, which limit their potential practicality.

  Specifically, the work [5] cannot batch the computation beyond a logarithmic number of ciphertexts. The work MS18 [29] can bootstrap λ (LWE) ciphertexts using roughly $O(3^{1/\epsilon}\lambda^{1+\epsilon})$ FHE multiplications, for any arbitrary constant $\epsilon > 0$, implying an amortized cost $O(3^{1/\epsilon}\lambda^{\epsilon})$ FHE multiplications per ciphertext. Theoretically, ϵ can be set close to 0, e.g., 0.01, at the cost of a large constant, e.g., 3^{100}, exceeding what can be considered as practical by a large margin. Thus, it is unclear whether MS18 [29] can lead to a practical solution that matches their best theoretical indication.

In this series of works, we aim to achieve *the best of both* by breaking the limitations as above. Our overall goal is to bootstrap λ (LWE) ciphertexts by using only $\tilde{O}(\lambda)$ FHE multiplications, meaning that the amortized cost of bootstrapping is essentially the same as that of the FHE multiplication, up to a factor of $\tilde{O}(1)$. Our goal is summarized by the following statement.

Main Goal. Bootstrapping within a polynomial modulus only requires $\tilde{O}(1)$ FHE multiplications in amortization, with a small hidden constant.

Note: the complexity of existing FHE multiplications in the ring settings (set to the same ring dimension) only differs by a (poly)-logarithmic factor, and thus it is without loss of generality to use the number of "FHE multiplications" as a clean measure of efficiency.

The outcome can consequently improve all the bootstrapping-based FHE for general computation, such as FHEW [17] and TFHE [13], and their applications. This would substantially advance the state of the art research.

1.1 Our Contributions

To achieve this, we present our new techniques in a series of two works – the first (*Batch Bootstrapping I*) focuses on the foundation, i.e., the establishment of a new mathematical framework and noise analysis for batch homomorphic computation. The new framework can improve the FHEW [17] and TFHE [13] bootstrapping methods by a factor of $O(\lambda^{0.25-o(1)})$, implying a new bootstrapping method of amortized cost equal to $\tilde{O}(\lambda^{0.75})$ FHE multiplications.

By using the framework of the first work as a solid foundation, the sequel (*Batch Bootstrapping II*) [24] further develops new critical methods to improve main components of the MS18 [29]. Jointly the two works achieve the main goal as stated above. Below we highlight the significant results of the first work, and then give a preview of the sequel for curious readers.

Significant Results of Bootstrapping I (This Work).

- First, we propose a new algebraic framework that naturally supports the batch homomorphic operations of the AP14/FHEW-like frameworks, e.g., FHEW and TFHE.
- Our next contribution is a new and refined algebraic analysis of the noise growth incurred in our new batch framework. We notice that using the existing existing analysis directly on our framework would result in a super-polynomial noise growth. Thus, our refined analysis is a necessary key to achieve batch bootstrapping within a polynomial modulus.
- Quantitatively, our batch framework allows an explicit batch bootstrapping on FHEW/TFHE with $O(\lambda^{1/4-o(1)})$ slots, where λ denotes the security parameter. This means that we can bootstrap λ (LWE) ciphertexts using $\tilde{O}(\lambda^{1.75})$ FHE multiplications, resulting in the following informal theorem.

Theorem 1.1 (Main Result of this Work, Informal) *Bootstrapping within a polynomial modulus requires $\tilde{O}(\lambda^{0.75})$ FHE multiplications in amortization.*

This result can improve the prior methods of AP14, FHEW and TFHE, which required $O(\lambda)$ FHE multiplications in amortization. We notice that in this series of work (i.e., AP14/FHEW/TFHE), the bootstrapping algorithm only requires workspace for computation (excluding the input and bootstrapping key) roughly $O(1)$ FHE ciphertexts. Thus, our framework yields the first non-trivial batch bootstrapping method that only requires workspace $O(1)$ FHE ciphertexts.

If we allow more workspace (e.g., $O(\lambda)$ FHE ciphertexts) for computation, then the MS18 [29] method provides a more asymptotically efficient bootstrapping, with amortized cost $O(3^{1/\epsilon}\lambda^{\epsilon})$ FHE multiplications. As argued however, there is an inherent barrier for a practical realization that matches the best theoretical indication, as it would require to set ϵ close to 0, implying a prohibitively large constant. It is our next target to get rid of the dependency of ϵ in MS18.

A Preview of Bootstrapping II. In our next work [24], we show how to use our batch framework as a key ingredient to improve the MS18 method [29]. Particularly, we apply the technical foundation in this work (along with many new ideas) to the homomorphic Discrete Fourier Transform (DFT) paradigm developed by MS18. The foundation of this work is the crux to achieve our main goal. More details can be found in the sequel [24].

We believe that the new algebraic framework and noise analysis in this work are both important and might find further optimizations and applications. Thus, the foundation can be valuable and deserve its independent merits.

1.2 Technical Overview

Now we present an overview of our new techniques. We first recall the task of bootstrapping and the framework of AP14/FHEW [4, 17] and later work TFHE [13,15], who designed an explicit bootstrapping method within a polynomial modulus. Then we discuss why it is inherently incompatible with existing batch computation. Finally, we present our new insights to break the barriers.

Backgrounds and Challenges

Bootstrapping. When we perform homomorphic operations from existing FHE schemes, the noise in the resulting ciphertext would grow with the number of operation, eventually becoming too big for correct decryption. To proceed homomorphic computation, Gentry [18] invented the bootstrapping technique that *refreshes* the noise. Currently, this is the only way to achieve *fully* homomorphic encryption which supports an arbitrary (polynomial) number of operations.

Briefly speaking, the task can be achieved as follow. Given an input ciphertext that encrypts m (i.e., ct, which might contain a large noise) and some evaluation key evk (i.e., some FHE encryption of the secret key $\mathsf{Enc}(\mathsf{sk})$), the goal is to homomorphically compute the decryption function, i.e., $\mathsf{Enc}(\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}))$[5], which by correctness would yield an encryption of m. Suppose the homomorphic decryption only incurs a small noise, then we have achieved the task.

Bootstrapping within a polynomial modulus was first achieved by [9]. However, the method requires to use the Barrington Theorem to convert an NC1 circuit into a polynomial length branching program, and thus does not give an efficient explicit construction. A subsequent work [4] presented the first explicit construction by using the idea of symmetric group, and FHEW [17] showed how to optimize the approach in the ring setting, extending the prior idea to a group

[5] More precisely, the computation should be denoted as $\mathsf{Eval}(\mathsf{Dec}(\mathsf{ct}, \cdot), \mathsf{Enc}(\mathsf{sk}))$. By correctness, the output ciphertext should belong to $\mathsf{Enc}(m)$, though perhaps distributed differently from a fresh ciphertext.

of roots of unity. Later on, TFHE [13,15] followed this idea and provided further optimizations, e.g., CMUX, external products, and computation over torus.

As mentioned in [28] that FHEW and TFHE are conceptually the same in the core bootstrapping procedure (with different optimizations and implementation techniques), we refer this approach as the name of the earlier work, i.e., AP14/FHEW framework. This framework in our view, gives a conceptually simple and modular approach. Below we present the high level idea.

The AP14/FHEW Framework. The framework takes input an LWE ciphertext $\mathsf{ct} = (b, a)$ that encrypts m and an evaluation key $\mathsf{evk} = \{\mathsf{BK}_i\}$ where each BK_i is a Ring-GSW ciphertext that encrypts the i-th bit of the secret. Let n, q be the LWE dimension and modulus, N, Q be the Ring-GSW dimension and modulus. Without loss of generality, n, q can be set to small quantities, e.g., $n = O(\lambda)$ and $q = \tilde{O}(\sqrt{n})$, via the dimension reduction and modulus switch [4].[6] We emphasize that $q = \tilde{O}(\sqrt{n})$ is an important setting of parameter. The reader should keep this in mind, and we will further elaborate. Moreover, we notice that the LWE ciphertext has the following structure: $b = sa + e + q/2 \cdot m$, where $m \in \{0, 1\}$ is the message, s is the secret key, and e is some perhaps non-small error. Here we do not need to worry about their actual space, and this presentation is sufficient to illustrate the core idea.

The decryption function can be done in two steps: (1) compute $m' = b - sa \mod q$, and (2) output $\mathsf{Round}(m')$ for some appropriate rounding function. This function can be computed by a low-depth function, i.e., NC1, but the question is how to compute it *efficiently* with an explicit procedure. And this is the insight of the AP14/FHEW framework as we present next.

Briefly, the approach identifies that the homomorphic decryption should use computation over the root of unity of cyclotomic rings. More specifically, let us consider a commonly used cyclotomic ring R of degree N of a power of 2. In this case, we can think of R as the polynomial ring $R = \mathbb{Z}[X]/(X^N + 1)$, satisfying $X^{2N} = 1$. Suppose $q|N$, $Y = X^{2N/q}$, and one can homomorphically compute *on the exponent* for the first step of linear operation of the decryption, i.e., obtain $\mathsf{Enc}(Y^{m'})$ where $m' = b - as$. Then we will have $Y^{m'} = Y^{m' \mod q}$, as $\{1, Y, Y^2, \ldots, Y^{q-1}\}$ forms a multiplicative (sub)-group of $\{1, X, X^2, \ldots, X^{2N-1}\}$. Then a very simple and efficient extraction procedure can be derived from the work [1,13], computing $\mathsf{Enc}(f(m'))$ given input $\mathsf{Enc}(Y^{m'})$ for *any arbitrary* $f : \mathbb{Z}_q \to \{0, 1\}$, which includes the non-linear Round function of the decryption procedure. These two insights yield a very efficient bootstrapping that outputs a ciphertext encrypting a single-bit.

Challenge for Batch Computation. As we discuss next, the AP14/FHEW framework is however not compatible with existing batch computation techniques, which heavily rely on the Chinese Remainder Theorem (CRT) decomposition. Roughly speaking, the CRT-based batch method supports computation over some ring R_t that is isomorphic to $\mathbb{Z}_t \times \mathbb{Z}_t \times \cdots \times \mathbb{Z}_t$ for properly chosen

[6] The work [4] sets $q = \tilde{O}(\lambda)$. If we use a randomized rounding for the modulus switch, q can be further reduced to $\tilde{O}(\sqrt{n}) = \tilde{O}(\sqrt{\lambda})$.

modulus t. In this way, we can pack N bits into these N slots, and multiplications and additions over R_t correspond to the component-wise operations over the N slots. This can be used to batch bootstrapping by expressing the decryption function as a boolean circuit as used in prior work [3, 21], though the method would incur a super-polynomial noise growth. On the other hand, the CRT slots are intrinsically different from the cyclotomic structure and thus cannot support the AP14/FHEW framework. This is the current major technical barrier.

Our New Techniques. As discussed, to support batch computation over the AP14/FHEW framework, the scheme must support batch homomorphic computation over the subgroup $\{1, Y, Y^2, \ldots, Y^{q-1}\}$. As a high level, we need a math structure that allows the following packing mechanism: let $\boldsymbol{x} = (x_1, \ldots, x_r)$ and $\boldsymbol{y} = (y_1, \ldots, y_r)$ where each vector component resides in a space containing the cyclotomic subgroup. The packing mechanism can pack $\boldsymbol{x}, \boldsymbol{y}$ into some x, y (in some appropriately designed space) such that $x + y$ corresponds to $\boldsymbol{x} + \boldsymbol{y}$, and $x \bullet y$ (for some operation $\bullet$) corresponds to $\boldsymbol{x} \odot \boldsymbol{y}$, where $\odot$ denotes the component-wise multiplication. This mechanism can then be used to perform the AP14/FHEW bootstrapping in a batch way.

To achieve this, this work proposes a new algebraic framework and refined homomorphic methods/analyses for efficient implementations. Particularly, we first describe our new design of for batch computation over the plaintexts, and then show how to do homomorphic computation with a small noise growth. We will point out multiple technical subtleties and challenges, so a straight-forward adoption of the existing noise analysis would not give a satisfactory solution. Our new analytical insights serve as the critical key.

New Batch Plaintext Computation. So now we present our new insights of a new math structure that supports the above property, by using tensor rings in a novel way. To illustrate our ideas, we first present some insightful yet failed attempts that gradually lead to our final construction.

Attempt 1. Let $\mathcal{R}_1 = \mathbb{Z}_q[X]/(X^q + 1)$ be the cyclotomic ring[7], which clearly contains the required subgroup, and $\mathcal{R}_2$ be some linearly disjoint ring with basis $\mathbf{B} = (\mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_\rho)$. Then we consider the tensor ring $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2$. Intuitively for $r \leq \rho$, we can pack $\boldsymbol{x} = (x_1, \ldots, x_r) \in \mathcal{R}_1^r$ as $x = \sum x_i \mathsf{v}_i$, i.e., each element is put on one basis, and similarly, $\boldsymbol{y} = (y_1, \ldots, y_r) \in \mathcal{R}_1^r$ to $y = \sum y_i \mathsf{v}_i$. Clearly, $x+y$ corresponds $\boldsymbol{x}+\boldsymbol{y}$, but the ring multiplication, i.e., $x \cdot y \in \mathcal{R}_1 \otimes \mathcal{R}_2$, does not correspond to the component-wise multiplication $\boldsymbol{x} \odot \boldsymbol{y}$. This is because there are a lot of uncanceled cross terms $\mathsf{v}_i \cdot \mathsf{v}_j$ when we compute $x \cdot y$.

Attempt 2. To cancel the cross terms, we can use the *dual* basis with the trace function in the following way. Let $\mathcal{R}_2^\vee$ be the dual ring with basis $\mathbf{B}^\vee = (\mathsf{v}_1^\vee, \mathsf{v}_2^\vee, \ldots, \mathsf{v}_\rho^\vee)$, i.e., the dual basis of $\mathbf{B}$. Now we can pack $\boldsymbol{x}$ the same way, i.e., $x = \sum x_i \mathsf{v}_i$, but pack $\boldsymbol{y}$ in the dual space, i.e., $y = \sum y_i \mathsf{v}_i^\vee$. Even though $x \cdot y$ has a lot of cross terms, we notice that $\mathsf{Tr}_{\mathcal{R}/\mathbb{Q}}(x \cdot y) = \sum x_i y_i$, as $\mathsf{Tr}_{\mathcal{R}/\mathbb{Q}}(\mathsf{v}_i \mathsf{v}_j^\vee)$

[7] The cyclotomic polynomial would be of a different form if q is not a power of two. Here we use this setting for simplicity of exposition, but note that our framework works for general cyclotomic rings.

acts as the Kronecker delta δ_{ij}, which is equal to 1 if $i = j$ or otherwise 0[8]. This method does cancel the cross terms, but it also mixes up the $x_i y_i$'s like the inner product. Thus, this attempt still does not achieve the goal.

Attempt 3. To further separate $x_i y_i$'s, we propose to use a third ring $\mathcal{R}_3$ with basis $\mathbf{W} = (\mathsf{w}_1, \mathsf{w}_2, \ldots, \mathsf{w}_\tau)$ for $\tau > r$. Then we consider the tensor ring $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$. In this setting, we still pack $\boldsymbol{x}$ in the same way, i.e., $x = \sum x_i \mathsf{v}_i$, but $\boldsymbol{y}$ in the space $\mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3$ as $y = \sum y_i \mathsf{v}_i^\vee \mathsf{w}_i$. Interestingly, now we have $w' = \mathsf{Tr}_{\mathcal{R}/\mathcal{R}_1 \otimes \mathcal{R}_3}(xy) = \sum x_i y_i \mathsf{w}_i$, as the trace function (over $\mathcal{R}/\mathcal{R}_1 \otimes \mathcal{R}_3$) would act as δ_{ij} on the term $\mathsf{v}_i \mathsf{v}_j^\vee$ and as a constant function on elements in $\mathcal{R}_1 \otimes \mathcal{R}_3$. Thus, the resulting w' can be viewed as a packed plaintext of $\boldsymbol{x} \odot \boldsymbol{y}$ in $\mathcal{R}_1 \otimes \mathcal{R}_3$. Here a natural question raises – how do we proceed with the computation?

A naive way to achieve this is to introduce a fourth ring $\mathcal{R}_4$, and then pack the next vector, e.g., say $\boldsymbol{z} = (z_1, \ldots, z_r)$ as z in the space $\mathcal{R}_3^\vee \otimes \mathcal{R}_4$, and then compute $w'' = \mathsf{Tr}_{\mathcal{R}/\mathcal{R}_3 \otimes \mathcal{R}_4}(zw')$. However, this way would require to blow up the ring dimension linearly to the number of multiplications. This is clearly unsatisfactory and impractical even in theory.

Final Idea. To tackle the above drawback, we observe that the space can be reused so that the tensor product of three rings, e.g., $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$, is sufficient. Particularly, consider this example where we want to compute $\boldsymbol{x} \otimes \boldsymbol{y} \otimes \boldsymbol{z}$. We can pack $\boldsymbol{x}$ as $\sum x_i \mathsf{v}_i$ and $y = \sum y_i \mathsf{v}_i^\vee \mathsf{w}_i$. Then by using the trace computation, we obtain the intermediate result $w' = \sum x_i y_i \mathsf{w}_i$. Then we can pack $\boldsymbol{z}$ as $z = \sum z_i \mathsf{w}_i^\vee \mathsf{v}_i$, and then compute $w'' = \mathsf{Tr}_{\mathcal{R}/\mathcal{R}_1 \otimes \mathcal{R}_2}(w'z) = \sum x_i y_i z_i \mathsf{v}_i$. Now, w'' is the packed element of $\boldsymbol{x} \odot \boldsymbol{y} \odot \boldsymbol{z}$ in the space $\mathcal{R}_1 \otimes \mathcal{R}_2$. Thus, by alternating between the spaces $\mathcal{R}_1 \otimes \mathcal{R}_2$ and $\mathcal{R}_1 \otimes \mathcal{R}_3$, we can batch plaintext computation for any arbitrary number of steps, without further blowing up the ring dimension.

Instantiation. There can be various ways to instantiate the framework. The most intuitive one is to use the decomposability of cyclotomic rings. Particularly, let ξ_m be the m-th root of unity for $m = qpt$ for co-prime factors q, p, t. Then the cyclotomic ring $\mathbb{Z}[\xi_m]$ is isomorphic to the tensor of the three smaller and linearly disjoint cyclotomic rings, i.e., $\mathbb{Z}[\xi_m] \cong \mathbb{Z}[\xi_q] \otimes \mathbb{Z}[\xi_p] \otimes \mathbb{Z}[\xi_t]$. In this case, we can define sub-rings $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$ as $\mathbb{Z}[\xi_q], \mathbb{Z}[\xi_p], \mathbb{Z}[\xi_t]$, respectively. We notice that the dimension of $\mathcal{R}$ would be $N = \phi(m)$, and the dimensions of the sub-rings would be $\phi(q)$, $\phi(p)$, $\phi(t)$, respectively, where ϕ is the Euler's phi function.

Homomorphic Computation over Batch Plaintexts. Next we present how to perform homomorphic computation over ciphertexts that encrypt the batch plaintexts as above. First we observe that the RGSW supports the computation naturally if we instantiate the scheme in the cyclotomic ring $\mathbb{Z}[\xi_m]$ as above. While the prior analyses of RGSW (e.g., noise growth) focused on the case when m is power-of-two, this work shows that similar analyses would also work in the general cyclotomic rings, by using the toolkits of [26]. However, as we elaborate

[8] We abuse the notation in the subscribe by using the rings for simplicity. Precisely, this should be $\mathsf{Tr}_{K/\mathbb{Q}}$ where K is the number field for which $\mathcal{R}$ is its ring of integers.

below, a direct adoption of the analyses would hit several technical challenges, and thus cannot derive a polynomial bound for the noise growth.

Notice that RGSW can be packed the same way as packing the plaintexts. Particularly, given $\mathbf{C}_1, \ldots, \mathbf{C}_r$ that encrypt $(x_1, \ldots, x_r) \in \mathcal{R}_1^r$, $\mathbf{C}' = \sum \mathbf{C}_i \mathsf{v}_i$ is an encryption of $x = \sum x_i \mathsf{v}_i$. And similarly, we can pack the ciphertexts in the other modes, e.g., $\mathbf{C}' = \sum \mathbf{C}_i \mathsf{v}_i^{\vee} \mathsf{w}_i$. As the batch multiplication (for plaintexts) consists of a multiplication followed by a trace computation, homomorphic multiplication would involve a RGSW multiplication followed by a homomorphic trace evaluation. Thus, the task is reduced to how to homomorphically compute the trace function.

Subtle Issue 1. We notice that $\mathsf{Tr}_{\mathcal{R}/\mathcal{R}_1 \otimes \mathcal{R}_3}(x) = \sum_{\sigma \in \mathsf{Gal}(\mathcal{R}/\mathcal{R}_1 \otimes \mathcal{R}_3)}(x)$, i.e., summation over all the automorphisms in the Galois group. Furthermore, homomorphic computation of the automorphism $\sigma()$ can be achieved by the BV key-switch technique [8], roughly of the same complexity of a RGSW homomorphic multiplication. Thus, a naive application of this idea would require $\tau = |\mathsf{Gal}(\mathcal{R}/\mathcal{R}_1 \otimes \mathcal{R}_3)|$ calls to the key-switch methods, meaning roughly τ RGSW homomorphic multiplications. However, this would require complexity as much as computing the individual unpacked homomorphic multiplications separately, meaning that the batch computation does not provide any advantage. Thus, to instantiate a meaningful batch homomorphic multiplication, we must be able to compute the trace homomorphically within $o(\tau)$ RGSW homomorphic multiplications.

While this task is in general difficult, we observe that if $\mathcal{R}_2$ has a tower structure, e.g., $\mathbb{Z}[\xi_\tau]$ for $\tau = 3^d$, then we can compute the homomorphic evaluation by making $O(\log \tau)$ calls to the key-switch algorithm, which is roughly $O(\log \tau)$ RGSW multiplications. Thus under this structure, to compute $\boldsymbol{x} \odot \boldsymbol{y}$ of size $r \lesssim \tau$, we need roughly $O(\log \tau)$ RGSW multiplications to compute in batch, which is significantly better than computing separately (which would require r RGSW multiplications). We notice that this more efficient trace (homomorphic) computation has been explored for cyclotomic rings of two's power [3, 11], and this work further extends the result to general cyclotomic rings.

Subtle Issue 2. Another perhaps even more subtle issue is the noise analysis. Consider messages (plaintexts) $\boldsymbol{x} := (x_1, \ldots, x_r)$ and $\boldsymbol{y} := (y_1, \ldots, y_r)$ are packed into $x, y \in \mathcal{R}$, and $\mathbf{C}_x = \mathsf{Enc}(x), \mathbf{C}_y = \mathsf{Enc}(y)$ (in the RGSW form). Then by the asymmetry noise growth property [4], the error of $\mathbf{C}_x \boxdot \mathbf{C}_y$ (where $\boxdot$ denotes RGSW homomorphic multiplication) would roughly be $e_x \sqrt{N} + x e_y$, where N is the dimension of the ring $\mathcal{R}$, e_x and e_y are the noise terms inside $\mathbf{C}_x$ and $\mathbf{C}_y$, respectively. Then the homomorphic trace evaluation would incur a blowup of some multiplicative factor W, which is some fixed polynomial. Thus, the overall noise behavior would be roughly $e_x \mathsf{poly}(\lambda) + xW e_y$.

However, to bootstrap within a polynomial modulus, the AP14/FHEW framework crucially requires that $\|xW\| \leq 1$. In the case of unpacked bit computation, this is true as $x \in \{0, 1\}$ and there is no need to do trace evaluation (so W can be thought as 1). However, in the packed computation of our framework, $\|xW\|$ is inherently greater to 1. Thus, a direct analysis would result in a super-polynomial blow up on noise, implying a super-polynomial modulus.

Even though the general analysis does not work, for our particular batch framework computation, we do identify a beautiful noise characterization if we alternate the multiplications between $\mathcal{R}_1 \otimes \mathcal{R}_2$ and $\mathcal{R}_1 \otimes \mathcal{R}_3$, under a careful algebraic analysis. Below we describe the core insight at a high level and refer curious readers to the proof of Theorem 6.2 for the details.

As we discussed, the W term comes from the trace evaluation, i.e., $\mathsf{Tr}(\cdot)$. If we perform homomorphic multiplications on multiple ciphertexts, then the following term will appear in the noise – $\mathsf{Tr}(x_1 \cdot \mathsf{Tr}(x_2 \cdot \mathsf{Tr}(\ldots \mathsf{Tr}(x_k e))$, where e is some fresh error, and x_i's are the packed messages. This term will approach W^k for the general case, resulting in an exponential blowup in the noise. However, under the following two conditions: (1) we alternate the trace functions between $\mathcal{R}_1 \otimes \mathcal{R}_2$ and $\mathcal{R}_1 \otimes \mathcal{R}_3$; (2) each x_i is a packing of $(x_{i1}, \ldots, x_{ir})$ where $\|x_{ij}\| = 1$, e.g., a power of some root of unity ξ_q^z, then we can derive a simple and small polynomial upper bound for this term.

This insight can be used to prove that the batch AP14/FHEW bootstrap algorithm only incurs a polynomial error growth under our framework.

Overall – How Many Slots can the Framework Batch. Now we determine how many slots our framework can batch when applying to the AP14/FHEW framework. We can set the tensor ring $\mathbb{Z}[\xi_q] \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$ to perform batch computation of the explicit framework of AP14/FHEW. Let N denote the dimension of the ring $\mathcal{R}$. By setting $n = O(\lambda), q \approx \tilde{O}(\sqrt{n})$ and $\mathcal{R}_2$ roughly of a similar dimension to $\mathcal{R}_3$, we can batch $r = O(\sqrt{N/q})$ slots. Asymptotically, we can set $N = O(n)$, resulting in $r = O(\lambda^{1/4-o(1)})$.

Thus, we can bootstrap λ LWE ciphertexts, using $O(\lambda/r) = \tilde{O}(\lambda^{1.75})$ FHE multiplications. This proves Theorem 1.1.

Comparison with a Recent Progress. Recently, the work [16] considered batch homomorphic computation, yet only achieved a weaker version of the task. Briefly speaking, [16] is able to bootstrap one LWE input ciphertext, e.g., $\mathsf{Enc}(\mu)$ to ℓ LWE ciphertexts $\mathsf{Enc}(f_1(\mu)), \mathsf{Enc}(f_2(\mu)), \ldots, \mathsf{Enc}(f_\ell(\mu))$, for ℓ different functions. However the message space of μ is small, e.g., a bit or $\mathbb{Z}_t$ for some small t, so one cannot squeeze a long string $x \in \{0,1\}^n$ into μ. Moreover, their method does not support batch computing on multiple LWE inputs, whereas our framework does. Thus, our framework has non-trivial advantages.

2 Preliminary

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 x_i to denote the i-th element of $\boldsymbol{x}$. We use $\|\boldsymbol{x}\|_2$ denotes the l_2-norm, i.e., $\|\boldsymbol{x}\|_2 = \sqrt{\sum_i \|x_i\|^2}$ and $\|\boldsymbol{x}\|_\infty$ denotes the l_∞-norm of $\boldsymbol{x}$, i.e., $\|\boldsymbol{x}\| = \max_i \{\|x_i\|\}$. We use bold capital letters to denote matrices. For a matrix $\mathbf{X}$, $\boldsymbol{x}_i$ denotes its i-th column vector without extra instructions, $\mathbf{X}^\top$ denotes the transpose of $\mathbf{X}$, $\|\mathbf{X}\|_2 := \max_i\{\|\boldsymbol{x}_i\|_2\}$, $\|\mathbf{X}\|_\infty := \max_i\{\|\boldsymbol{x}_i\|_\infty\}$.

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 $\mathbf{Y}$.

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 $\mathcal{P}$.

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 [4,17], 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 to the full version of this work, and note that more details can be found in the work [26].

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

Lemma 2.2 *[26] 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 [26], 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 present the details in the full version of this work.

Trace. We notice that the cyclotomic field $K = \mathbb{Q}(\xi_m)$ is a Galois extension over $\mathbb{Q}$, and thus the homomorphisms $\{\sigma_i\}$ are automorphisms that form the Galois group, denoted by $\mathsf{Gal}(K/\mathbb{Q})$. The trace $\mathsf{Tr} = \mathsf{Tr}_{K/\mathbb{Q}} : K \to \mathbb{Q}$ of an element $a \in K$ can be defined as the sum of the embeddings: $\mathsf{Tr}(a) = \sum_i \sigma_i(a) = \sum_{\sigma_i \in \mathsf{Gal}(K/\mathbb{Q})} \sigma_i(a)$. Clearly, the trace is $\mathbb{Q}$-linear, and also notice that $\mathsf{Tr}(a \cdot b) = \langle \sigma(a), \overline{\sigma(b)} \rangle$, so $\mathsf{Tr}(a \cdot b)$ is a symmetric bilinear form akin to the inner product of the embeddings of a and b.

By the Galois theory, there is a bijection between the set of subfields E of K containing $\mathbb{Q}$ and the set of subgroups G of $\mathsf{Gal}(K/\mathbb{Q})$. Thus, for any intermediate subfield E, the Galois group of $\mathsf{Gal}(K/E)$ is also well-defined. Furthermore, we can define the trace function for the intermediate subfields as: $\mathsf{Tr}_{K/E}(a) = \sum_{\sigma \in \mathsf{Gal}(K/E)} \sigma(a)$ for $a \in K$ and $\mathsf{Tr}_{E/\mathbb{Q}}(b) = \sum_{\sigma \in \mathsf{Gal}(E/\mathbb{Q})} \sigma(b)$ for $b \in E$. The trace functions behave well in towers, i.e. for $a \in K$,

$$\mathsf{Tr}_{K/\mathbb{Q}}(a) = \mathsf{Tr}_{E/\mathbb{Q}}(\mathsf{Tr}_{K/E}(a)).$$

Ring of Integers and Ideals. An algebraic integer is an algebraic number whose minimal polynomial over the rationals has integer coefficients. For a number field K, denote its subset of algebraic integers by $\mathcal{O}_K$, which forms a ring, called the ring of integers of K. The norm of any algebraic integer is in $\mathbb{Z}$.

An (integer) ideal $\mathcal{I} \subset \mathcal{O}_K$ is an additive subgroup and for any $x \in K$, $x\mathcal{I} \subset \mathcal{I}$. Every ideal in $\mathcal{O}_K$ is a set of all $\mathbb{Z}$-linear combinations of some basis.

The sum of two ideals $\mathcal{I}, \mathcal{J}$ is the set of all $x + y$ for $x \in \mathcal{I}$, $y \in \mathcal{J}$, and the product ideal $\mathcal{I}\mathcal{J}$ is the ideal generated by terms of xy. A fractional ideal $\mathcal{I} \subset K$ is a set such that $d\mathcal{I} \subset \mathcal{O}_K$ is an integral ideal for some $d \in \mathcal{O}_K$. A fractional ideal $\mathcal{I}$ is invertible if there exists a fractional ideal $\mathcal{J}$ such that $\mathcal{O}_K = \mathcal{I} \cdot \mathcal{J}$, which is unique and denoted as $\mathcal{I}^{-1}$.

Duality. For any lattice $\mathcal{L} \subset K$(i.e. the $\mathbb{Z}$-span of any $\mathbb{Q}$-basis of K), its dual is defined as $\mathcal{L}^\vee = \{x \in K | \mathsf{Tr}(x\mathcal{L}) \subset \mathbb{Z}\}$. Then $\mathcal{L}^\vee$ embeds as the complex conjugate of the dual lattice, which means $\sigma(\mathcal{L}^\vee) = \overline{\sigma(\mathcal{L})^*}$ due to the fact that $\mathsf{Tr}(xy) = \sum_i \sigma_i(x)\sigma_i(y) = \langle \sigma(x), \overline{\sigma(y)} \rangle$. It is easy to check that $\mathcal{L} = (\mathcal{L}^\vee)^\vee$, and that if $\mathcal{L}$ is a fractional ideal, so is the $\mathcal{L}^\vee$.

For any $\mathbb{Q}$-basis $\mathbf{B} = \{\mathsf{b}_j\}$ of K, we denote its dual basis by $\mathbf{B}^\vee = \{\mathsf{b}_j^\vee\}$, which is characterized by $\mathsf{Tr}(\mathsf{b}_i \cdot \mathsf{b}_j^\vee) = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$. It is immediate that $(\mathbf{B}^\vee)^\vee = \mathbf{B}$, and if $\mathbf{B}$ is a $\mathbb{Z}$-basis of some fractional ideal $\mathcal{I}$, then $\mathbf{B}^\vee$ is a $\mathbb{Z}$-basis of its dual ideal $\mathcal{I}^\vee$. If $a = \sum a_j \cdot \mathsf{b}_j$ for $a_j \in \mathbb{R}$ is the unique presentation of $a \in K_\mathbb{R}$ in basis $\mathbf{B}$, then $a_j = \mathsf{Tr}(a\mathsf{b}_j^\vee)$. For a fixed $\mathbb{Z}_q$-basis $\{\mathsf{b}_1, ..., \mathsf{b}_n\}$ of $\mathcal{O}_K/q\mathcal{O}_K$, the randomized algorithm $\boldsymbol{g}^{-1}(\cdot)$ can be extended to the subring of K modulo q, $\mathcal{O}_K/q\mathcal{O}_K$.

Lemma 2.3 *For a given integer q, let $\ell = \lceil \log q \rceil$, $\boldsymbol{g}^\top = (1, 2, .., 2^{\ell-1})$ and a fixed $\mathbb{Z}_q$-basis $\{\mathsf{b}_1, ..., \mathsf{b}_n\}$ of $\mathcal{O}_K/q\mathcal{O}_K$, then there is a randomized, efficiently computable function $\boldsymbol{g}^{-1} : \mathcal{O}_K/q\mathcal{O}_K \to \mathcal{O}_K^\ell$, such that the output of the function, $\boldsymbol{x} \leftarrow \boldsymbol{g}^{-1}(a)$, always satisfies $\langle \boldsymbol{g}, \boldsymbol{x} \rangle = a \mod q$.*

In details, if $a = a_1\mathsf{b}_1 + ... + a_n\mathsf{b}_n$ where $a_i \in \mathbb{Z}_q$ and $\boldsymbol{x}_i \leftarrow \boldsymbol{g}^{-1}(a_i)$ where the function $\boldsymbol{g}^{-1}(\cdot)$ is defined in Lemma 2.1, then $\boldsymbol{x} = \boldsymbol{x}_1\mathsf{b}_1 + ... + \boldsymbol{x}_n\mathsf{b}_n$ and each vector $\boldsymbol{x}_i \in \mathbb{Z}_q^\ell$ is sub-gaussian with parameter $O(1)$.

2.3 Learning with Errors Assumption

The learning with errors (LWE) problem was introduced by Regev [34], which is as hard as several worst-case lattice problems. For the definition of LWE, we need the following distribution $A_{s,\chi}$. If χ is a distribution over $\mathbb{Z}$ and $\boldsymbol{s} \in \mathbb{Z}_q^n$, a sample from the distribution $A_{\boldsymbol{s},\chi}$ is of the form $(b, \boldsymbol{a}) \in \mathbb{Z}_q \times \mathbb{Z}_q^n$ with $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e \mod q$, where $\boldsymbol{a}$ is chosen from $\mathbb{Z}_q^n$ uniformly and e is chosen from the distribution χ. Now we propose the problem formally in the following definitions.

Definition 2.4 (LWE) Let χ be a distribution over $\mathbb{Z}$, an integer modulus $q \geq 2$. The decision version of LWE, denoted as $\mathsf{LWE}_{n,q,\chi}$, is given m pairs of $(b', \boldsymbol{a}') \in \mathbb{Z}_q \times \mathbb{Z}_q^n$ and decide these pairs are from the uniform distribution or $A_{\boldsymbol{s},\chi}$.

The ring variant of LWE is the foundation of this work. In the rest of the paper, the special ring $\mathcal{R} = \mathcal{O}_K$ is used by default. We present the definition of RLWE by defining the distribution $A_{s,\chi}$ as follow. Let χ be a distribution over $K_\mathbb{R}$ and $\boldsymbol{s} \in \mathcal{R}_q$, and then a sample from $A_{s,\chi}$ is of the form

$$(b = s \cdot a + e \mod q\mathcal{R}, a) \in K_\mathbb{R}/q \times \mathcal{R}_q,$$

where $a \leftarrow \mathcal{R}_q$ and $e \leftarrow \chi$. Then, the definition of RLWE is presented as follows.

Definition 2.5 (RLWE) For security parameter λ, let $n = n(\lambda)$ be the dimension, $q = q(\lambda) \geq 2$ be an integer modulus, and $\chi = \chi(\lambda)$ be a distribution over $K_\mathbb{R}$. The task of decision $\mathsf{RLWE}_{n,q,\chi}$ is, given m pairs of $(b', a') \in K_\mathbb{R}/q \times \mathcal{R}_q$, decide whether the pairs are from the uniform distribution or $A_{s,\chi}$.

There is strong evidence showing hardness of LWE e.g., [7, 34] and RLWE, e.g., [25,26,33]. These problems have been extensively studied in the NIST's post-quantum standardization process in recent years. Particularly, many plausible candidates are LWE or RLWE-based designs.

Remark 2.6 *In this work, we present the primal version of the* RLWE *where the secret $\boldsymbol{s}$ lies in the primal ring $\mathcal{R}_q$, where the original* RLWE *[25] is defined in the dual. Nevertheless, the dual and primal versions are equivalent up to a tweak factor [31], and thus the primal variant is also plausibly as hard. For simplicity of presentation, this work uses secrets in the primal ring by default.*

3 RGSW in General Cyclotomic Rings

In this section, we first revisit the basic RLWE encryption scheme and the FHE scheme GSW [4, 20] in the ring settings, denoted as RGSW. The RGSW has been analyzed in power-of-two cyclotomic rings, e.g., [13,17], yet the prior analyses on noise growth crucially relies on the cyclotomic polynomial $\Phi_m(X) = X^{m/2} + 1$, due to some nice properties of the coefficient embedding in such type of rings. However, this work requires to work with RGSW in general cyclotomic rings, where the prior analyses do not carry over directly. To handle this, we re-analyze the noise growth of RGSW by using the techniques of canonical embedding as described in [26], showing that RGSW in general cyclotomic rings behaves basically the same as that in the power-of-two setting.

Important Note. Throughout this work, we use the algebraic representation for ring elements for better mathematical insights.

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 $\ell = \lceil \log Q \rceil$.

3.1 RLWE Scheme

We start from 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.

3.2 RGSW Scheme

Now we present the RGSW scheme, which is basically the same as the work [4] by moving the algebraic structure to the setting of general cyclotomic rings. We notice that it suffices to develop our further results by using the symmetric-key version of RGSW, and thus we just present this version for convenience. The public-key version works analogously.

We denote the fixed gadget vector as $\boldsymbol{g}^\top = (1, 2, ..., 2^{\ell-1})$, and the gadget matrix is defined as $\mathbf{G} = \boldsymbol{g}^\top \otimes \mathbf{I}_2$. As demonstrated by [4, 27], the gadget vector/matrix play a vital role in the homomorphic computation methods. Similar to the RLWE scheme, we present the primal version of RGSW.

- **KeyGen**(1^λ): Choose randomly $s \leftarrow \mathcal{R}_Q$ and set $\mathsf{sk} := (1, -s)^\top \in \mathcal{R}_Q^2$.
- **Enc**$(\mathsf{sk}, \mu \in \mathcal{R})$: 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 $\mathbf{C} := \begin{pmatrix} s\boldsymbol{a}^\top + \boldsymbol{e}^\top \\ \boldsymbol{a}^\top \end{pmatrix} + \mu\mathbf{G} \in \mathcal{R}_Q^{2\times 2\ell}$.
- **Dec**$(\mathbf{C}, \mathsf{sk})$: The algorithm outputs an element μ in $\mathcal{R}/2\mathcal{R}$ as follow[9]:

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

  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 [4], 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 [4,20]. Therefore, this work focuses on the noise analyses, which are not trivial when porting the results to general cyclotomic rings.

Definition 3.1 *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.*

[9] We notice that $\mathcal{R}_2$ is used to denote a second ring in our framework. To avoid notation overloading, we use $\mathcal{R}/2\mathcal{R}$ to denote $\mathcal{R}$ modulo 2.

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

Below we present a lemma that summarizes the error behavior of the homomorphic operations. The error behavior in the general cyclotomic rings is similar to that in the case of power-of-two as in the prior work [13, 17], yet requires a more refined analysis. Due to the space limit, we describe the statement and defer the proof to the full version of this work.

Lemma 3.2 *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* $\max_{i\in[n]}\{\|\sigma(\mathsf{b}_i)\|_\infty\} \leq 1$ *as Lemma 2.3. Then the following facts hold.*

- *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* $r \leq \sqrt{N \cdot \log Q} \cdot \|\sigma(\boldsymbol{e}_1)\|_\infty$.

4 New Batch Homomorphic Methods via Tensor Rings

We present our framework for batch (or SIMD) homomorphic computation by using the tensor of linearly disjoint fields (and their rings of integers). Our framework is naturally compatible with the AP14/FHEW/TFHE bootstrapping methods, resulting in more efficient batch bootstrapping mechanisms. We present our new framework for batch plaintext computation, and then show how to perform homomorphically with the framework. In this section, we present in a more abstract and algebraic way, and in Section 5 we show instantiations.

4.1 Framework of Batch Plaintext Computation

We first present some math background and then our new framework.

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 $\mathcal{R} \equiv \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$, $\mathcal{R}_{12} \equiv \mathcal{R}_1 \otimes \mathcal{R}_2$, and $\mathcal{R}_{13} \equiv \mathcal{R}_1 \otimes \mathcal{R}_3$.
- 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

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

Construction. Now we present our plaintext encoding/computation methods.

– **Plaintext Packing.** The algorithm takes input $(x_1, \ldots, x_r) \in \mathcal{R}_1^r$, and an index to one of the four modes: (1) "$\mathcal{R}_{12}$", (2) "$\mathcal{R}_{13}$", (3) "$\mathcal{R}_{12} \to \mathcal{R}_{13}$", and (4) "$\mathcal{R}_{13} \to \mathcal{R}_{12}$", and outputs an encoding of the input. The packing algorithm does one of the following, selected by the mode.
  - Mode "$\mathcal{R}_{12}$": output $\sum_{i=1}^{r} x_i \cdot \mathsf{v}_i \in \mathcal{R}_{12}$.
  - Mode "$\mathcal{R}_{13}$": output $\sum_{i=1}^{r} x_i \cdot \mathsf{w}_i \in \mathcal{R}_{13}$.
  - Mode "$\mathcal{R}_{12} \to \mathcal{R}_{13}$": output $\sum_{i=1}^{r} x_i \cdot \mathsf{v}_i^{\vee}\mathsf{w}_i \in \mathcal{R}_1 \otimes \mathcal{R}_2^{\vee} \otimes \mathcal{R}_3$.
  - Mode "$\mathcal{R}_{13} \to \mathcal{R}_{12}$": output $\sum_{i=1}^{r} x_i \cdot \mathsf{v}_i\mathsf{w}_i^{\vee} \in \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3^{\vee}$.

  We assume that the packing algorithm will attach an index to its mode.
– **Addition.** The algorithm takes as input two encodings, namely (x, mode_1), (y, mode_2), outputs $(x + y, \mathsf{mode}_1)$ if $\mathsf{mode}_1 = \mathsf{mode}_2$, otherwise $\perp$.
– **Multiplication.** The algorithm takes input two encodings, namely (x, mode_1) and (y, mode_2), and does one of the following, selected by the modes.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{12}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{12} \to \mathcal{R}_{13}$": output $\mathsf{Tr}_{K/K_{13}}(xy) \in \mathcal{R}_{13}$.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{13}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{13} \to \mathcal{R}_{12}$": output $\mathsf{Tr}_{K/K_{12}}(xy) \in \mathcal{R}_{12}$.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{12} \to \mathcal{R}_{13}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{12}$": output $\mathsf{Tr}_{K/K_{13}}(xy) \in \mathcal{R}_{13}$.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{13} \to \mathcal{R}_{12}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{13}$": output $\mathsf{Tr}_{K/K_{12}}(xy) \in \mathcal{R}_{12}$.
  - Otherwise, output $\perp$.

Correctness of these operations can be easily checked as we summarize in the following theorems. We present the proof in the full version of this work.

Theorem 4.1 (Correctness of Addition) *For any $\boldsymbol{x} = (x_1, \ldots, x_r) \in \mathcal{R}_1^r$ and $\boldsymbol{y} = (y_1, \ldots, y_r) \in \mathcal{R}_1^r$, let x, y be encodings of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively of the same mode under the plaintext packing. Then $x + y$ is an encoding of $\boldsymbol{x} + \boldsymbol{y}$ of the same mode under the plaintext packing.*

Theorem 4.2 (Correctness of Multiplication) *For any $\boldsymbol{x} = (x_1, \ldots, x_r) \in \mathcal{R}_1^r$ and $\boldsymbol{y} = (y_1, \ldots, y_r) \in \mathcal{R}_1^r$, let x, y be encodings of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively of modes "$\mathcal{R}_{1b}$" and "$\mathcal{R}_{1b} \to \mathcal{R}_{1f(b)}$" under the plaintext packing for $b \in \{2, 3\}$ and mapping $f(2) = 3$, $f(3) = 2$, and let c be the output of the multiplication algorithm on inputs x, y. Then c is an encoding of $(x_1y_1, x_2y_2, \ldots, x_ry_r) \in \mathcal{R}_1^r$, with the mode "$\mathcal{R}_{1f(b)}$" under the plaintext encoding.*

4.2 Homomorphic Encoding and Computation

We now present how to homomorphically perform the batch plaintext computation in the prior section. Here we assume two homomorphic evaluation algorithms, $\mathsf{Eval\text{-}Tr}_{K/K_{12}}(\cdot)$ and $\mathsf{Eval\text{-}Tr}_{K/K_{13}}(\cdot)$, as black-boxes, and will instantiate these algorithms in the next section (i.e., Section 4.3). We first describe the syntax of these two algorithms and some other necessary backgrounds.

Homomorphic Eval of Trace. Let $\mathsf{Eval\text{-}Tr}_{K/K_{12}}(\cdot)$ be a homomorphic evaluation algorithm that takes input either a RGSW ciphertext $\mathbf{C} \in \mathsf{RGSW}_s(\mu)$ or a RLWE ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_s(\mu)$, and outputs a RGSW ciphertext $\mathbf{C}' \in \mathsf{RGSW}_s(\mathsf{Tr}_{K/K_{12}}(\mu))$, or respectively a RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_s(\mathsf{Tr}_{K/K_{12}}(\mu))$. Importantly, here each entry of the input ciphertext, e.g., $\mathbf{C}$ or $\boldsymbol{c}$, and the message μ may be in a slightly larger (tensor) ring, i.e., $(\mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3)$. The output $\mathbf{C}'$ or $\boldsymbol{c}'$, and the underlying message go back to the original ring $(\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3)$.

The syntax of $\mathsf{Eval\text{-}Tr}_{K/K_{13}}(\cdot)$ can be defined analogously, so here we omit the statement to avoid repetition.

$\mathbf{G^{-1}}$ for the dual spaces. Our homomorphic computation uses $\mathbf{G}^{-1}(\mathbf{C})$ for $\mathbf{C} \in (\mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3)^{2\times 2\ell}$ or $(\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3^\vee)^{2\times 2\ell}$ when computing the homomorphic multiplications, and analogously uses $\boldsymbol{g}^{-1}(\boldsymbol{c})$ for $\boldsymbol{c} \in (\mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3)^2$ or $\in (\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3^\vee)^2$ when computing the homomorphic external products. We recall that in the ring/module settings, the function $\mathbf{G}^{-1}$ or $\boldsymbol{g}^{-1}$ is defined with respect to some $\mathbb{Z}$-basis, i.e., express the ring element as integer coefficients with respect to the basis, and then do some (randomized) bit-decomposition. (Ref. Lemma 2.3).

Now we present the homomorphic computation methods corresponding to the plaintext packing/computation in Section 4.1.

- $\mathsf{RGSW\text{-}Pack}$. The algorithm takes input r RGSW ciphertexts $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_r \in \mathcal{R}^{2\times 2\ell}$, where each $\mathbf{C}_i \in \mathsf{RGSW}_s(\mu_i)$ for $\mu_i \in \mathcal{R}_1$, and an index to one of the four modes: (1) "$\mathcal{R}_{12}$", (2) "$\mathcal{R}_{13}$", (3) "$\mathcal{R}_{12} \to \mathcal{R}_{13}$", and (4) "$\mathcal{R}_{13} \to \mathcal{R}_{12}$". The algorithm outputs a packed RGSW ciphertext, by doing one of the following four according to the mode.
  - Mode "$\mathcal{R}_{12}$": output $\sum_{i=1}^r \mathbf{C}_i \cdot \mathsf{v}_i \in \mathcal{R}^{2\times 2\ell}$.
  - Mode "$\mathcal{R}_{13}$": output $\sum_{i=1}^r \mathbf{C}_i \cdot \mathsf{w}_i \in \mathcal{R}^{2\times 2\ell}$.
  - Mode "$\mathcal{R}_{12} \to \mathcal{R}_{13}$": output $\sum_{i=1}^r \mathbf{C}_i \cdot \mathsf{v}_i^\vee \mathsf{w}_i \in (\mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3)^{2\times 2\ell}$.
  - Mode "$\mathcal{R}_{13} \to \mathcal{R}_{12}$": output $\sum_{i=1}^r \mathbf{C}_i \cdot \mathsf{v}_i \mathsf{w}_i^\vee \in (\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3^\vee)^{2\times 2\ell}$.

  The packing algorithm attaches the index of its mode in the clear.
- $\mathsf{RLWE\text{-}Pack}$. The algorithm takes input r RLWE ciphertexts $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_r$, where each $\boldsymbol{c}_i \in \mathsf{RLWE}_s(\mu_i)$ for $\mu_i \in \mathcal{R}_1$, and an index to one of the four modes the same as RGSW-packing. The algorithm outputs an encoding of the RLWE ciphertexts.
  - Mode "$\mathcal{R}_{12}$": output $\sum_{i=1}^r \boldsymbol{c}_i \cdot \mathsf{v}_i \in \mathcal{R}^2$.
  - Mode "$\mathcal{R}_{13}$": output $\sum_{i=1}^r \boldsymbol{c}_i \cdot \mathsf{w}_i \in \mathcal{R}^2$.
  - Mode "$\mathcal{R}_{12} \to \mathcal{R}_{13}$": output $\sum_{i=1}^r \boldsymbol{c}_i \cdot \mathsf{v}_i^\vee \mathsf{w}_i \in (\mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3)^2$.
  - Mode "$\mathcal{R}_{13} \to \mathcal{R}_{12}$": output $\sum_{i=1}^r \boldsymbol{c}_i \cdot \mathsf{v}_i \mathsf{w}_i^\vee \in (\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3^\vee)^2$.

  We assume that the mode is included in the clear.
- Add, **(Addition for RGSW-encodings).** The algorithm takes as input two RGSW-encodings, namely $(\mathbf{C}_1, \mathsf{mode}_1), (\mathbf{C}_2, \mathsf{mode}_2)$, outputs $(\mathbf{C}_1 + \mathbf{C}_2, \mathsf{mode}_1)$ if $\mathsf{mode}_1 = \mathsf{mode}_2$, otherwise $\perp$.
- Add[10], **(Addition for RLWE-encodings).** The algorithm takes as input two RLWE-encodings, namely $(\boldsymbol{c}_1, \mathsf{mode}_1), (\boldsymbol{c}_2, \mathsf{mode}_2)$, outputs $(\boldsymbol{c}_1 + \boldsymbol{c}_2, \mathsf{mode}_1)$ if $\mathsf{mode}_1 = \mathsf{mode}_2$, otherwise $\perp$.

[10] Here we use the same function name as the above, where the input type specifies which function the call refers to.

– Mult, **(Homomorphic Product for** RGSW**-**RGSW**).** The algorithm takes input two (packed) RGSW ciphertexts, namely $(\mathbf{C}_1, \mathsf{mode}_1)$ and $(\mathbf{C}_2, \mathsf{mode}_2)$, and then computes $\mathbf{C} = \mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$. Then it outputs as:
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{12}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{12} \to \mathcal{R}_{13}$": output $(\mathsf{Eval\text{-}Tr}_{K/K_{13}}(\mathbf{C}), \mathcal{R}_{13})$.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{13}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{13} \to \mathcal{R}_{12}$": output $(\mathsf{Eval\text{-}Tr}_{K/K_{12}}(\mathbf{C}), \mathcal{R}_{12})$.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{12} \to \mathcal{R}_{13}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{12}$": output $(\mathsf{Eval\text{-}Tr}_{K/K_{13}}(\mathbf{C}), \mathcal{R}_{13})$.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{13} \to \mathcal{R}_{12}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{13}$": output $(\mathsf{Eval\text{-}Tr}_{K/K_{12}}(\mathbf{C}), \mathcal{R}_{12})$.
  - Otherwise, output $\perp$.

– $\mathsf{Ext\text{-}Prod}$, **(External Product for** RGSW**-**RLWE**).** The algorithm takes inputs a RGSW encoding $(\mathbf{C}_1, \mathsf{mode}_1)$ and a RLWE encoding $(\boldsymbol{c}_2, \mathsf{mode}_2)$, and then computes $\boldsymbol{c} = \mathbf{C}_1 \boxtimes \boldsymbol{c}_2 = \mathbf{C}_1 \cdot \boldsymbol{g}^{-1}(\boldsymbol{c}_2)$. Then it outputs according as:
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{12}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{12} \to \mathcal{R}_{13}$": output $\mathsf{Eval\text{-}Tr}_{K/K_{13}}(\boldsymbol{c})$.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{13}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{13} \to \mathcal{R}_{12}$": output $\mathsf{Eval\text{-}Tr}_{K/K_{12}}(\boldsymbol{c})$.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{12} \to \mathcal{R}_{13}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{12}$": output $\mathsf{Eval\text{-}Tr}_{K/K_{13}}(\boldsymbol{c})$.
  - $\mathsf{mode}_1 =$ "$\mathcal{R}_{13} \to \mathcal{R}_{12}$" and $\mathsf{mode}_2 =$ "$\mathcal{R}_{13}$": output $\mathsf{Eval\text{-}Tr}_{K/K_{12}}(\boldsymbol{c})$.
  - Otherwise, output $\perp$.

The readers should keep it in mind that the above operations are in $\mathcal{R}_Q$, where the modulo Q is taken implicitly. Next we describe theorems to summarize the correctness and error growth. Detailed proofs appear in the full version.

Theorem 4.3 *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*

– $\mathsf{RGSW\text{-}Pack}(\mathbf{C}_1, \ldots, \mathbf{C}_r,$ "$\mathcal{R}_{12}$"$)$ *as* $\mathbf{D}$*,*
– $\mathsf{RGSW\text{-}Pack}(\mathbf{C}'_1, \ldots, \mathbf{C}'_r,$ "$\mathcal{R}_{12} \to \mathcal{R}_{13}$"$)$ *as* $\mathbf{D}'$*,*
– $\mathsf{Mult}(\mathbf{D}', \mathbf{D})$ *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* $\mathcal{R}_{13}$*.*

Theorem 4.4 *Let* $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r$ *be* RLWE *ciphertexts with error terms* $e_1, \ldots, e_r$*, 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*

– $\mathsf{RLWE\text{-}Pack}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r,$ "$\mathcal{R}_{12}$"$)$ *as* $\boldsymbol{d}$*,*
– $\mathsf{RGSW\text{-}Pack}(\mathbf{C}'_1, \ldots, \mathbf{C}'_r,$ "$\mathcal{R}_{12} \to \mathcal{R}_{13}$"$)$ *as* $\mathbf{D}'$*,*
– $\mathsf{Ext\text{-}Prod}(\mathbf{D}', \boldsymbol{d})$ *as* $\boldsymbol{f}$*,*
– *the encrypted messages of the packed ciphertexts* $\boldsymbol{d}$ *as* $\mu_{\boldsymbol{d}}$*,*
– *the encrypted messages of the packed ciphertexts* $\mathbf{D}'$ *as* $\mu_{\mathbf{D}'}$*.*

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

Assuming that for any $\boldsymbol{x}$ *(in the input domain),* $\mathsf{Err}(\mathsf{Eval\text{-}Tr}_{K/K_{13}}(\boldsymbol{x})) = \mathsf{Tr}_{K/K_{13}}(\mathsf{Err}(\boldsymbol{x})) + e'$ *for some* e'*, whose norm upper bound can be independent of* $\boldsymbol{x}$*, then we have* $\mathsf{Err}(\boldsymbol{f}) = \mathsf{Tr}_{K/K_{13}}\left(\sum_i \boldsymbol{e}'_i \mathsf{v}_i^{\vee} \mathsf{w}_i \boldsymbol{g}^{-1}(\boldsymbol{d}) + \mu_{\mathbf{D}'}(e_i \mathsf{v}_i)\right) + e'$*.*

We notice the above two theorems can be easily adapted to the setting of the modes $\mathcal{R}_{13}$ and $\mathcal{R}_{13} \to \mathcal{R}_{12}$. We omit the statement to avoid repetition.

4.3 Homomorphic Evaluation of the Trace Function

In this section, we present an efficient method for homomorphic evaluation of the trace function, which was used as a black-box in the homomorphic multiplication in the prior subsection. There have been efficient methods studied in the literature, e.g., [3, 11, 22], in the cyclotomic rings of powers of two, and here we generalize the prior methods to the setting of general cyclotomic rings.

General Method over RLWE Ciphertexts. Suppose E/F is an algebraic extension and the degree $[E : F] = d$, then the function $\mathsf{Tr}_{E/F}$ is the sum of d automorphisms on E that fix every element in F. These d automorphisms form a group, namely the Galois group denoted as $\mathsf{Gal}(E/F)$. Then we can express $\mathsf{Tr}_{E/F}(\cdot) = \sum_{\sigma \in \mathsf{Gal}(E/F)} \sigma(\cdot)$.

The general way to compute homomorphic evaluation of $\mathsf{Tr}_{E/F}$ is to compute homomorphic evaluation of all the σ's in the Galois group, and then sum them up. We notice that homomorphic evaluation of any automorphism σ can be achieved using the classic key-switch technique [8] as follows. We first present the syntax of the key-switch algorithm.

Key-Switch Algorithm. Let KS be the key-switch algorithm (the ring variant of [8]) that takes input a RLWE ciphertext $(b, a) \in \mathsf{RLWE}_s(\mu) \in \mathcal{R}_Q^2$ and an evaluation key $\mathsf{evk}^{(\sigma)}$ and outputs a RLWE ciphertext $(b', a') \in \mathsf{RLWE}_{\sigma(s)}(\mu) \in \mathcal{R}_Q^2$. We present the details of KS in the full version of this work.

Given the evaluation algorithm KS, homomorphic evaluation of Tr can be achieved by the following. Given input $(b, a) \in \mathsf{RLWE}_s(\mu)$ and evaluation keys $\{\mathsf{evk}^{\sigma}\}_{\sigma \in \mathsf{Gal}(E/F)}$, the algorithm does:

1. For each $\sigma \in \mathsf{Gal}(E/F)$, compute $c_\sigma = (\sigma(b), \sigma(a))$ and set $c'_\sigma = \mathsf{KS}(c_\sigma, \mathsf{evk}^{(\sigma^{-1})})$.
2. Output $\sum_{\sigma \in \mathsf{Gal}(E/F)} c'_\sigma$ as the resulting ciphertext.

It is not hard to check that for each σ, $c_\sigma \in \mathsf{RLWE}_{\sigma(s)}(\sigma(\mu))$, and by correctness of KS, $c'_\sigma \in \mathsf{RLWE}_s(\sigma(\mu))$. Thus, the above is a correct algorithm. Moreover, it requires d calls[11] to the underlying KS algorithm.

More Efficient Evaluation with Algebraic Structures. If there is an intermediate field K between E and F, then we can (homomorphically) compute the trace function more efficiently via the composition property of the trace function. Let $F \subset K \subset E$ be algebraic extensions, $[E : K] = d_1$ and $[K : F] = d_2$, then we have $d = d_1 \cdot d_2$ and $\mathsf{Tr}_{E/F} = \mathsf{Tr}_{K/F} \circ \mathsf{Tr}_{E/K}$. By definition, we have $\mathsf{Tr}_{E/K}(\cdot) = \sum_{\sigma \in \mathsf{Gal}(E/K)} \sigma(\cdot)$, and $\mathsf{Tr}_{K/F}(\cdot) = \sum_{\sigma \in \mathsf{Gal}(K/F)} \sigma(\cdot)$. To compute $\mathsf{Tr}_{E/F}(x)$, we can first compute $x' = \mathsf{Tr}_{E/K}(x)$, and then output $\mathsf{Tr}_{K/F}(x')$. The homomorphic evaluation just computes the basic trace evaluation twice of the cases E/K and K/F. In this way, the algorithm would require only $d_1 + d_2$ calls to the underlying KS. This is more efficient than the basic algorithm applied to the case E/F directly, which would require $d = d_1 \cdot d_2$ calls to the KS.

[11] For small d's, the *Hoistng* technique [22] can be used to improve efficiency.

The Tower Case. The above idea works best in the *tower* case, where there are many intermediate fields between E and F. In the rest, we present an optimized homomorphic evaluation algorithm for $\mathsf{Tr}_{K/K_{13}} : \mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3 \mapsto \mathcal{R}_1 \otimes \mathcal{R}_3$, assuming there are many intermediate fields. We discuss how to instantiate this later in Section 5. Note that a homomorphic algorithm for the other case $\mathsf{Tr}_{K/K_{12}}$ can be derived similarly.

Assume the following tower structure: $K_{13} = E_t \subset E_{t-1} \subset \cdots \subset E_1 = K$. Then we can express $\mathsf{Tr}_{K/K_{13}} = \mathsf{Tr}_{E_{t-1}/E_t} \circ \mathsf{Tr}_{E_{t-2}/E_{t-1}} \circ \cdots \circ \mathsf{Tr}_{E_1/E_2}$. We will present how to instantiate this tower structure in the cyclotomic fields in Section 5. By using the basic homomorphic evaluation on the cases E_i/E_{i+1}, we can derive a more efficient algorithm.

Before presenting formally the procedure, we notice that there is a technical subtlety – the input RLWE ciphertext is in the dual module, e.g., $(\mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3)^2$, so that we cannot directly apply the above procedure. To tackle this, we first observe that there is an integer P that is invertible under modulo Q, and can map an element in the dual module to a ring element by the multiplication, i.e., (1) $P^{-1} \mod Q$ exists, and (2) for every $x \in \mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3$, $P \cdot x \in \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$. In Section 5, we show how to set P concretely with detailed instantiations of the required tensor rings. By using this number P, we present a tweaked method in Algorithm 4.1 below.

Algorithm 4.1: (RLWE)-Eval-$\mathsf{Tr}_{K/K_{13}}$ with the tower structure

Input :
- A RLWE ciphertext $(b, a) \in (\mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3)^2$ that encrypts a message $\mu \in \mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3$ under a secret $s \in \mathcal{R}$.
- **Evaluation Key:** $\{\mathsf{evk}^{(\sigma)}\}_{\sigma \in \bigcup_{i \in [t-1]} \mathsf{Gal}(E_i/E_{i+1})}$, and $\mathsf{evk} \in \mathsf{RGSW}_s(P^{-1} \cdot s) \in \mathcal{R}^2$.

Output : A RLWE ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_s(\mathsf{Tr}_{K/K_{13}}(\mu))$.

Initialize $\boldsymbol{c} = (b, a)$, and set $\bar{a} = P \cdot a$ (interpreted as an element in $\mathcal{R}$) ;
Set $\boldsymbol{c}' = (0, \bar{a})$ and compute $\boldsymbol{d} = \mathsf{evk} \boxtimes \boldsymbol{c}'$; $/\!/ \boldsymbol{d} \in \mathsf{RLWE}_s(P^{-1} s \cdot \bar{a}) \in \mathcal{R}^2$;
for $i = 1$ **to** $t - 1$ **do**
  Let $(d_1, d_2) = \boldsymbol{d}$;
  Compute $\boldsymbol{d}' = \sum_{\sigma \in \mathsf{Gal}(E_i/E_{i+1})} \mathsf{KS}((\sigma(d_1), \sigma(d_2)), \mathsf{evk}^{(\sigma^{-1})})$;
  Set $\boldsymbol{d} = \boldsymbol{d}'$ and $\boldsymbol{d}' = (0, 0)$
Return $(\mathsf{Tr}_{K/K_{13}}(b), 0) - \boldsymbol{d}$. $/\!/ \boldsymbol{d} \in \mathsf{RLWE}_s(\mathsf{Tr}_{K/K_{13}}(sa)) \in \mathcal{R}^2$

Remark 4.5 *Here we slightly abuse the notation of the automorphism $\sigma \in \mathsf{Gal}(E_i/E_{i+1})$ for simplicity of presentation.*

As the input domain of such σ is E_i, we should not give $(d_1, d_2) \in \mathcal{R} \subset K$ (elements of the full ring) as the input to the automorphism. Nevertheless, by the Galois theorem, for any $\sigma \in \mathsf{Gal}(E_i/E_{i+1})$, there exists at least one $\sigma' \in$

$\mathsf{Gal}(K/E_{i+1})$ *such that* $\sigma'|_{E_i} = \sigma$. *In this paper,* σ *refers to (an arbitrary) of such* σ' *who acts identically as* σ *for all inputs in* E_i.

To verify correctness, we first notice that $\mathsf{Tr}_{K/K_{13}}(b) = \mathsf{Tr}_{K/K_{13}}(sa)+\mathsf{Tr}_{K/K_{13}}(e)+\mathsf{Tr}_{K/K_{13}}(\mu) \in \mathcal{R}$. Next, at the end of the for loop (line 6), we can easily check that $\boldsymbol{d} \in \mathsf{RLWE}_s(\mathsf{Tr}_{K/K_{13}}(P^{-1}s\bar{a}))$. Then we have

$$\mathsf{Tr}_{K/K_{13}}(P^{-1}s\bar{a}) = P^{-1}\mathsf{Tr}_{K/K_{13}}(s\bar{a}) = P^{-1}\mathsf{Tr}_{K/K_{13}}(sPa) = \mathsf{Tr}_{K/K_{13}}(sa).$$

Crucially the last equality holds because $\mathsf{Tr}_{K/K_{13}}(sa) \in \mathcal{R}_1 \otimes \mathcal{R}_3$, and $P^{-1} \cdot \mathsf{Tr}_{K/K_{13}}(sPa) = P^{-1} \cdot P \cdot \mathsf{Tr}_{K/K_{13}}(sa) = \mathsf{Tr}_{K/K_{13}}(sa)$ in modulo Q. Then correctness simply follows. As the modulus does not change in the whole procedure, we omit the modulo Q description in the algorithm.

To analyze efficiency, we first denote the degrees as $d_i = [E_i : E_{i+1}]$ for $i \in [t-1]$. Then the above algorithm makes $\sum_{i\in[t-1]} d_i$ calls to the underlying KS algorithm, which is again way more efficient than the basic algorithm applied to the case K/K_{13}, which would require $d = \prod_{i\in[t-1]} d_i$ calls to KS. Moreover, if each $d_i = O(1)$, then the efficient algorithm as above would require $O(\log d)$ calls to the KS, which is significantly better than d calls by the basic approach.

Below we present the noise analysis and defer the proof to the full version.

Theorem 4.6 *Adapt the notations of Algorithm 4.1. Assume that for every* $\boldsymbol{d} \in \mathsf{RLWE}_s(\mu)$, $\mathsf{Err}(\mathsf{KS}(\boldsymbol{d})) = \mathsf{Err}(\boldsymbol{d}) + e'$ *where* $\|e'\|_\infty$ *is a sub-Gaussian with parameter* B, *and for the initial* $\boldsymbol{d}$, $\|\mathsf{Err}(\boldsymbol{d})\|_\infty$ *is also a sub-Gaussian with parameter* B. *Let* $\boldsymbol{c}$ *be the output ciphertext of the algorithm. Then* $\mathsf{Err}(\boldsymbol{c}) = \mathsf{Tr}_{K/K_{13}}(e)+e''$ *where* e *is the noise of the input ciphertext, and* $\|e''\|_\infty$ *is a sub-Gaussian with parameter upper bounded by* $3dB$.

Eval-Tr for RGSW. The Eval-$\mathsf{Tr}_{K/K_{13}}$ algorithm for RLWE ciphertexts can be extended to RGSW ciphertexts. Details are in the full version.

5 Instantiations

In this section, we present how to instantiate all the components used in the abstraction in Section 4, so that the parameters can be analyzed concretely. Particularly, we need to instantiate: (1) tensor ring $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$, and (2) good bases of these rings and their duals. Then we can further determine parameters for the noise growth in Theorems 4.3 and 4.4, under the instantiations.

Tensor Fields/Rings. We notice that any cyclotomic field has some nice properties of decomposability, i.e., for $m = q\rho'\tau'$ where q, ρ', τ' are co-prime integers, then $\mathbb{Q}(\xi_m) \cong \mathbb{Q}(\xi_q) \otimes \mathbb{Q}(\xi_{\rho'}) \otimes \mathbb{Q}(\xi_{\tau'})$. Thus, we can use their rings of integers to instantiate our framework. Particularly, we set $\mathcal{R}_1 = \mathbb{Z}[\xi_q]$, $\mathcal{R}_2 = \mathbb{Z}[\xi_{\rho'}]$, and $\mathcal{R}_3 = \mathbb{Z}[\xi_{\tau'}]$. We notice that the tensor ring $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$ has dimension $N = \phi(m)$, $\mathcal{R}_2$ has dimension $\rho = \phi(\rho')$, and $\mathcal{R}_3$ has dimension $\tau = \phi(\tau')$. Moreover, we have $N = \phi(q)\rho\tau$. We notice that the hardness of RLWE scales with N [2, 25, 32].

To apply the fast trace evaluation as Section 4.3, we choose ρ' and τ' as powers of primes, i.e., $\rho' = p_1^{d_1}$ and $\tau' = p_2^{d_2}$, for some small primes p_1, p_2 of constant sizes, e.g., $3, 5$. We notice that for any element $x \in \mathcal{R}_2^\vee$, $\rho' x \in \mathcal{R}_2$, and therefore we can set $P = \rho'$ in Algorithm 4.1. Similarly, we can set $P = \tau'$ for computing (RLWE)-Eval-$\mathsf{Tr}_{K/K_{12}}$. As argued before, the homomorphic evaluation of the trace function would need $O(\log \rho')$ or $O(\log \tau')$ calls to the key-switch function (for the RLWE case). To maximize the space utility, we would set $\rho \approx \tau$. For the batch bootstrapping of the AP14/FHEW framework, we set q to be the input LWE modulus, which can be $\tilde{O}(\sqrt{n})$ where n is the LWE dimension.

Bases. We next determine concrete bases for $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$ (and their dual rings), denoted as $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$(and $\mathcal{B}_1^\vee, \mathcal{B}_2^\vee, \mathcal{B}_3^\vee$, respectively).

Particularly, we set $\mathcal{B}_i^\vee$ to be the decoding basis of the work [26] for each $i = 1, 2, 3$. As argued in [26], the primal bases $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ are defined as the conjugate of the powerful bases. These bases are "short", and thus would give tighter bounds for our analyses. Below we briefly summarize some nice properties about the decoding bases and their duals from [26].

Lemma 5.1 ([26]) *Let $z = w^e$ be some prime power, $\boldsymbol{d}$ be the decoding basis of $\mathbb{Z}^\vee[\xi_z]$, and $\boldsymbol{b}$ be the dual of $\boldsymbol{d}$. Then for any element $d \in \boldsymbol{d}$, $b \in \boldsymbol{b}$, we have $\|b\|_\infty = 1$ and $\|d\|_\infty \leq 2(w-1)/z$.*

By using our notation in Section 4.1, we denote the conjugate of the powerful basis as $\mathcal{B}_2 = \{\mathsf{v}_i\}_{i\in[\rho]}$, $\mathcal{B}_3 = \{\mathsf{w}_i\}_{i\in[\tau]}$, and the decoding bases (their dual) as $\mathcal{B}_2^\vee = \{\mathsf{v}_i^\vee\}_{i\in[\rho]}$, $\mathcal{B}_3^\vee = \{\mathsf{w}_i^\vee\}_{i\in[\tau]}$. Then by the above lemma, we have (1) $\|\mathsf{v}_i\|_\infty = 1$ and $\|\mathsf{w}_i\|_\infty = 1$, and (2) $\|\mathsf{v}_i^\vee\|_\infty \leq 2(p_1-1)/\rho'$ and $\|\mathsf{w}_i^\vee\|_\infty \leq 2(p_2-1)/\tau'$.

As we choose q, ρ', τ' to be relatively prime, we can use the individual bases to determine a basis of the tensor ring, i.e., $\mathcal{B}_i \otimes \mathcal{B}_j$ is the powerful basis of $\mathcal{R}_i \otimes \mathcal{R}_j$. Also, we set parameter $r = \min(\rho, \tau)$ as the batch parameter, i.e., the maximal number of slots our method can pack. If we set $\rho \approx \tau$, then $r \approx \sqrt{N/q}$.

Examples. We give some examples of concrete numbers to illustrate the above ideas. Let Q be the RLWE modulus, and σ be the noise parameter (in the absolute value). The RLWE hardness can be estimated by N (the ring dimension) and noise-to-modulus ratio σ/Q (also known as α) from the estimator [2].

	m-cyclotomic $= q \times \rho' \times \tau'$	Dim $N = \phi(m)$ $= \phi(q) \times \rho \times \tau$	Batch param r	Modulus Q (approx)	Noise σ	Hardness (in bit)	Input Dim n
Set 1	$251 * 2^3 * 3^2$	$250 * 4 * 6 = 6000$	4	2^{128}	3.2	129.9	500
Set 2	$211 * 3^2 * 7$	$210 * 6 * 6 = 7560$	6	2^{128}	3.2	178.4	500

Table 1. Some examples of parameters

Note: These examples demonstrate some ideas to set concrete parameters. How to optimize the concrete performance is an interesting future work. The moduli Q's here are approximated at this order. There can be other constraints, e.g., Q and m are co-prime for NTT accelerations.

Key-Switch Instantiation. Our trace evaluation algorithms (both the RLWE and RGSW settings) require to use the key-switch procedure. This can be achieved with existing techniques, e.g., [22]. Details are presented in the full version.

Particularly, by the parameters of the key-switch (in the full version) and Lemma 3.2, we can set $B = \sqrt{N \cdot \log Q} \cdot E$ in Theorem 4.6, where E is the noise bound in the key-switch keys. By using these instantiations applied to Theorem 4.4, we can achieve the following corollary for the external product:

Corollary 5.2 *Adapt the notations of Theorems 4.4. If the errors of the key-switch keys is upper bounded by E, and $\boldsymbol{g}^{-1}$ is with respect to the basis $\mathcal{B}_1 \otimes \mathcal{B}_2 \otimes \mathcal{B}_3$. Then $\|\mathsf{Err}(\boldsymbol{f})\|_\infty$ is upper bounded by*

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

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

A similar bound can be derived for the RGSW multiplication of Theorem 4.3 under the instantiation in this section. Details are in the full version.

6 Batch Bootstrapping via Our New Framework

Now we present how to batch the AP14/FHEW bootstrapping [4,13,17] within a polynomial modulus. We first present some background and notations, and then describe how to apply our new batch framework to the bootstrapping procedure.

6.1 Bootstrapping Background

Input. The general bootstrapping algorithm takes an LWE ciphertext $(b, \boldsymbol{a}) \in \mathbb{Z}_q^{1+n}$ as input, where n and q are small, i.e., $n = \widetilde{O}(\lambda)$, $q = \widetilde{O}(\sqrt{\lambda})$[12]. Also it is without loss of generality to assume the input ciphertexts are encrypted under binary secret vectors. These can be achieved without loss of generality by applying the dimension reduction, modulus switch (the randomized version), and bit-decomposition/power-of-two[13] as described in [4]. We know for any GSW, RLWE, or RGSW ciphertext, we can always publicly extract an LWE ciphertext that encrypts the same message [4]. Therefore, assuming the input to be the LWE form is without loss of generality.

The Batch Setting. Let r be the batch parameter as we instantiate in Section 5. Our bootstrapping algorithm takes input r LWE ciphertexts, i.e., $\{(b_i, \boldsymbol{a}_i)\}_{i\in[r]}$, encrypting perhaps different messages under the same secret key $\boldsymbol{s}$.

[12] In the full version of this work, we present how to achieve such a q.

[13] For any $(\boldsymbol{s}, \boldsymbol{a}) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^n$, $\langle \boldsymbol{s}, \boldsymbol{a}\rangle = \langle \boldsymbol{s}', \boldsymbol{a}'\rangle$ where $\boldsymbol{a}' \in \mathbb{Z}_q^{n\log q}$ is the power-of-two of $\boldsymbol{a}$ and $\boldsymbol{s}' \in \mathbb{Z}_q^{n\log q}$ is the bit-decomposition of $\boldsymbol{s}$. Using this insight, it is without loss of generality to just consider binary secret vectors in the bootstrapping task. Some practical optimizations, e.g., [6, 13, 17, 28] use binary or ternary LWE, so that the secret vector $\boldsymbol{s}$ is set directly to binary or ternary. In this case, there is no need to blow up the dimension of $\boldsymbol{a}$.

Output. The output of bootstrapping algorithm is a ciphertext encrypting the same as the input ciphertexts. In the batch setting, the output can be either a packed ciphertext, or r different ciphertexts, encrypting the same message vector. Let N, Q denote the dimension and modulus used by the output ciphertext.

6.2 Batch Bootstrapping

Notations. We use the instantiation in Section 5 for the batch framework and present the required parameters in our batch algorithm.

- n: the dimension of the input LWE scheme.
- q: the modulus of the input LWE scheme, set as a prime of size $\tilde{O}(\sqrt{\lambda})$.
- r: the number of slots we can pack, where $r = \min\{\rho, \tau\}$.
- $\boldsymbol{s}$: the secret key of the input LWE ciphertexts.
- $\mathcal{R}$: the underlying ring of the RGSW scheme. We use the instantiation in Section 5, i.e., the tensor ring $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$.
- $\mathcal{R}_1$: the first ring is set as $\mathbb{Q}(\xi_q)$.
- $\mathcal{R}_2$: the second ring with dimension $\rho = \phi(\rho')$, $\rho' = p_1^{d_1}$
- $\mathcal{R}_3$: the third ring with dimension $\tau = \phi(\tau')$, $\tau' = p_2^{d_2}$
- Q: the modulus of the RGSW scheme.
- s': the secret of the RGSW scheme.

Auxiliary algorithm. In Algorithm 6.1 below, we describe a batch blind-rotate (BR) algorithm, which is an SIMD version of FHEW/TFHE blind-rotate of [14, 17], under our framework.

Algorithm 6.1: Batch-BR (i.e., Batch Blind Rotate)

Input :
- A packed RLWE ciphertext ACC_0.
- (Partial) **Bootstrapping key:** $\{\mathsf{BK} \in \mathsf{RGSW}^Q_{s'}(s)\}$ where $s \in \{0, 1\}$.
- Integers $\{a_i\}_{i\in[r]}$.

Output : A packed RLWE ciphertext.

if *the mode of* ACC_0 *is* "$\mathcal{R}_{12}$" **then**
  Set $\mathsf{ACC} = \mathsf{Ext\text{-}Prod}((\mathsf{BK} \cdot (\sum_i \xi_q^{a_i} \mathsf{v}_i^\vee \mathsf{w}_i) + (\mathbf{G} - \mathsf{BK}) \cdot \sum_i \mathsf{v}_i^\vee \mathsf{w}_i, \text{"}\mathcal{R}_{12} \to \mathcal{R}_{13}\text{"}), (\mathsf{ACC}_0, \text{"}\mathcal{R}_{12}\text{"}))$
else if *the mode of* ACC_0 *is* "$\mathcal{R}_{13}$" **then**
  Set $\mathsf{ACC} = \mathsf{Ext\text{-}Prod}((\mathsf{BK} \cdot (\sum_i \xi_q^{a_i} \mathsf{v}_i \mathsf{w}_i^\vee) + (\mathbf{G} - \mathsf{BK}) \cdot \sum_i \mathsf{v}_i \mathsf{w}_i^\vee, \text{"}\mathcal{R}_{13} \to \mathcal{R}_{12}\text{"}), (\mathsf{ACC}_0, \text{"}\mathcal{R}_{13}\text{"}))$
Return: ACC

We next present a high level description of what the batch BR algorithm is computing. Suppose the input ACC_0 is a packed ciphertext that encrypts $(\xi_q^{x_1}, \ldots, \xi_q^{x_r})$ under mode "$\mathcal{R}''_{12}$. Then the result of the algorithm will produce a packed ciphertext that encrypts $(\xi_q^{x_1+a_1 s}, \ldots, \xi_q^{x_r+a_r s})$, under mode "$\mathcal{R}''_{13}$. The formal analysis is captured by the following theorem.

Theorem 6.1 *Adapt the notations in Algorithm 6.1. Let the input* ACC_0 *be a packed* RLWE *encrypting* μ *of mode*"$\mathcal{R}''_{1b}$ *for* $b \in \{2, 3\}$. *If* $b = 2$, *then the output is a packed* RLWE *ciphertext encrypting* $\mathsf{Tr}_{K/K_{13}}(\mu \cdot \sum_{i\in[r]} \xi_q^{a_i s} \mathsf{v}_i^\vee \mathsf{w}_i)$ *of mode "*$\mathcal{R}_{13}$*", or* $\mathsf{Tr}_{K/K_{12}}(\mu \cdot \sum_{i\in[r]} \xi_q^{a_i s} \mathsf{v}_i \mathsf{w}_i^\vee)$ *of mode "*$\mathcal{R}_{12}$*" if* $b = 3$.

Proof. By symmetry, it suffices to prove the case $b = 2$, and the other case follows analogously. Since $s \in \{0, 1\}$, we have $\mathsf{BK} \cdot (\sum_i \xi_q^{a_i} \mathsf{v}_i^\vee \mathsf{w}_i) + (\mathbf{G} - \mathsf{BK}) \cdot \sum_i \mathsf{v}_i^\vee \mathsf{w}_i \in \mathsf{RGSW}(\sum_{i\in[r]} \xi_q^{a_i s} \mathsf{v}_i^\vee \mathsf{w}_i)$. Then by the Theorem 4.4, ACC belongs to $\mathsf{RLWE}(\mathsf{Tr}_{K/K_{13}}(\mu \cdot \sum_{i\in[r]} \xi_q^{a_i s} \mathsf{v}_i^\vee \mathsf{w}_i))$.

Batch Bootstrapping. In Algorithm 6.2, we present our final batch bootstrapping algorithm, using the batch BR (Algorithm 6.1) as a subroutine. To analyze the concrete bounds, we use the instantiation in Section 5. We recall some basic facts: let $\{\mathsf{v}_i\}_{i\in[\rho]}$, $\{\mathsf{w}_i\}_{i\in[\tau]}$ be the bases of $\mathcal{R}_2$ and $\mathcal{R}_3$. Then we have

1. $\|\mathsf{v}_i\|_\infty = 1$ and $\|\mathsf{w}_i\|_\infty = 1$,
2. $\|\mathsf{v}_i^\vee\|_\infty \le 2(p_1 - 1)/\rho'$ and $\|\mathsf{w}_i^\vee\|_\infty \le 2(p_2 - 1)/\tau'$.

The analysis of Algorithm 6.2 is summarized by the theorem.

Theorem 6.2 *Let $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_r)$ be binary messages encrypted in the input* LWE *ciphertexts, $\{(b_i, \boldsymbol{a}_i)\}_{i\in[r]}$. Then the algorithm outputs a fresh packed* RLWE *ciphertext $\boldsymbol{c}$, either in $\mathsf{RLWE}^Q_{s'}(\sum \mu_i \mathsf{v}_i)$ or $\mathsf{RLWE}^Q_{s'}(\sum \mu_i \mathsf{w}_i)$.*

Moreover, $\|\mathsf{Err}(\boldsymbol{c})\|_\infty$ is bounded by a sub-Gaussian variable with parameter $O(\gamma)$ such that $\gamma \le nrqN\sqrt{N \log Q}E$, where E is the upper bound (infinity norm of the canonical embedding) of errors in all bootstrapping/evaluation keys.

Algorithm 6.2: Batch-BTS

Input :
- r LWE ciphertexts $(b_i, \boldsymbol{a}_i) = (b_i, a_{i1}, ..., a_{in}) \in \mathsf{LWE}_{\mathbf{s}}(\mu_i)$ for $i \in [r]$.
- **Bootstrapping key:** $\{\mathsf{BK}_i \in \mathsf{RGSW}^{Q'}_{s'}(s_i)\}_{i\in[n]}$, where s_i is the i-th entry of the common secret $\mathbf{s}$ of the LWE ciphertexts, and evk is the evaluation key for the homomorphic trace algorithms.

Output : A packed RLWE ciphertext.

1 Set $\mathsf{ACC}_0 = \mathsf{RLWE\text{-}Pack}((q^{-1}\xi_q^{b_1}, 0), ..., (q^{-1}\xi_q^{b_r}, 0))$, "$\mathcal{R}_{12}$"), where $q^{-1} \in \mathcal{R}_Q$;
2 **for** $k = 1$ **to** n **do**
3 $\quad \mathsf{ACC}_k = \text{Batch-BR}\left(\mathsf{ACC}_{k-1}, \mathsf{BK}_k \in \mathsf{RGSW}(s_k), \{a_{ik}\}_{i\in[r]}\right)$;
4 Set $\mathsf{test} = \left(\sum_{y\in\mathbb{Z}_q \& \lfloor y \rceil_2 = 1} \xi_q^{-y}\right)$, $\boldsymbol{d} = (\sum_{i\in[r]} q^{-1}\mathsf{v}_i, 0)$;
$\quad$ // $\boldsymbol{d} \in \mathsf{RLWE}_{s'}(q^{-1}\sum_{i\in[r]} \mathsf{v}_i)$;
5 Return $\boldsymbol{c} = \boldsymbol{d} + \mathsf{Eval\text{-}Tr}_{K/K_{23}}(\mathsf{test} \cdot \mathsf{ACC}_n)$.

Proof. We first analyze the correctness. By applying Theorem 6.1 to the for loop in Step 2, we can obtain that ACC_n encrypts $\sum_{i\in[r]} q^{-1}\xi_q^{b_i - \langle \boldsymbol{a}_i, \mathbf{s}\rangle}\mathsf{v}_i$, (assuming n is even, which is without loss of generality). Next we use an important fact observed by the work [1] – For any $y = \xi_q^z$, the following equation holds.

$$1 + y + y^2 + \cdots + y^{q-1} = \begin{cases} q & \text{if } z = 0 \mod q \\ 0 & \text{otherwise} \end{cases}.$$

As y is a power of ξ_q, we can further express the equation as $\sum_{0\le i<q} y^i = 1 + \mathsf{Tr}_{K/K_{23}}(y)$, when q is a prime (which follows by our parameter choice). This corresponds to what step 5 does.

By using this fact, it is easy to verify the following: for any $\xi_q^z, z \in \mathbb{Z}_q$, let $x = \mathsf{test} \cdot (q^{-1}\xi_q^z)$. Then we have $q^{-1} + \mathsf{Tr}_{K/K_{23}}(x) = \begin{cases} 1 & \text{if } \lfloor z \rceil_2 = 1 \\ 0 & \text{otherwise.} \end{cases} = \lfloor z \rceil_2 .$

By the above equation with our batch computation, the final output of the algorithm would be encrypting $\sum_{i\in[r]} \lfloor b_i - \langle \boldsymbol{a}_i, \mathbf{s}\rangle \rceil_2 \cdot \mathsf{v}_i$. By the correctness of decryption for LWE, the resulting ciphertext belongs to $\mathsf{RLWE}^Q_{s'}(\sum \mu_i \mathsf{v}_i)$. The same analysis works for the case if n is odd, i.e., the resulting ciphertext belongs to $\mathsf{RLWE}^Q_{s'}(\sum \mu_i \mathsf{w}_i)$. Thus, the correctness is proved.

Next we analyze the noise growth. We first analyze the noise of the ACC_n in the for loop and then that of the next stage. By our batch computation framework, we have for $k \in [n]$, $\mathsf{Err}(\mathsf{ACC}_k) = \mathsf{Err}\left(\mathsf{Eval\text{-}Tr}(\mathsf{BK}_k \boxtimes \mathsf{ACC}_{k-1})\right)$. We denote e_k as the $\mathsf{Err}(\mathsf{ACC}_k)$, m_k as the message of BK_k, e'_k as the additive error from the key-switching algorithm in $\mathsf{Eval\text{-}Tr}$. Without loss of generality, we consider that ACC_{k-1} is of the mode "$\mathcal{R}_{12}$", and then identify the recursive relation as following:

$$\begin{aligned} e_k =& e'_k + \mathsf{Tr}_{K/K_{13}}(\mathsf{Err}(\mathsf{BK}_k)\mathbf{G}^{-1}(\mathsf{ACC}_{k-1})) + \mathsf{Tr}_{K/K_{13}}(m_k \cdot e_{k-1}) \\ =& e''_k + \mathsf{Tr}_{K/K_{13}}(m_k \cdot e_{k-1}), \end{aligned}$$

where $e''_k = e'_k + \mathsf{Tr}_{K/K_{13}}(\mathsf{Err}(\mathsf{BK}_k)\mathbf{G}^{-1}(\mathsf{ACC}_{k-1}))$. We notice that e'_k is *fresh noise* from KS and $\mathsf{Err}(\mathsf{BK}_k)\mathbf{G}^{-1}(\mathsf{ACC}_{k-1})$ is also independent of the recursion index k. Thus, e''_k can also be viewed as *non-accumulating noise* that does not increase over the recursion. Next we further expand the equation and obtain:

$$\begin{aligned} e_k =& e''_k + \mathsf{Tr}_{K/K_{13}}(m_k \cdot e_{k-1}) \\ =& e''_k + \mathsf{Tr}_{K/K_{13}}\left(m_k \cdot \left(e''_{k-1} + \mathsf{Tr}_{K/K_{12}}(m_{k-1} \cdot e_{k-2})\right)\right) \\ =& e''_k + \mathsf{Tr}_{K/K_{13}}\left(m_k \cdot e''_{k-1}\right) + \mathsf{Tr}_{K/K_{13}}\left(m_k \cdot \mathsf{Tr}_{K/K_{12}}(m_{k-1} \cdot e_{k-2})\right) \\ =& \widetilde{e}_k + \mathsf{Tr}_{K/K_{13}}\left(m_k \cdot \mathsf{Tr}_{K/K_{12}}(m_{k-1} \cdot e_{k-2})\right), \end{aligned}$$

where $\widetilde{e}_k = e''_k + \mathsf{Tr}_{K/K_{13}}\left(m_k \cdot e''_{k-1}\right)$. A similar argument as above shows that $\widetilde{e}_k$ is independent of the recursion index k and thus non-accumulating.

To proceed with the analysis, we first define the following notation. Without loss of generality, we only consider the case where n and k are both even.

Definition 6.3 *Let $m_k, \ldots, m_1$ be the packed messages as used in the algorithm, and let $e \in \mathcal{R}$ be some input. Define*

$$T^{2j}(e) = \begin{cases} \mathsf{Tr}_{K/K_{13}}\left(m_k \mathsf{Tr}_{K/K_{12}}\left(m_{k-1} \cdot e\right)\right) & \text{for } j = 1 \\ T^{2j-2}\left(\mathsf{Tr}_{K/K_{13}}\left(m_{k-2j+1}\mathsf{Tr}_{K/K_{12}}\left(m_{k-2j} \cdot e\right)\right)\right) & \text{for } j \in [2, k/2] \end{cases}.$$

Then we can unfold the recursive formula and obtain the following expression.

$$e_k = \widetilde{e}_k + T^2(\widetilde{e}_{k-2}) + T^4(\widetilde{e}_{k-4}) + \cdots + T^k(e_0).$$

To derive an upper bound for the above, we first prove the following claim:

Claim 6.4 *For $j \le k/2$ and any $e \in \mathcal{R}$ such that $\|e\|_\infty$ is B bounded, then $\|T^{2j}(e)\|_\infty \le 4p_1p_2r^2B$.*

Proof. We start from the base case $j = 1$. First we can express $e = \sum_{i\in[\rho]} e_i \mathsf{v}_i$ where each $e_i \in \mathcal{R}_{13}$. From our design, we notice that $m_{k-1} = \sum_{i\in[r]} \xi_q^{s_{k-1}} \mathsf{v}_i^\vee \mathsf{w}_i$, and $m_k = \sum_{i\in[r]} \xi_q^{s_k} \mathsf{w}_i^\vee \mathsf{v}_i$. Therefore, $E := \mathsf{Tr}_{K/K_{13}}(m_{k-1} \cdot e) = \sum_{i\in[r]} \xi_q^{s_{k-1}} e_i \mathsf{w}_i$ as the cross terms $\mathsf{v}_i^\vee \mathsf{v}_j$ are all cancelled out by the trace for $i \neq j$. According to our choice of the basis, we have $\|E\|_\infty \leq \rho \cdot \|m_{k-1}\| \cdot \|e\|_\infty \leq \rho \cdot r \cdot 2(p_1-1)/\rho' \cdot B \leq 2p_1 rB$. We can further express $E = \sum_{i\in[r]} \xi_q^{s_{k-1}} z_i \mathsf{w}_i$, where each $z_i \in \mathcal{R}_1$. We then use the fact $\|z_i\|_\infty \leq \tau \cdot \|E\|_\infty \cdot \|\mathsf{w}_i^\vee\|_\infty$ as implicitly analyzed in [26]. By plugging the bound of the basis and $\|E\|_\infty$, we have $\|z_i\|_\infty \leq 4p_1p_2rB$, for every $i \in [r]$. Then $\mathsf{Tr}_{K/K_{12}}(m_k \cdot \sum_{i\in[r]} \xi_q^{s_{k-1}} z_i \mathsf{w}_i) = \sum_{i\in[r]} \xi_q^{s_k+s_{k-1}} z_i \mathsf{v}_i$. Thus, $\|T^2(e)\|_\infty \leq \sum_{i\in[r]} \|z_i\|_\infty \leq 4p_1p_2r^2B$.

For $j \geq 2$, let $s'_j = \sum_{t\leq j}(s_{k-t+1} + s_{k-t})$. Then we can use the same calculation to obtain $T^{2j}(e) = \sum_{i\in[r]} \xi_q^{s'_{ij}} z_i \mathsf{v}_i$. This means that the coefficient with respect to v_i remains the same but with different phase, i.e., and only the exponent on ξ_q changes but z_i does not. Therefore, $\|T^{2j}(e)\|_\infty \leq 4p_1p_2r^2B$ under the same analysis as above. □

Next, we can check that each coefficient of $\widetilde{e}_k$ is bounded by a subgaussian with parameter less than $O(r\sqrt{N \log Q} \cdot E)$. Therefore, by setting B as this quantity, the above claim proves that $\|e_k\|_\infty$ is bounded by a subgaussian with parameter less than $O(kr^3\sqrt{N \log Q}E)$ (as p_1, p_2 are constants according to our parameter selection). By plugging $k = n$, we conclude that $\|e_n\|_\infty$ is bounded by a subgaussian with parameter less than $O(nr^3\sqrt{N \log Q}E)$. In Step 4, we further multiply the ACC by the test vector, which at most increase the error by a factor of q. In step 5, we apply another homomorphic trace function $\mathsf{Eval\text{-}Tr}_{K/K_{23}}()$, which increases the error by at most q. So the final error is bounded by a subgaussian with parameter less than $O(nr^3q^2\sqrt{N \log Q}E) = O(nrqN\sqrt{N \log Q}E)$ and we prove this theorem. □

Remark 6.5 *We can easily unpack the output* RLWE *ciphertext. If the output* $\boldsymbol{c} \in \mathsf{RLWE}^Q_{s'}(\sum \mu_i \mathsf{v}_i)$*, then by applying* $(\mathsf{RLWE})\text{-}\mathsf{Eval\text{-}Tr}_{K/K_{13}}$ *to* $\mathsf{v}_i^\vee \boldsymbol{c}$*, the result is a* RLWE *ciphertext in* $\mathsf{RLWE}^Q_{s'}(\mu_i)$*. The other case can be achieved similarly.*

Remark 6.6 *We notice that the techniques in the analysis of Theorem 6.2 (specifically Claim 6.4) can be used to analyze batch homomorphic computation of branching programs (e.g., [9]) under our framework. In the same way, we can show that our batch framework only incurs a polynomial error growth for computing any constant-width polynomial-depth branching program.*

6.3 Efficiency

Finally, we compare the efficiency of the batch bootstrapping with the sequential non-batch bootstrapping (that can be achieved within a polynomial modulus). We first notice that one call to the non-batch AP14/FHEW framework would require at least $O(n)$ external products, even just counting the step of blind rotate. Thus, to bootstrap r input ciphertexts sequentially, it would require $O(rn)$ external products.

On the other hand, our batch blind-rotate for bootstrapping r input ciphertexts would require $O(n)$ external products and $O(n)$ calls to the homomorphic trace evaluation. We notice that each homomorphic trace evaluation would make $O(\log r)$ calls to the key-switch algorithm, which is roughly equal to $O(\log r)$ external products. The final step of equality test take q queries to the underlying key-switches. Thus, the overall algorithm would require $O(n \log r + q)$ external products to bootstrap r input ciphertexts.

Asymptotic Setting. Now we determine all the parameters in λ as follow. We can set $n = O(\lambda)$, $q = \tilde{O}(\sqrt{n})$, $N = O(n)$, and $r \approx O(\sqrt{N/q}) = O(\lambda^{1/4-o(1)})$ as the AP14/FHEW framework [4, 13, 17]. By plugging these parameters to the above analysis, our batch algorithm can therefore bootstrap $O(\lambda^{1/4-o(1)})$ input ciphertexts by using $\tilde{O}(\lambda)$ external products, implying the amortized complexity $\tilde{O}(\lambda^{0.75})$ external products per input ciphertext. On the other hand, the non-batch method would require the amortized complexity $O(\lambda)$ external products per input ciphertext.

Our theoretical advances can potentially lead to noticeable practical improvements, as all the components are explicit and have been implemented in the power-of-two's settings. By using the insights of [26], it is possible to port the existing implementations to the general cyclotomic rings, with the same asymptotic computational efficiency. We leave it as an interesting open direction to determine the concrete practical performances of our framework.

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. P. Abla, F.-H. Liu, H. Wang, and Z. Wang. Ring-based identity based encryption - asymptotically shorter MPK and tighter security. In K. Nissim and B. Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 157–187. Springer, Heidelberg, Nov. 2021.
2. M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. Estimate all the LWE, NTRU schemes! In D. Catalano and R. De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 351–367. Springer, Heidelberg, Sept. 2018.
3. J. Alperin-Sheriff and C. Peikert. Practical bootstrapping in quasilinear time. In Canetti and Garay [10], pages 1–20.
4. 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.
5. 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.

6. C. Bonte, I. Iliashenko, J. Park, H. V. L. Pereira, and N. P. Smart. FINAL: Faster FHE instantiated with NTRU and LWE. Cryptology ePrint Archive, Report 2022/074, 2022. https://eprint.iacr.org/2022/074.
7. 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.
8. Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, Oct. 2011.
9. Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In M. Naor, editor, *ITCS 2014*, pages 1–12. ACM, Jan. 2014.
10. R. Canetti and J. A. Garay, editors. *CRYPTO 2013, Part I*, volume 8042 of *LNCS*. Springer, Heidelberg, Aug. 2013.
11. H. Chen, W. Dai, M. Kim, and Y. Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In K. Sako and N. O. Tippenhauer, editors, *ACNS 21, Part I*, volume 12726 of *LNCS*, pages 460–479. Springer, Heidelberg, June 2021.
12. J. H. Cheon, A. Kim, M. Kim, and Y. S. Song. Homomorphic encryption for arithmetic of approximate numbers. In Takagi and Peyrin [36], pages 409–437.
13. 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.
14. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In Takagi and Peyrin [36], pages 377–408.
15. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, Jan. 2020.
16. I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In M. Tibouchi and H. Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 670–699. Springer, Heidelberg, Dec. 2021.
17. 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.
18. C. Gentry. Fully homomorphic encryption using ideal lattices. In Mitzenmacher [30], pages 169–178.
19. C. Gentry, S. Halevi, and N. P. Smart. Better bootstrapping in fully homomorphic encryption. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 1–16. Springer, Heidelberg, May 2012.
20. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Canetti and Garay [10], pages 75–92.
21. S. Halevi and V. Shoup. Bootstrapping for HElib. Cryptology ePrint Archive, Report 2014/873, 2014. https://eprint.iacr.org/2014/873.
22. S. Halevi and V. Shoup. Faster homomorphic linear transformations in HElib. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 93–120. Springer, Heidelberg, Aug. 2018.
23. Y. Lee, D. Micciancio, A. Kim, R. Choi, M. Deryabin, J. Eom, and D. Yoo. Efficient fhew bootstrapping with small evaluation keys, and applications to thresh-

old homomorphic encryption. Cryptology ePrint Archive, Paper 2022/198, 2022. https://eprint.iacr.org/2022/198.
24. F.-H. Liu and H. Wang. Batch bootstrapping II: Bootstrapping in polynomial modulus only requires $\tilde{O}(1)$ fhe multiplications in amortization. In Eurocrypt 2023.
25. 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.
26. 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.
27. 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.
28. D. Micciancio and Y. Polyakov. Bootstrapping in fhew-like cryptosystems. In *WAHC '21: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021*, pages 17–28. WAHC@ACM, 2021.
29. 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.
30. M. Mitzenmacher, editor. *41st ACM STOC*. ACM Press, May / June 2009.
31. 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.
32. C. Peikert and Z. Pepin. Algebraically structured LWE, revisited. In D. Hofheinz and A. Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 1–23. Springer, Heidelberg, Dec. 2019.
33. 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.
34. 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.
35. R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, 1978.
36. T. Takagi and T. Peyrin, editors. *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*. Springer, Heidelberg, Dec. 2017.
37. V. Vaikuntanathan. Homomorphic encryption references. https://people.csail.mit.edu/vinodv/FHE/FHE-refs.html.