








# General Bootstrapping Approach for RLWE-Based Homomorphic Encryption

Andrey Kim , Maxim Deryabin , Jieun Eom , Rakyong Choi , Yongwoo Lee , *Member, IEEE*,  
Whan Ghang , and Donghoon Yoo 

**Abstract**—Homomorphic Encryption (HE) makes it possible to compute on encrypted data without decryption. In lattice-based HE, a ciphertext contains noise, which accumulates along with homomorphic computations. Bootstrapping refreshes the noise and it is possible to perform arbitrary-depth computations on HE with bootstrapping, which we call Fully Homomorphic Encryption (FHE). In this article, we propose a new general bootstrapping technique for RLWE-based schemes and its practical instantiation for FHE. It can be applied to all three RLWE-based leveled FHE schemes: Brakerski-Gentry-Vaikuntanathan (BGV), Brakerski/Fan-Vercauteren (BFV), and Cheon-Kim-Kim-Song (CKKS) with minor deviations in the algorithms. Our new construction of bootstrapping extracts a noiseless ciphertext for a part of the input, scales it, and finally removes it. In contrast with previous bootstrapping algorithms, the proposed method consumes only 1–2 levels and uses smaller parameters. For BGV and BFV, our new bootstrapping does not have any restrictions on a plaintext modulus unlike typical cases of the previous methods. The error introduced by our approach for CKKS is comparable to a rescaling error, allowing us to preserve a large amount of precision after bootstrapping.

**Index Terms**—Bootstrapping, fully homomorphic encryption, public key cryptography.

## I. INTRODUCTION

**H**OMOMORPHIC encryption (HE) is a form of encryption that enables computations on encrypted data without revealing the secret key. The majority of HE schemes rely on Learning with Errors (LWE) [1] or Ring Learning with Errors (RLWE) [2] problem, and their ciphertexts contain “noise” to ensure security. However, since the noise grows during computations and eventually can destroy the message, the number of operations in encrypted form is limited. After the first construction of the fully homomorphic encryption (FHE) scheme by Gentry [3], significant progress has been made in the direction of research on HE. Gentry’s idea of *bootstrapping* as a homomorphic evaluation of a decryption circuit allows to refresh the noise, and thus more computation can be done on the ciphertext.

The most common FHE schemes can be categorized into FHEW-type, which operate primarily on Boolean circuits [4],

[5], [6], BGV/BFV-type [7], [8], [9] commonly designed for computations on finite rings and CKKS-type [10], [11] designed for computations over real and complex numbers. This approach differs not only by plaintext space, but also by the design of the bootstrapping algorithms.

The core idea of the bootstrapping procedure for FHEW-type schemes is a so-called *blind rotation technique* [4], [5]. It introduces small controllable additive errors, and can be realized in small parameters. For BGV/BFV/CKKS schemes, the bootstrapping is considered to be much more complex homomorphic operation which requires substantial computational resources at the same time with significant storage and communication overhead. This drawback is mitigated by the amortized structure of the RLWE ciphertext which allows to process large amount of encrypted values simultaneously. For BGV and BFV schemes, all known bootstrapping techniques are performed by removing bits of error homomorphically without damaging encrypted message [12], [13]. A more general approach to the digit extraction procedure was proposed by Halevi and Shoup [14] and the Chen/Han procedure [15] improved it in terms of multiplicative depth in cases when high exponents in the plaintext modulus are used. Meanwhile, for the CKKS scheme, the bootstrapping procedure is based on the polynomial approximation to a modular reduction function proposed by Cheon et al. [16]. Subsequent studies have focused on more accurate approximations of the modular reduction function to improve accuracy [17], [18], [19], [20], [21], [22]. Bae et al. [23] achieved any desired level of precision for the bootstrapped ciphertext by recursively applying a polynomial approximation.

In summary for all BGV/BFV/CKKS schemes the conventional bootstrapping methods are based on predefined high-order polynomial evaluation, consuming considerable multiplicative levels, and thus large parameters are required to make this bootstrapping procedure feasible. For BGV/BFV, the complexity of bootstrapping increases with the size of the plaintext modulus, so to make bootstrapping practical, significant constraints must be applied to the plaintext modulus [15]. At the same time, the approximation approach of the CKKS scheme relies on the small size of secret key distribution. Commonly, sparse key distribution is used to construct the CKKS scheme with bootstrapping. Thus, approximation approaches become less attractive for the case of larger secret key distributions, which arises in different applications such as threshold cryptography [6].

Manuscript received 14 December 2022; revised 3 September 2023; accepted 10 September 2023. Date of publication 22 September 2023; date of current version 22 December 2023. Recommended for acceptance by A. Stavrou. (*Corresponding author: Andrey Kim.*)

The authors are with the Samsung Advanced Institute of Technology, Suwon 16678, Republic of Korea (e-mail: kimandr.kz@gmail.com; max.deriabin@samsung.com; jieun.eom@samsung.com; rakyong.choi@samsung.com; yongwoo@inha.ac.kr; whan.ghang@samsung.com; donghoon.yoo@desilo.ai).

Digital Object Identifier 10.1109/TC.2023.3318405

### A. Our Results

We introduce a new bootstrapping technique that can be applied to RLWE-based HE schemes generally and enjoys the following properties:

- The proposed bootstrapping supports all three BGV/BFV/CKKS schemes only with small modifications.
- For BGV/BFV, our technique does not have restrictions on the size of plaintext modulus.
- For CKKS, our technique almost does not lose the precision of the message and is suitable for employing noise flooding workarounds to resist the CKKS vulnerability described in [24].
- Our bootstrapping consumes only 1–2 levels and can be performed for smaller ring dimensions, making it an attractive method for hardware acceleration.
- Our bootstrapping supports both ternary and discrete Gaussian secret key distributions due to the state-of-the-art blind rotation technique [6].

The proposed technique does not use homomorphic digit extraction or polynomial approximation to evaluate the modular reduction function. Instead, it squeezes out a *noiseless* ciphertext of the quotient and scales it up. Then, it performs modular reduction by canceling out this scaled quotient term from its original ciphertext. More precisely, it consists of three separated stages: HomRound, ScaledMod, and Combine. For the input ciphertext with a small modulus  $q$ , HomRound outputs a *noiseless* ciphertext of the quotient  $u$  to be removed by modular reduction. ScaledMod scales up the ciphertext by  $q$ , which outputs a ciphertext of  $qu$  with a higher modulus  $Q$  and eventually  $qu$  is removed from the original ciphertext in Combine step.

For the concrete implementation, we adapt the blind rotation technique [4], [5], [6] and the RLWE ciphertext repacking technique [25] to evaluate ScaledMod. The advantage of blind rotation, first introduced by Ducas and Micciancio [4] for the FHEW scheme, is that it is not only based on RGSW multiplications [26] instead of polynomial evaluation, but also introduces only small controllable additive errors and consequently can be realized at smaller parameters.

We would like to note that, due to the non-amortized nature of blind rotation in the current state, the proposed bootstrapping approach is not considered as an optimization over state-of-the-art bootstrapping techniques in the general case. Instead, our approach can be seen as one of the earliest successful attempts to combine the benefits of using smaller parameters in bootstrapping for FHEW-type schemes with BGV, BFV, and CKKS schemes. The newest results for conventional CKKS bootstrapping [23] published since the first preprint of this work showed reasonable performance while achieving arbitrary precision. Our proposed method demonstrates its best performance with only small number of slots while allows to preserve small number of levels for the refreshed ciphertext. Although we did not achieve better practical outcomes compared to existing methods, we strongly believe that the technique we propose has great potential for future development in the field of RLWE/LWE-based cryptography and homomorphic cryptography.

Meanwhile many recent research results [27], [28], [29] are paying attention to FHEW-type schemes. It is shown that conversions between LWE and RLWE combined with the blind rotation can be an efficient base technique for evaluating non-polynomial functions for FHE such as a sign function and other neural network activation functions [27], [28], [29]. Very recent results on amortized functional bootstrapping [30] demonstrates that FHEW-type bootstrapping also can be parallelized.

### B. Organization

This article is organized as follows. In Section II, we start with some preliminaries on lattice-based structures and operations with them. We present the idea and the core algorithm for our BGV/BFV/CKKS bootstrapping in Section III. In Section IV, the detailed instantiation of ScaledMod using blind rotation and repacking technique is explained. Section V provides discussion and the implementation results. Finally, Section VI concludes the article.

## II. PRELIMINARIES

All logarithms are base 2 unless otherwise indicated. For two vectors  $\vec{a}$  and  $\vec{b}$ , we denote their inner product by  $\langle \vec{a}, \vec{b} \rangle$ . Let  $N$  be a power of two, we denote the  $2N$ -th cyclotomic ring by  $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$  and its quotient ring by  $\mathcal{R}_Q := \mathcal{R}/Q\mathcal{R}$ . Ring elements are indicated in bold, e.g.  $\mathbf{a} = \mathbf{a}(X)$ . We write the floor, ceiling and round functions as  $\lfloor \cdot \rfloor$ ,  $\lceil \cdot \rceil$  and  $\lfloor \cdot \rceil$ , respectively. For  $q \in \mathbb{Z}$ ,  $q > 1$  we identify the ring  $\mathbb{Z}_q$  with  $(-q/2, q/2]$  as the representative interval, and for  $x \in \mathbb{Z}$  we denote the centered remainder of  $x$  modulo  $q$  by  $[x]_q \in \mathbb{Z}_q$ . We extend these notations to elements of  $\mathcal{R}$  by applying them coefficient-wise. For  $\mathbf{a} = a_0 + a_1 \cdot X + \dots + a_{N-1} \cdot X^{N-1} \in \mathcal{R}$ , we denote the  $\ell_\infty$  norm of  $\mathbf{a}$  as  $\|\mathbf{a}\|_\infty = \max_{0 \leq i < N} \{|a_i|\}$ . We use  $\mathbf{a} \leftarrow S$  to denote uniform sampling from the set  $S$ . We denote sampling according to a distribution  $\chi$  by  $\mathbf{a} \leftarrow \chi$ . We assume that  $\vec{t}_g = (t_0, \dots, t_{d-1})$  is a gadget decomposition of  $\mathbf{t} \in \mathcal{R}_Q$  if  $\mathbf{t} = \sum_{i=0}^{d-1} g_i \cdot \mathbf{t}_i$  where  $\vec{g} = (g_0, \dots, g_{d-1})$  is a gadget vector [31].

### A. Basic Lattice-Based Encryption

For positive integers  $q$  and  $n$ , basic LWE encryption of  $m \in \mathbb{Z}$  under the secret key  $\vec{s} \leftarrow \chi_{\text{key}}$  is defined as

$$\mathcal{L}_{q,\vec{s}}(m) = (\vec{a}, b) = (\vec{a}, -\langle \vec{a}, \vec{s} \rangle + e + m) \in \mathbb{Z}_q^{n+1},$$

where  $\vec{a} \leftarrow \mathbb{Z}_q^n$  and error  $e \leftarrow \chi_{\text{err}}$ . We occasionally drop subscripts  $q$  and  $\vec{s}$  when they are obvious from the context. We use the notation  $\mathcal{L}_{q,\vec{s}}^0(m)$  if the error  $e$  is zero.

For a positive integer  $Q$  and a power of two  $N$ , basic RLWE encryption of  $\mathbf{m} \in \mathcal{R}$  under the secret key  $\mathbf{s} \leftarrow \chi_{\text{key}}$  is defined as

$$\mathcal{R}_{Q,\mathbf{s}}(\mathbf{m}) := (\mathbf{a}, -\mathbf{a} \cdot \mathbf{s} + e + \mathbf{m}) \in \mathcal{R}_Q^2,$$

where  $\mathbf{a} \leftarrow \mathcal{R}_Q$  and  $e_i \leftarrow \chi_{\text{err}}$  for each coefficient  $e_i$  of  $e \in \mathcal{R}$ , for  $i \in [0, N-1]$ . As in LWE, we will occasionally drop subscripts  $Q$  and  $\mathbf{s}$ . We also use the notation  $\mathcal{R}_{Q,\mathbf{s}}^0(\mathbf{m})$  if the error  $e$  is zero.

A ciphertext  $\text{ct} = \mathcal{R}_{Q,s}(\mathbf{m}) = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$  is decrypted by computing

$$\mathcal{R}_{Q,s}^{-1}(\mathbf{a}, \mathbf{b}) := \mathbf{a} \cdot \mathbf{s} + \mathbf{b} = \mathbf{m} + \mathbf{e} \in \mathcal{R}_Q. \quad (1)$$

We use shorthand notation  $\text{ct}(\mathbf{s}) := \mathcal{R}_{Q,s}^{-1}(\text{ct})$ . Following [32], we define  $\mathcal{R}^{\vec{g}}$  encryption as

$$\mathcal{R}^{\vec{g}}(\mathbf{m}) := \begin{pmatrix} \mathcal{R}(g_0 \cdot \mathbf{m}) \\ \vdots \\ \mathcal{R}(g_{d-1} \cdot \mathbf{m}) \end{pmatrix} \in \mathcal{R}^{d \times 2}$$

for a gadget vector  $\vec{g} \in \mathcal{R}^d$  and represent the scalar multiplication as

$$\odot : \mathcal{R}_Q \times \mathcal{R}_{Q,s}^{\vec{g}} \rightarrow \mathcal{R}_{Q,s}.$$

Let  $\vec{t}_g = (t_0, \dots, t_{d-1})$  is a gadget decomposition of  $\mathbf{t} \in \mathcal{R}_Q$  corresponding to  $\vec{g}$ . The scalar multiplication between  $\mathbf{t}$  and  $\mathcal{R}_{Q,s}^{\vec{g}}$  ciphertext is defined using the following rule

$$\begin{aligned} \mathbf{t} \odot \mathcal{R}^{\vec{g}}(\mathbf{m}) &= \sum_{i=0}^{d-1} t_i \cdot \mathcal{R}(g_i \cdot \mathbf{m}) \\ &= \mathcal{R} \left( \sum_{i=0}^{d-1} g_i \cdot t_i \cdot \mathbf{m} \right) = \mathcal{R}(\mathbf{t} \cdot \mathbf{m}) \in \mathcal{R}_Q^2. \end{aligned}$$

For each error  $\mathbf{e}_i$  in  $\mathcal{R}(g_i \cdot \mathbf{m})$ , the error after multiplication is equal to  $\sum_{i=0}^{d-1} t_i \cdot \mathbf{e}_i$  which is small as  $t_i$  and  $\mathbf{e}_i$  are small.

### B. Key Switching in $\mathcal{R}$

The key switching operation converts  $\mathcal{R}_{Q,s_1}(\mathbf{m})$  encrypted by a secret key  $\mathbf{s}_1$  to  $\mathcal{R}_{Q,s_2}(\mathbf{m})$  encrypted by a new secret key  $\mathbf{s}_2$ . There are different variants of the key switching technique and readers can refer to literature such as [33] for more details. We focus on the BV key switching type [34] and its Residue Number System (RNS) variants [12] that fit our approach.

- **KEYSWITCHGEN**( $\mathbf{s}_1, \mathbf{s}_2$ ): Outputs  $\mathcal{R}_{Q,s_2}^{\vec{g}}(\mathbf{s}_1)$ .
- **KEYSWITCH** $_{\mathbf{s}_1 \rightarrow \mathbf{s}_2}(\mathcal{R}_{Q,s_1}(\mathbf{m}))$ : Given  $\mathcal{R}_{Q,s_1}(\mathbf{m}) = (\mathbf{a}, \mathbf{b})$ , it evaluates

$$\mathcal{R}_{Q,s_2}(\mathbf{m}) = \mathbf{a} \odot \mathcal{R}_{Q,s_2}^{\vec{g}}(\mathbf{s}_1) + (0, \mathbf{b}) \pmod{Q}.$$

$\mathcal{R}_{Q,s_2}^{\vec{g}}(\mathbf{s}_1)$  generated by **KEYSWITCHGEN** is a public key switching key. The key switching error is equal to the error of  $\mathcal{R} \odot \mathcal{R}^{\vec{g}}$  multiplication.

*Remark 1:* Key switching usually requires auxiliary modulus to manage the error growth. However, we do not employ an auxiliary modulus as the key switching error in our approach will be managed in a different way.

### C. Automorphism in $\mathcal{R}$

To perform some operations in FHE, we use an automorphism procedure over  $\mathcal{R}$ . There are  $N$  automorphisms of  $\mathcal{R}$ , namely  $\psi_t : \mathcal{R} \rightarrow \mathcal{R}$  given by  $\mathbf{a}(X) \mapsto \mathbf{a}(X^t)$  for  $t \in \mathbb{Z}_{2N}^*$ . Automorphism over  $\mathcal{R}$  instances can be defined as follows.

- **EVALAUTO**( $\mathcal{R}_{Q,s}(\mathbf{m}), t$ ): Given  $\mathcal{R}_{Q,s}(\mathbf{m}(X)) = (\mathbf{a}(X), \mathbf{b}(X))$ , it applies  $\psi_t$  to  $\mathbf{a}(X)$  and  $\mathbf{b}(X)$  and

obtains  $(\mathbf{a}(X^t), \mathbf{b}(X^t))$  which is an  $\mathcal{R}$  encryption of  $\mathbf{m}(X^t)$  under the secret key  $\mathbf{s}(X^t)$ . Then it performs key switching from  $\mathbf{s}(X^t)$  to  $\mathbf{s}(X)$  and outputs  $\mathcal{R}_{Q,s}(\mathbf{m}(X^t)) = \mathcal{R}_{Q,s}(\psi_t(\mathbf{m}))$ .

The additional error after applying an automorphism is equal to key switching error as an automorphism  $\psi_t$  is a norm-preserving map.

### D. Rescaling in $\mathcal{R}$

Rescaling is used in  $\mathcal{R}$  to control the error or message growth. We consider rescaling of  $\mathcal{R}_{Q,s}$  instance by  $q$  that divides  $Q$ .

- **RESCALE**( $\mathcal{R}_{Q,s}(\mathbf{m}), q$ ): Given  $\mathcal{R}_{Q,s}(\mathbf{m}) = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$ , it outputs

$$\mathcal{R}_{Q/q,s} \left( \frac{1}{q} \mathbf{m} \right) = \left( \left\lfloor \frac{\mathbf{a}}{q} \right\rfloor, \left\lfloor \frac{\mathbf{b}}{q} \right\rfloor \right) \in \mathcal{R}_{Q/q}^2.$$

The rescaling procedure also divides the error of  $\text{ct}$  by  $q$ , but introduces an additional rescaling error  $\mathbf{e}_{rs}$ . The rescaling error  $\mathbf{e}_{rs}$  is small and is bounded by a constant  $c_{rs} = \frac{1}{2}(1 + 6\sigma\delta_{\mathcal{R}})$  for Discrete Gaussian secret key  $\mathbf{s}$  with standard deviation  $\sigma$ , where  $\delta_{\mathcal{R}}$  be a polynomial multiplication expansion factor, i.e.  $\|\mathbf{a} \cdot \mathbf{b}\|_{\infty} \leq \delta_{\mathcal{R}} \|\mathbf{a}\|_{\infty} \|\mathbf{b}\|_{\infty}$  for any  $\mathbf{a}, \mathbf{b} \in \mathcal{R}$ . For  $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ , we use the bound  $\delta_{\mathcal{R}} = 2\sqrt{N}$  as shown in [35].

### E. RLWE Based HE Schemes

We briefly describe the encryption procedures for the three most common FHE schemes based on RLWE. The main differences in encryption are the message representation and the encoding procedure.

In the BGV scheme with the plaintext modulus  $t$ , a plaintext  $\mathbf{m}$  is encoded in the least significant bits in  $\mathcal{R}_Q$ , and its encryption is given as follows:

$$\text{ENC}_{\text{BGV}}(\mathbf{m}) = (\mathbf{a}, -\mathbf{a} \cdot \mathbf{s} + t \cdot \mathbf{e} + \mathbf{m}).$$

In this case error  $\mathbf{e}$  is scaled by modulus  $t$  that gives an ability to distinguish it from the plaintext.

In the BFV scheme with the plaintext modulus  $t$ , a plaintext  $\mathbf{m}$  is encoded in the most significant bits in  $\mathcal{R}_Q$  and its encryption is given as follows:

$$\text{ENC}_{\text{BFV}}(\mathbf{m}) = \left( \mathbf{a}, -\mathbf{a} \cdot \mathbf{s} + \mathbf{e} + \left\lfloor \frac{Q}{t} \cdot \mathbf{m} \right\rfloor \right).$$

The CKKS scheme is an approximate homomorphic encryption scheme and RLWE errors are considered a part of messages. Its encryption of a plaintext  $\mathbf{m}$  is given as follows:

$$\text{ENC}_{\text{CKKS}}(\mathbf{m}) = (\mathbf{a}, -\mathbf{a} \cdot \mathbf{s} + \mathbf{e} + \mathbf{m}).$$

All encryption algorithms assume that the message is already encoded into the polynomial  $\mathbf{m}$ . Encoding techniques for BGV and BFV can be found in [7], [8], [9] and for CKKS in [10], [11].



### F. Bootstrapping

Bootstrapping is an essential part of fully homomorphic encryption. Informally it can be understood as updating the ciphertext in such a way as to provide more room for operations. According to Gentry [3], the bootstrapping can be performed by *computing the decryption algorithm in encrypted form* under the same scheme. With the right choice of parameters, the noise after the bootstrapping becomes smaller and more operations could be performed on the *bootstrapped* ciphertext. In the case of Leveled HE, such as BGV and CKKS, the modulus switching operation ensures that the noise remains small, but at the cost of reducing the ciphertext modulus. Thus the bootstrapping in *leveled schemes* can be viewed as a procedure for *increasing the ciphertext modulus rather than reducing noise*. In both cases bootstrapping reduces the relative noise compared to ciphertext modulus. All three BGV, BFV, and CKKS schemes capture this idea, but all previous bootstrapping techniques are highly dependent on the encryption/decryption procedures and differ from each other [14], [15], [16].

### III. GENERAL BOOTSTRAPPING

We first describe the general bootstrapping approach at a high level for the CKKS scheme, for simplicity. As mentioned above, bootstrapping can be understood as an homomorphic evaluation of decryption query (1), which includes modular reduction operation. We remark that given an RLWE ciphertext  $\text{ct} = (a, b) \in \mathcal{R}_q^2$  of a message  $m$ , the decryption of the CKKS scheme is defined as

$$\text{ct}(s) = a \cdot s + b = m + e \in \mathcal{R}_q,$$

where  $s$  is a secret key and  $e$  is an error. The output of the bootstrapping should be an encryption  $\text{ct}_{\text{boot}} = (a_{\text{boot}}, b_{\text{boot}}) \in \mathcal{R}_Q^2$  of  $m$  for some  $Q > q$  such that

$$\text{ct}_{\text{boot}}(s) = a_{\text{boot}} \cdot s + b_{\text{boot}} = m + e_{\text{boot}} \in \mathcal{R}_Q,$$

where the error  $e_{\text{boot}} \approx e$ . If we represent the ciphertext  $\text{ct} \in \mathcal{R}_q$  in a higher modulus  $Q$ , then the decryption query becomes

$$\text{ct}'(s) = a \cdot s + b = m + e + qu \in \mathcal{R}_Q. \quad (2)$$

Our goal is to remove  $q \cdot u$  part from (2). The proposed bootstrapping exploits the novel HomRound, which finds an *errorless* encryption of the quotient  $u$ . HomRound corresponds to *homomorphic evaluation of rounding-by- $q$* , but the result does not contain noise. In detail, the proposed bootstrapping consists of three parts as follows:

1.  $\text{HomRound}(\text{ct}) \rightarrow \text{ct}_{\text{init}} = (a_{\text{init}}, b_{\text{init}}) \in \mathcal{R}_{2N}^2$   
We first compute the errorless encryption of a small polynomial  $-u$  modulo  $p$  (to utilize blind rotation, we use  $p = 2N$  in the following section) from the input ciphertext  $\text{ct}$ , satisfying

$$\text{ct}_{\text{init}}(s) = a_{\text{init}} \cdot s + b_{\text{init}} = -u \in \mathcal{R}_p.$$

2.  $\text{ScaledMod}(\text{ct}_{\text{init}}, q, Q) \rightarrow \text{ct}_{\text{sm}} = (a_{\text{sm}}, b_{\text{sm}}) \in \mathcal{R}_Q^2$

The scaled modulus raising procedure that we call ScaledMod gives us a scaled encryption of  $q \cdot u$  which satisfies

$$\text{ct}_{\text{sm}}(s) = a_{\text{sm}} \cdot s + b_{\text{sm}} = -qu + e_{\text{sm}} \in \mathcal{R}_Q.$$

We can see that the modulus is scaled up from  $p$  to  $Q$ , and the message is scaled by  $q$  times. We note that  $Q/p > q$  is what we achieve in ScaledMod as  $Q/p \leq q$  case is trivial.<sup>1</sup> The input ciphertext  $\text{ct}_{\text{init}}$  is errorless, but  $\text{ct}_{\text{sm}}$  may contain error  $\|e_{\text{sm}}\| \ll q$ .

3.  $\text{Combine}(\text{ct}_{\text{sm}}, \text{ct}) \rightarrow \text{ct}_{\text{boot}} \in \mathcal{R}_Q^2$

Finally, we Combine the ciphertext  $\text{ct}$  and  $\text{ct}_{\text{sm}}$  modulo  $Q$  to eliminate  $qu$  term from  $\text{ct}$  and increase the size of the modulus from  $q$  to  $Q$  as

$$\text{ct}_{\text{boot}}(s) = a_{\text{boot}} \cdot s + b_{\text{boot}} = m + e + e_{\text{sm}} \in \mathcal{R}_Q.$$

The error grows linearly during ScaledMod procedure in our instantiation with blind rotation in the later section. Thus, we can adjust the error  $e_{\text{sm}}$  to be comparable to the rescaling error with only a single rescaling.

We present our bootstrapping in more details for the CKKS scheme, and provide full algorithms for BGV/BFV schemes together with their RNS variants. We use  $p = 2N$  in our instantiation of ScaledMod with blind rotation. Hence, the following algorithm is described for finding  $\text{ct}_{\text{init}} \in \mathcal{R}_{2N}^2$ , where  $N$  is a power of two. However, the following method can be easily generalized to any integer  $p$ .

#### A. Bootstrapping for CKKS

Given a ciphertext  $\text{ct} = \mathcal{R}_{q,s}(\mathbf{m}) = (a, b) \in \mathcal{R}_q^2$  for a small modulus  $q$ , the goal of the CKKS bootstrapping is to obtain a ciphertext  $\text{ct}_{\text{boot}} = \mathcal{R}_{Q,s}(\mathbf{m}) = (a_{\text{boot}}, b_{\text{boot}}) \in \mathcal{R}_Q^2$  of the same message  $\mathbf{m}$  for a bigger modulus  $Q > q$ . It starts from the decryption query of  $\text{ct}$  in the higher modulus  $Q$  represented by

$$\text{ct}(s) = a \cdot s + b = m + e + qv \in \mathcal{R}_Q$$

for some small polynomial  $v$ . To remove the  $qv$ , previous CKKS bootstrapping methods use homomorphic linear transformations and evaluation of approximating polynomials for modular reduction functions. On the other hand, our new bootstrapping technique computes  $qv$  part directly by using ScaledMod algorithm.

1) *Multiprecision CKKS*: In the multiprecision CKKS scheme [10], the ciphertext modulus  $q$  is a power of two. Given a ciphertext  $\text{ct} = (a, b) \in \mathcal{R}_q^2$  for a small modulus  $q$ , the decryption query is described as

$$\begin{aligned} \text{ct}(s) &= a \cdot s + b = m + e \pmod{q} \\ &= m + e + qv. \end{aligned}$$

Let  $q' = q/2N$  and  $\|m + e\|_{\infty} \leq \gamma < \frac{q}{4} - q'c_{\text{rs}}$  for some  $\gamma$ . Firstly, we compute  $\text{ct}' = \text{ct} \pmod{q'} = ([a]_{q'}, [b]_{q'}) \in \mathcal{R}_{q'}^2$  and obtain

$$\begin{aligned} \text{ct}'(s) &= [a]_{q'} \cdot s + [b]_{q'} = m + e \pmod{q'} \\ &= m + e + q'u. \end{aligned}$$

<sup>1</sup>It is done by multiplying by  $-q$  to  $\text{ct}_{\text{init}}$  and take modulus  $Q$ . Note that it is possible only if  $\text{ct}_{\text{init}}$  is noiseless as noise is also amplified by multiplying  $-q$  otherwise.

**Algorithm 1** Bootstrapping for CKKS

---

```

procedure BOOT-CKKS( $\text{ct} = (a, b) \in \mathcal{R}_q^2$ )
  HomRound( $\text{ct}$ )  $\rightarrow \text{ct}', \text{ct}_{\text{init}}$ 
  •  $\text{ct}' \leftarrow \text{ct} \pmod{q'}$ 
  •  $\text{ct}_{\text{init}} \leftarrow \left( \frac{\text{ct} - \text{ct}'}{q'} \right)$ 
  ScaledMod( $\text{ct}_{\text{init}}, q', Q$ )  $\rightarrow \text{ct}_{\text{sm}}$ 
  Combine( $\text{ct}_{\text{sm}}, \text{ct}'$ )  $\rightarrow \text{ct}_{\text{boot}}$ 
  •  $\text{ct}_{\text{boot}} \leftarrow \text{ct}_{\text{sm}} + \text{ct}' \pmod{Q}$ 
return  $\text{ct}_{\text{boot}} = (a_{\text{boot}}, b_{\text{boot}}) \in \mathcal{R}_Q^2$ 

```

---

Due to  $q'u = [a]_{q'} \cdot s + [b]_{q'} - (m + e)$ , we have  $\|u\|_\infty < c_{rs} + \frac{\gamma}{q'} < \frac{N}{2}$ . Now both  $a - [a]_{q'}$  and  $b - [b]_{q'}$  are divisible by  $q'$ , thus we can obtain a ciphertext

$$\text{ct}_{\text{init}} = \left( \frac{a - [a]_{q'}}{q'}, \frac{b - [b]_{q'}}{q'} \right) \in \mathcal{R}_{2N}^2.$$

It is easy to see that the initialized ciphertext  $\text{ct}_{\text{init}} = \mathcal{R}_{2N,s}^0(-u)$ , so we can evaluate  $\text{ScaledMod}(\text{ct}_{\text{init}}, q', Q)$  by setting  $c = \lfloor c_{rs} + \frac{\gamma}{q'} \rfloor$  for the initial function  $f$  and obtain  $\text{ct}_{\text{sm}} = \mathcal{R}_{Q,s}(-q'u)$  with an error  $e_{\text{sm}}$  such that

$$\text{ct}_{\text{sm}}(s) = a_{\text{sm}} \cdot s + b_{\text{sm}} = -q'u + e_{\text{sm}} \pmod{Q}.$$

We add it with  $\text{ct}'$  modulo  $Q$  and finally obtain the ciphertext  $\text{ct}_{\text{boot}} = \text{ct}_{\text{sm}} + \text{ct}' \pmod{Q}$  as

$$\begin{aligned} \text{ct}_{\text{boot}}(s) &= a_{\text{boot}} \cdot s + b_{\text{boot}} \\ &= m + e + q'u - q'u + e_{\text{sm}} \\ &= m + e + e_{\text{sm}} \pmod{Q}. \end{aligned}$$

The full algorithm is described in Algorithm 1.

2) *RNS-CKKS*: In the RNS-CKKS scheme [11], the modulus  $q$  is not a power of two but a product of primes, so the initializing steps are different. We start from the decryption query for the given ciphertext  $\text{ct} = (a, b) \in \mathcal{R}_q^2$  described as

$$\text{ct}(s) = a \cdot s + b = m + e + qv \in \mathcal{R}.$$

Let  $\|m + e\|_\infty \leq \gamma < \frac{q}{4} - \frac{qc_{rs}}{2N}$  for some  $\gamma$ . We first compute  $\text{ct}' = 2N \cdot \text{ct} \pmod{q} = ([2Na]_q, [2Nb]_q) \in \mathcal{R}_q^2$  to obtain

$$\text{ct}'(s) = [2Na]_q \cdot s + [2Nb]_q = 2Nm + 2Ne + qu \in \mathcal{R},$$

where  $\|u\|_\infty < c_{rs} + \frac{2N}{q} \cdot \gamma < \frac{N}{2}$ . Now both  $2Na - [2Na]_q$  and  $2Nb - [2Nb]_q$  are divisible by  $q$ , thus we can obtain a ciphertext

$$\text{ct}_{\text{init}} = \left( \frac{2Na - [2Na]_q}{q}, \frac{2Nb - [2Nb]_q}{q} \right) \in \mathcal{R}_{2N}^2.$$

Again for the initialized  $\text{ct}_{\text{init}} = \mathcal{R}_{2N,s}^0(-u)$ , we can evaluate  $\text{ScaledMod}(\text{ct}_{\text{init}}, q, Qp)$  by setting  $c = \lfloor c_{rs} + \frac{\gamma}{q'} \rfloor$  for the initial function  $f$  and obtain a ciphertext  $\text{ct}_{\text{sm}} = (a_{\text{sm}}, b_{\text{sm}})$  which satisfies

$$\text{ct}_{\text{sm}}(s) = a_{\text{sm}} \cdot s + b_{\text{sm}} = -qu + e_{\text{sm}} \pmod{Qp},$$

**Algorithm 2** Bootstrapping for RNS-CKKS

---

```

procedure BOOT-RNSCKKS( $\text{ct} = (a, b) \in \mathcal{R}_q^2$ )
  HomRound( $\text{ct}$ )  $\rightarrow \text{ct}', \text{ct}_{\text{init}}$ 
  •  $\text{ct}' \leftarrow 2N \cdot \text{ct} \pmod{q}$ 
  •  $\text{ct}_{\text{init}} \leftarrow \left( \frac{2N \cdot \text{ct} - \text{ct}'}{q} \right)$ 
  ScaledMod( $\text{ct}_{\text{init}}, q, Qp$ )  $\rightarrow \text{ct}_{\text{sm}}$ 
  Combine( $\text{ct}_{\text{sm}}, \text{ct}'$ )  $\rightarrow \text{ct}_{\text{boot}}$ 
  •  $\text{ct}'' \leftarrow \text{ct}_{\text{sm}} + \text{ct}' \pmod{Qp}$ 
  •  $\text{ct}_{\text{boot}} \leftarrow \text{RESCALE}\left(\frac{p}{2N} \cdot \text{ct}'', p\right)$ 
return  $\text{ct}_{\text{boot}} = (a_{\text{boot}}, b_{\text{boot}}) \in \mathcal{R}_Q^2$ 

```

---

**Algorithm 3** Bootstrapping for BGV

---

```

procedure BOOTSTRAP-BGV( $\text{ct} = (a, b) \in \mathcal{R}_q^2$ )
  HomRound( $\text{ct}$ )  $\rightarrow \text{ct}', \text{ct}_{\text{init}}$ 
  •  $\text{ct}' \leftarrow \text{ct} \pmod{q'}$ 
  •  $\text{ct}_{\text{init}} \leftarrow \left( \frac{\text{ct} - \text{ct}'}{q'} \right)$ 
  ScaledMod( $\text{ct}_{\text{init}}, q', Q$ )  $\rightarrow \text{ct}_{\text{sm}}$ 
  Combine( $\text{ct}_{\text{sm}}, \text{ct}'$ )  $\rightarrow \text{ct}_{\text{boot}}$ 
  •  $\text{ct}_{\text{boot}} \leftarrow \text{ct}_{\text{sm}} + \text{ct}' \pmod{Q}$ 
return  $\text{ct}_{\text{boot}} = (a_{\text{boot}}, b_{\text{boot}}) \in \mathcal{R}_Q^2$ 

```

---

where  $p$  is an auxiliary prime which we will rescale by later. Now we add  $\text{ct}'' = \text{ct}_{\text{sm}} + \text{ct}'$  modulo  $Qp$  and it satisfies

$$\begin{aligned} \text{ct}''(s) &= a'' \cdot s + b'' \\ &= 2N \cdot m + 2N \cdot e + qu - qu + e_{\text{sm}} \\ &= 2N \cdot m + 2N \cdot e + e_{\text{sm}} \pmod{Qp}. \end{aligned}$$

To get rid of the scaling factor  $2N$  in the message, we multiply  $\text{ct}''$  by  $\frac{p}{2N}$  and rescale the result by  $p$  as  $\text{ct}_{\text{boot}} = \text{RESCALE}(\frac{p}{2N} \cdot \text{ct}'', p)$ , then we have

$$\begin{aligned} \text{ct}_{\text{boot}}(s) &= a_{\text{boot}} \cdot s + b_{\text{boot}} \\ &= m + e + \frac{1}{2N} e_{\text{sm}} + e_{\text{rs}} \pmod{Q}. \end{aligned}$$

The full algorithm is described in Algorithm 2.

**B. Bootstrapping for BGV**

Given a plaintext space  $\mathcal{R}_t$  for some  $t$  which is normally taken as a prime power, the decryption query of the BGV scheme is

$$\text{ct}(s) = a \cdot s + b = m + te \in \mathcal{R}_q.$$

The bootstrapping algorithm for BGV is similar to that for CKKS with the only difference that public keys are generated with errors of the form  $te$  for multiprecision BGV and  $2Nte$  for RNS-BGV instead of  $e$  and the automorphism and rescale in Repack procedure are evaluated in accordance with BGV style. The algorithms for BGV and RNS-BGV bootstrappings are described in Algorithm 3 and Algorithm 4 respectively.

**C. Bootstrapping for BFV**

The decryption query of the BFV scheme is

$$\text{ct}(s) = a \cdot s + b = \frac{Q}{t} \cdot m + e \in \mathcal{R}_Q.$$

**Algorithm 4** Bootstrapping for RNS-BGV

---

```

procedure BOOT-RNSBGV( $\text{ct} = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$ )
  HomRound( $\text{ct}$ )  $\rightarrow$   $\text{ct}', \text{ct}_{\text{init}}$ 
  •  $\text{ct}' \leftarrow 2N \cdot \text{ct} \pmod{q}$ 
  •  $\text{ct}_{\text{init}} \leftarrow \left( \frac{2N \cdot \text{ct} - \text{ct}'}{q} \right)$ 
  ScaledMod( $\text{ct}_{\text{init}}, q, Q$ )  $\rightarrow$   $\text{ct}_{\text{sm}}$ 
  Combine( $\text{ct}_{\text{sm}}, \text{ct}'$ )  $\rightarrow$   $\text{ct}_{\text{boot}}$ 
  •  $\text{ct}'' \leftarrow (\text{ct}_{\text{sm}} + \text{ct}') \pmod{Q}$ 
  •  $\text{ct}_{\text{boot}} \leftarrow ([ (2N)^{-1} ]_Q) \cdot \text{ct}'' \pmod{Q}$ 
return  $\text{ct}_{\text{boot}} = (\mathbf{a}_{\text{boot}}, \mathbf{b}_{\text{boot}}) \in \mathcal{R}_Q^2$ 

```

---

**Algorithm 5** Bootstrapping for BFV

---

```

procedure BOOT-BFV( $\text{ct} = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$ )
  HomRound( $\text{ct}$ )  $\rightarrow$   $\text{ct}', \text{ct}'', \text{ct}_{\text{init}}$ 
  •  $\text{ct}' \leftarrow t \cdot \text{ct} \pmod{Q}$ 
  •  $\text{ct}'' \leftarrow \text{ct}' \pmod{Q'}$ 
  •  $\text{ct}_{\text{init}} \leftarrow \left( \frac{\text{ct}' - \text{ct}''}{Q'} \right) \pmod{2N}$ 
  ScaledMod( $\text{ct}_{\text{init}}, -Q', Qt$ )  $\rightarrow$   $\text{ct}_{\text{sm}}$ 
  Combine( $\text{ct}_{\text{sm}}, \text{ct}, \text{ct}''$ )  $\rightarrow$   $\text{ct}_{\text{boot}}$ 
  •  $\text{ct}''' \leftarrow \text{ct}_{\text{sm}} + t \cdot \text{ct} - \text{ct}'' \pmod{Qt}$ 
  •  $\text{ct}_{\text{boot}} \leftarrow \text{RESCALE}(\text{ct}''', t)$ 
return  $\text{ct}_{\text{boot}} = (\mathbf{a}_{\text{boot}}, \mathbf{b}_{\text{boot}}) \in \mathcal{R}_Q^2$ 

```

---

**Algorithm 6** Bootstrapping for RNS-BFV

---

```

procedure BOOT-RNSBFV( $\text{ct} = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$ )
  HomRound( $\text{ct}$ )  $\rightarrow$   $\text{ct}', \text{ct}'', \text{ct}_{\text{init}}$ 
  •  $\text{ct}' \leftarrow t \cdot \text{ct} \pmod{Q}$ 
  •  $\text{ct}'' \leftarrow 2N \cdot \text{ct}' \pmod{Q}$ 
  •  $\text{ct}_{\text{init}} \leftarrow \left( \frac{2N \cdot \text{ct}' - \text{ct}''}{Q} \right) \pmod{2N}$ 
  ScaledMod( $\text{ct}_{\text{init}}, -Q, Qpt$ )  $\rightarrow$   $\text{ct}_{\text{sm}}$ 
  Combine( $\text{ct}_{\text{sm}}, \text{ct}, \text{ct}''$ )  $\rightarrow$   $\text{ct}_{\text{boot}}$ 
  •  $\text{ct}''' \leftarrow \text{ct}_{\text{sm}} + 2Nt \cdot \text{ct} - \text{ct}'' \pmod{Qpt}$ 
  •  $\text{ct}_{\text{boot}} \leftarrow \text{RESCALE}(\frac{p}{2N} \cdot \text{ct}''', pt)$ 
return  $\text{ct}_{\text{boot}} = (\mathbf{a}_{\text{boot}}, \mathbf{b}_{\text{boot}}) \in \mathcal{R}_Q^2$ 

```

---

The goal of bootstrapping for BFV is to reduce the accumulated error instead of increasing the modulus size. During the bootstrapping procedure, the previous big error  $e$  is removed and replaced with a small refreshed error generated from ScaledMod and rescaling. The algorithms for BFV and RNS-BFV bootstrappings are described in Algorithm 5 and Algorithm 6 respectively.

## IV. SCALED MODULUS RAISING USING BLIND ROTATION

Our bootstrapping is described in the Section III in general way. Note that all three core operations HomRound, ScaledMod and Combine may be changed and adjusted to the concrete scheme or substituted with different algorithms. In this section, we discuss ScaledMod operation in detail. We show how blind rotation can be used to compute the ScaledMod function.

The ScaledMod algorithm transforms  $\mathcal{R}_{2N,s}^0(\mathbf{u})$ , where  $\|\mathbf{u}\|_\infty < N/2$ , to  $\mathcal{R}_{Q,s}(\Delta \cdot \mathbf{u})$  for a scaling factor  $\Delta$  and a

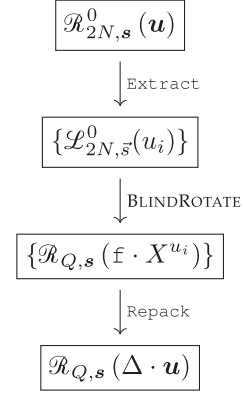


Fig. 1. The basic building block of the proposed ScaledMod.

large modulus  $Q$ . The ScaledMod algorithm is formalized as follows:

- **ScaledMod**( $\mathcal{R}_{2N,s}^0(\mathbf{u}), \Delta, Q$ ): It takes as input an errorless  $\mathcal{R}$  encryption of  $\mathbf{u}$  for a modulus  $2N$ , a scaling factor  $\Delta$ , and a modulus  $Q$ , where  $\|\mathbf{u}\|_\infty < N/2$ . Then it outputs an  $\mathcal{R}$  encryption of a scaled  $\Delta \cdot \mathbf{u}$  for a larger modulus  $Q$ , as  $\mathcal{R}_{Q,s}(\Delta \cdot \mathbf{u})$ .

To construct the ScaledMod algorithm, we adopt the blind rotation technique [4], [5], [6] used as a part of bootstrapping in FHEW-like schemes. We also employ the RLWE ciphertext repacking technique [25] and tailor it to suit our specific requirements. It is noted that ScaledMod is not limited to blind rotation and in the later section we will explore the possibility.

We first extract  $\mathcal{L}_{2N,s}^0(u_i)$  ciphertexts from the  $\mathcal{R}_{2N,s}^0(\mathbf{u})$  ciphertext. For each extracted  $\mathcal{L}_{2N,s}^0(u_i)$ , we perform the blind rotation with an initial function  $\mathbf{f} = -\Delta \cdot \sum_{j=-c}^c j \cdot X^j \in \mathcal{R}_Q$ , where  $\|\mathbf{u}\|_\infty \leq c < \frac{N}{2}$  for some  $c$ , and obtain  $\mathcal{R}$  encryptions of  $\mathbf{u}^{(i)}$  which has a constant term of  $\Delta \cdot u_i$ . Finally, we repack our  $\mathcal{R}$  encryptions of  $\mathbf{u}^{(i)}$  into a single  $\mathcal{R}$  encryption of  $\Delta \cdot \mathbf{u}$ . The flow of the proposed ScaledMod procedure is shown in Fig. 1.

*Step 1. Extraction of LWE*

We start with a pair  $(\mathbf{a}, \mathbf{b}) = \mathcal{R}_{2N,s}^0(\mathbf{u})$ . Since the error is zero, we have  $\mathbf{s} \cdot \mathbf{a} + \mathbf{b} = \mathbf{u} \pmod{2N}$ . Multiplication of two polynomials  $\mathbf{a}$  and  $\mathbf{s}$  in  $\mathcal{R}_{2N}$  is described as

$$\mathbf{s} \cdot \mathbf{a} = \sum_{i=0}^{N-1} \left( \sum_{j=0}^i s_j a_{i-j} - \sum_{j=i+1}^{N-1} s_j a_{i-j+N} \right) X^i \pmod{2N}.$$

Let  $\vec{s} = (s_0, \dots, s_{N-1})$  be a vector of coefficients of  $\mathbf{s}$ . We can extract  $\mathcal{L}_{2N,s}^0(u_i) = (\vec{a}^{(i)}, b_i)$  for all  $i \in [0, N-1]$  from  $(\mathbf{a}, \mathbf{b}) = \mathcal{R}_{2N,s}^0(\mathbf{u})$ , where

$$\vec{a}^{(i)} = (a_i, a_{i-1}, \dots, a_0, -a_{N-1}, -a_{N-2}, \dots, -a_{i+1}).$$

*Step 2. Blind Rotation*

The blind rotation procedure [4], [6], [36] transforms a single LWE ciphertext  $\mathcal{L}_{2N,s}^0(\mathbf{u}) = (\vec{a}, \beta)$  obtained from Extract into  $\mathcal{R}$  encryption of  $\mathbf{f} \cdot X^u$  where  $\mathbf{f} = -\Delta \cdot \sum_{j=-c}^c j \cdot X^j$  for  $\|\mathbf{u}\|_\infty \leq c < \frac{N}{2}$ . The result of the blind rotation is

$$\mathcal{R}(\mathbf{f} \cdot X^{\beta + \alpha_0 s_0 + \dots + \alpha_{N-1} s_{N-1}}) = \mathcal{R}(\mathbf{f} \cdot X^u) = \mathcal{R}(\mathbf{u}_\mathbf{f}).$$

Due to the initial function  $f$  and the boundary of  $\|u\|_\infty$ , the polynomial  $u_f$  has  $\Delta \cdot u$  as its constant term. We apply the blind rotation to each  $\mathcal{L}_{2N,s}^0(u_i)$  for all  $i \in [0, N-1]$  and obtain

$$\mathcal{R}_{Q,s}(f \cdot X^{u_i}) := \mathcal{R}_{Q,s}(u^{(i)}) := (a_i, b_i) \in \mathcal{R}_Q^2$$

such that

$$\begin{aligned} a_i \cdot s + b_i &= u^{(i)} + e_{br} \\ &= \Delta \cdot u_i + * \cdot X + \dots + * \cdot X^{N-1} + e_{br} \pmod{Q}, \end{aligned}$$

where  $*$  denotes some value in  $\mathbb{Z}_Q$ . As  $c < \frac{N}{2}$ , we have

$$\begin{aligned} u^{(i)} &= \Delta \cdot u_i + * \cdot X + \dots + * \cdot X^{2c} \\ &\quad + 0 \cdot X^{2c+1} + \dots + 0 \cdot X^{N-2c-2} \\ &\quad + * \cdot X^{N-2c-1} + \dots + * \cdot X^{N-1}. \end{aligned}$$

*Remark 2:* It is worth noting that all errors in our approach are additive which means that the error grows linearly, so we do not have to control the error with rescaling every time as in usual key-switching [12], [37]. Instead, we can postpone rescaling to the end to reduce the complexity.

### Step 3. Repacking

After BLINDROTATE, we receive  $N$  of  $\mathcal{R}$  ciphertexts which encrypt polynomials  $u^{(i)}$  introduced earlier. Only constant coefficients of the encrypted polynomials contain useful information. Thus, other coefficients of these polynomials must be removed. The goal of the Repack procedure is to combine all constant coefficients of encryptions of  $u^{(i)}$  into a single encrypted polynomial without decryption.

Let  $n = n_c$  be the smallest power of two satisfying  $n > 2c$  for some  $c$  defined in the initial function  $f$ . Given  $\mathcal{R}_{Q,s}(u^{(i)})$  for  $i \in [0, N-1]$ , the Repack algorithm is performed in the following two steps. First, we consider a subset of the given ciphertexts as  $\{\mathcal{R}_{Q,s}(u^{(nk)})\}_{k \in [0, \frac{N}{n}-1]}$ . We pack these ciphertexts as

$$\sum_{k=0}^{\frac{N}{n}-1} \mathcal{R}_{Q,s}(u^{(nk)}) \cdot X^{nk} = \mathcal{R}_{Q,s} \left( \sum_{k=0}^{\frac{N}{n}-1} u^{(nk)} \cdot X^{nk} \right).$$

Let  $u^{(0,n)} := \sum_{k=0}^{\frac{N}{n}-1} u^{(nk)} \cdot X^{nk}$ . Since  $n > 2c$ ,  $u^{(0,n)}$  has coefficients  $\Delta \cdot u_{nk}$  at  $X^{nk}$  as

$$\begin{aligned} u^{(0,n)} &= \Delta \cdot u_0 + * \cdot X + \dots + * \cdot X^{n-1} \\ &\quad + \Delta \cdot u_n X^n + \dots + \Delta \cdot u_{2n} X^{2n} + \dots + * \cdot X^{N-1} \end{aligned}$$

where  $*$  denotes some value in  $\mathbb{Z}_Q$ . In a similar way, we pack subsets  $\{\mathcal{R}_{Q,s}(u^{(i+nk)})\}_{k \in [0, \frac{N}{n}-1]}$  into  $\mathcal{R}_{Q,s}(u^{(i,n)})$  for all  $i \in [1, n-1]$  where  $u^{(i,n)}$  has coefficients  $\Delta \cdot u_{i+nk}$  at  $X^{nk}$ .

Second, we adapt the repacking technique from [25]. We consider a pair  $\mathcal{R}_{Q,s}(u^{(0,n)})$  and  $\mathcal{R}_{Q,s}(u^{(\frac{n}{2},n)})$ . Notice that the automorphism  $\psi_{1+\frac{2N}{n}}$  applied to  $u^{(0,n)}$  preserves all coefficients at  $X^{nk}$ , for  $k \in [0, \frac{N}{n}-1]$ , changes the sign of coefficients at  $X^{nk+\frac{n}{2}}$ , and shuffles the other coefficients with possible changes in sign. We only focus on the coefficients of  $X^{nk}$  and  $X^{nk+\frac{n}{2}}$  and do not track how this automorphism

### Algorithm 7 Repacking

---

**Require:**  $\{\mathcal{R}_{Q,s}(u^{(i)})\}_{i \in [0, N-1]}$ ,  $n$   
**Ensure:**  $\mathcal{R}_{Q,s}(n\Delta u)$

**procedure** REPACK  
  **for**  $(i = 0; i < n; i = i + 1)$  **do**  
     $ct^{(i,n)} \leftarrow \mathcal{R}_{Q,s}(u^{(i)})$   
    **for**  $(j = 1; j < \frac{N}{n}; j = j + 1)$  **do**  
       $ct^{(i,n)} \leftarrow ct^{(i,n)} + X^{nj} \cdot ct^{(i+nj)}$   
    **for**  $(k = n; k > 1; k = \frac{k}{2})$  **do**  
      **for**  $(i = 0; i < \frac{k}{2}; i = i + 1)$  **do**  
         $ct^{(i, \frac{k}{2})} \leftarrow ct^{(i,k)} + X^{\frac{k}{2}} \cdot ct^{(i+\frac{k}{2},k)}$   
         $ct^{\text{rot}} \leftarrow ct^{(i,k)} - X^{\frac{k}{2}} \cdot ct^{(i+\frac{k}{2},k)}$   
         $ct^{\text{rot}} \leftarrow \text{EVALAUTO}(ct^{\text{rot}}, 1 + \frac{2N}{k})$   
         $ct^{(i, \frac{k}{2})} \leftarrow ct^{(i, \frac{k}{2})} + ct^{\text{rot}}$   
  **return**  $ct^{(0,1)}$

---

### Algorithm 8 Zeroizing coefficients

---

**procedure** ZEROIZECOEFFS( $ct', n$ )  
   $ct'' \leftarrow ct' \pmod{Q}$   
  **for**  $(k = N; k > 2n; k = k/2)$  **do**  
     $ct''' \leftarrow \text{EVALAUTO}(ct'', k + 1)$   
     $ct'' \leftarrow ct''' + ct'' \pmod{Q}$   
   $ct'' \leftarrow \text{RESCALE}(ct'', \frac{N}{2n})$   
  **return**  $ct''$

---

operates on the other coefficients. We can merge  $\mathcal{R}_{Q,s}(u^{(0,n)})$  and  $\mathcal{R}_{Q,s}(u^{(\frac{n}{2},n)})$  as

$$\begin{aligned} \mathcal{R}(2u^{(0, \frac{n}{2})}) &= \mathcal{R}(u^{(0,n)}) + X^{\frac{n}{2}} \cdot \mathcal{R}(u^{(\frac{n}{2},n)}) \\ &\quad + \text{EVALAUTO} \left( \mathcal{R}(u^{(0,n)}) - X^{\frac{n}{2}} \cdot \mathcal{R}(u^{(\frac{n}{2},n)}), 1 + \frac{2N}{n} \right), \end{aligned}$$

where  $u^{(0, \frac{n}{2})}$  is a polynomial which has coefficients  $\Delta u_{\frac{nk}{2}}$  at  $X^{\frac{nk}{2}}$ . We apply the same procedure to pairs  $\mathcal{R}_{Q,s}(u^{(i,n)})$  and  $\mathcal{R}_{Q,s}(u^{(i+\frac{n}{2},n)})$  for all  $i \in [1, \frac{n}{2}-1]$  and obtain  $\mathcal{R}_{Q,s}(2u^{(i, \frac{n}{2})})$ . We continue this merging process until we get

$$\mathcal{R}_{Q,s}(n \cdot u^{(0,1)}) = \mathcal{R}_{Q,s}(n\Delta \cdot u).$$

The full Repack algorithm is described in Algorithm 7.

*Remark 3:* We obtain  $\mathcal{R}_{Q,s}(n\Delta \cdot u)$  instead of  $\mathcal{R}_{Q,s}(\Delta \cdot u)$  after the ScaledMod procedure and errors are accumulated during the blind rotation and repacking procedures. By modifying the initial state, we can address the first issue. We start with  $[n]_Q^{-1} \cdot \Delta$  instead of  $\Delta$  when  $Q$  and  $n$  are coprime. When  $Q$  is a power of two, we start with  $\mathcal{R}$  modulus  $Qn$  and then rescale by  $n$ . For the second issue we use an auxiliary modulus  $P > \text{Bnd}_{\text{sm}}$  and do all the computations modulo  $QP$  instead of  $Q$  and rescale the result by  $P$  at the end. To do that, we also start with  $\Delta P$  instead of  $\Delta$ . Finally we obtain  $\mathcal{R}_{Q,s}(\Delta \cdot u)$  with only rescaling error  $e_{\text{sm}} = e_{\text{rs}}$ .

### A. Sparsely Packed Ciphertext

We can reduce the complexity of ScaledMod procedure for sparsely packed plaintext encoding. The main idea is to reduce



TABLE I  
PARAMETER SETS FOR BGV/BFV/CKKS SCHEMES WITH THE PROPOSED  
BOOTSTRAPPING;  $t$  DENOTES THE PLAINTEXT MODULUS FOR BGV/BFV AND  $\Delta$   
DENOTES THE SCALING FACTOR FOR CKKS.  $B$  DENOTES DECOMPOSITION BASE IN  
BLIND ROTATION AND KEY SWITCHING PROCEDURES

	$N$	$\log QP$	$\log Q$	$\log q$	$\log B$	$t$	$\Delta$
BGV/BFV-12	$2^{12}$	110	76	42	16	$2^{16} + 1$	-
CKKS-12						-	$2^{36}$
BGV/BFV-13	$2^{13}$	219	170	90	16	$2^{30} - 2^{18} + 1$	-
CKKS-13						-	$2^{80}$

the number of coefficients of  $\mathbf{u}$  which will be inputs of the blind rotations in *ScaledMod* procedure.

We first consider case of CKKS encoding. Let  $\text{ct}$  be an encryption of a plaintext  $\mathbf{m}$  which encodes  $n = n_s$  values where  $n \leq \frac{N}{2}$ . We firstly prepare  $\text{ct}'$  and  $\text{ct}_{\text{init}}$  as previously and execute an additional processing for  $\text{ct}'$  to obtain  $\text{ct}''$ . Then we apply a variant of *ScaledMod* to  $\text{ct}_{\text{init}}$  and combine it to  $\text{ct}''$ .

The additional processing for  $\text{ct}'$  is zeroizing certain coefficients of  $\mathbf{u}$ . We take a similar approach used in the original CKKS bootstrapping [16] which is presented in Algorithm 8. It increases the modulus of  $\text{ct}'$  from  $q'$  to  $Q$  and then applies automorphisms and additions to  $\text{ct}'$ . After the zeroizing procedure, we obtain a ciphertext  $\text{ct}''$  which is described as

$$\text{ct}''(s) = \mathbf{a}'' \cdot \mathbf{s} + \mathbf{b}'' = \mathbf{m} + \mathbf{e}'' + q'\mathbf{u}' \pmod{Q'},$$

where  $Q' = Q \cdot \frac{2n}{N}$  and  $\mathbf{u}'$  has same coefficients as  $\mathbf{u}$  at degrees which are multiples of  $\frac{N}{2n}$  and zero coefficients at other degrees. Since  $\mathbf{m}$  is a sparsely packed plaintext, the message in each slot does not change under the automorphisms used in *ZeroizeCoeffs*.

Due to the structure of  $\mathbf{u}'$ , given  $\text{ct}_{\text{init}} = \mathcal{R}_{2N,s}^0(-\mathbf{u})$  as an input of *ScaledMod*, we can evaluate the blind rotations only for the subset of coefficients  $\{u_{i \cdot \frac{N}{2n}}\}$  for  $i \in [0, 2n - 1]$ , instead of evaluating for every coefficient of  $\mathbf{u}$ . It reduces the number of blind rotations to  $2n$  and also reduces the number of iterations of the *Repack* algorithm. The output of the variant *ScaledMod* is  $\text{ct}'_{\text{sm}}$  which satisfies

$$\text{ct}'_{\text{sm}}(s) = \mathbf{a}'_{\text{sm}} \cdot \mathbf{s} + \mathbf{b}'_{\text{sm}} = -q'\mathbf{u}' + \mathbf{e}'_{\text{sm}} \pmod{Q'}$$

and we combine it with  $\text{ct}''$  to have  $\text{ct}_{\text{boot}} = \text{ct}'' + \text{ct}'_{\text{sm}} \pmod{Q'}$  which satisfies

$$\begin{aligned} \text{ct}_{\text{boot}}(s) &= \mathbf{a}_{\text{boot}} \cdot \mathbf{s} + \mathbf{b}_{\text{boot}} \\ &= \mathbf{m} + \mathbf{e}'' + q'\mathbf{u}' - q'\mathbf{u}' + \mathbf{e}'_{\text{sm}} \\ &= \mathbf{m} + \mathbf{e}'' + \mathbf{e}'_{\text{sm}} \pmod{Q'}. \end{aligned}$$

For the BGV/BFV case, the same idea can be applied, if  $2N|t - 1$ , i.e. if  $X^N + 1$  splits modulo plaintext modulus  $t$ . The case when  $2N$  does not divide  $t - 1$  is more involved, and we do not discuss it here. For more information refer to [38].

## V. EXPERIMENTAL RESULTS

### A. Parameters Selection

All parameter sets for our experiments are chosen to achieve 128 bits of security [39]. Following [39] we use discrete

Gaussian error distribution with standard deviation  $\sigma = 3.2$ . All existing efficient implementations of blind rotation targeting FHEW and TFHE schemes consider parameters  $N \leq 2^{11}$  and  $\log Q \leq 54$ , which allows to use standard 64 bit arithmetic. In contrast BGV/BFV/CKKS schemes require  $N \geq 2^{12}$  to allow homomorphic computations, which results with a necessity to support computations with  $\log Q$  up to 109 (up to 218 for  $N = 2^{13}$ ) [39]. To fit in 64 bit arithmetic, we applied the RNS technique to our blind rotation. For efficiency, we have chosen Lee et al.'s blind rotation technique [6]. To improve the performance of the proposed bootstrapping, we also adapt an algorithm of ring dimension reduction, introduced in [40] during our bootstrapping algorithm.

As the bootstrapping procedures for all three BGV, BFV, CKKS schemes are similar, we have chosen similar parameter sets for all three schemes. The only difference is that for CKKS we specify the scaling factor  $\Delta$ , and for BGV/BFV we specify the plaintext modulus  $t$ . We have chosen a single parameter set for each  $N = 2^{12}$  and  $N = 2^{13}$ . For each parameter set we chose parameters to have a single multiplicative level. The auxiliary modulus  $P$  is introduced as in the remark 3. The parameter sets for our experiments are summarized in Table I.

### B. Implementation

The implementation is done using C++, NTL library [41] for multiprecision integer and floating-point arithmetic and Intel HEXL library [42] for Number Theoretic Transform (NTT) operations. All experiments are run on Intel Xeon Gold 8280 CPU 2.70GHz, single thread and Ubuntu 18.04. Our implementation results show that the new bootstrapping approach is fully generalizable; most of the code is shared across BGV/BFV/CKKS bootstrapping implementations. The experimental results are given in Table II.

The variance of the bootstrapping error is presented in Table II, and it shows that the bootstrapping error is comparable to the rescaling error even for small parameters, as the rescaling error variance is  $\frac{1}{12}(N + 1)$ .

Our implementation does not benefit from SIMD because we use blind rotation for *ScaledMod*. The experiment results confirm that the bottleneck in the computation is the blind rotation algorithm, and the bootstrapping time per slot is independent of the number of slots as well as the scheme. Thus, our bootstrapping has lower throughput compared to the conventional amortized bootstrapping. Besides, due to the complexity of blind rotation, the runtime of our technique increases significantly with a large  $N$  unlike more conventional



TABLE II  
THE PERFORMANCE OF RUNTIME AND ERROR.  $Var[e_{boot}]$  DENOTES VARIANCE OF  
BOOTSTRAPPING ERROR

Parm. Set	# slots	HomRound	ScaledMod	Combine	Tot. time per slot	$Var[e_{boot}]$
BGV/BFV-12	2	0.004 s	4.601 s	0.000 s	2.302 s	244.09
CKKS-12	1				4.605 s	
BGV/BFV-12	8	0.004 s	18.127 s	0.000 s	2.266 s	304.76
CKKS-12	4				4.533 s	
BGV/BFV-13	4	0.011 s	23.630 s	0.001 s	11.821 s	463.61
CKKS-13	2				23.642 s	

approaches to bootstrapping algorithms, which are impossible for small  $N$ . For instance, state-of-the-art BFV bootstrapping [15] varies from 6.75 seconds with  $N = 2^{14}$  to 1381 seconds with  $N = 2^{15}$  (with multiple slots in both cases). Meanwhile, bootstrapping for the CKKS scheme takes 18.1 seconds for  $N = 2^{15}$  [20].

The new bootstrapping avoids homomorphic evaluation of a high-degree polynomial so that it is working with much smaller parameters. For example, the ciphertext size is **370** times smaller compared to the latest work [20] (for  $\sim 30$ -bit precision). Thus, for a small number of slots, our bootstrapping is fast and enjoys low latency.

There are various factors and settings that differ our technique with previous techniques based on polynomial approximations. Consequently we decide not to compare our technique with previous methods directly because it is challenging to make a fair comparison.

## VI. CONCLUSION

Our general bootstrapping technique deviates from the conventional bootstrapping approaches and advances research in this field in a new direction. With this new structure of bootstrapping we have achieved the goal of making bootstrapping general and more flexible. Our current version uses blind rotation as a core function to calculate ScaledMod. With this, our bootstrapping for RLWE-based schemes works even for relatively small rings (with  $N$  equals to  $2^{12}$  or  $2^{13}$ ), which is impossible with conventional bootstrapping based on linear transformations and polynomial evaluation. Although the ring parameters and thus the size of the ciphertext and secret key are reduced, our bootstrapping enjoys high precision comparable to rescaling error instead of approximation error as in previous CKKS bootstrapping methods, and arbitrary plaintext modulus for BGV/BFV schemes where the choice of plaintext modulus suitable for conventional bootstrapping has been restricted.

The flexibility of our bootstrapping opens up new directions for building schemes which could not be achievable with conventional versions. By using method from [6] to construct ScaledMod operation, our bootstrapping supports not only binary and ternary secret key distributions but also larger secret key, making the technique suitable for a wider range of potential applications of the RLWE-based schemes.

A central goal of future work is concentrated on the optimization of the proposed bootstrapping method. An interesting direction is to adapt recent promising results on amortized FHEW bootstrapping [30] to ScaledMod. At the same time, we see the potential of our bootstrapping in the development of efficient bootstrapping hardware for RLWE-based schemes. While conventional bootstrapping requires hundreds to thousands of bits of modulus, the current construction of our bootstrapping uses only 100-200 bit moduli which makes it easier to adapt to hardware. FHEW-type schemes also have a higher potential for reducing the bootstrapping keys size as demonstrated in [43] than conventional bootstrapping techniques.

## ACKNOWLEDGMENT

The authors thank Yuriy Polyakov and Daniele Micciancio for their careful review, feedback, and insightful discussions that helped them to improve the article.

## REFERENCES

- [1] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 1–40, 2009.
- [2] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Berlin, Germany: Springer-Verlag, 2010, pp. 1–23.
- [3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2009, pp. 169–178.
- [4] L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Berlin, Germany: Springer-Verlag, 2015, pp. 617–640.
- [5] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, 2020.
- [6] 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 threshold homomorphic encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Cham, Germany: Springer-Verlag, 2023, pp. 227–256.
- [7] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, Jul. 2014.
- [8] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. Annu. Cryptol. Conf.*, Berlin, Germany: Springer-Verlag, 2012, pp. 868–886.
- [9] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012. [Online]. Available: <https://eprint.iacr.org/2012/144>
- [10] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Cham, Germany: Springer-Verlag, 2017, pp. 409–437.
- [11] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. Int. Conf. Sel. Areas Cryptogr.*, Berlin, Germany: Springer-Verlag, 2018, pp. 347–368.

- [12] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca, "A full RNS variant of FV like somewhat homomorphic encryption schemes," in *Proc. Int. Conf. Sel. Areas Cryptogr.*, Cham, Germany: Springer-Verlag, 2016, pp. 423–442.
- [13] C. Gentry, S. Halevi, and N. P. Smart, "Better bootstrapping in fully homomorphic encryption," in *Proc. Int. Workshop Public Key Cryptogr.*, Berlin, Germany: Springer-Verlag, 2012, pp. 1–16.
- [14] S. Halevi and V. Shoup, "Bootstrapping for HELib," *J. Cryptol.*, vol. 34, no. 1, pp. 1–44, 2021.
- [15] H. Chen and K. Han, "Homomorphic lower digits removal and improved FHE bootstrapping," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Cham, Germany: Springer-Verlag, 2018, pp. 315–337.
- [16] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Cham, Germany: Springer-Verlag, 2018, pp. 360–384.
- [17] H. Chen, I. Chillotti, and Y. Song, "Improved bootstrapping for approximate homomorphic encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Cham, Germany: Springer-Verlag, 2019, pp. 34–54.
- [18] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," in *Proc. Cryptographers' Track RSA Conf.*, Cham, Germany: Springer-Verlag, 2020, pp. 364–390.
- [19] Y. Lee, J.-W. Lee, Y.-S. Kim, and J.-S. No, "Near-optimal polynomial for modulus reduction using L2-norm for approximate homomorphic encryption," *IEEE Access*, vol. 8, pp. 144 321–144 330, Aug. 2020.
- [20] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux, "Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Cham, Germany: Springer-Verlag, 2021, pp. 587–617.
- [21] J.-W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No, "High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Cham, Germany: Springer-Verlag, 2021, pp. 618–647.
- [22] Y. Lee, J.-W. Lee, Y.-S. Kim, J.-S. No, and H. Kang, "High-precision bootstrapping for approximate homomorphic encryption by error variance minimization," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Cham, Germany: Springer-Verlag, 2022, pp. 551–580.
- [23] Y. Bae, J. H. Cheon, W. Cho, J. Kim, and T. Kim, "META-BTS: Bootstrapping precision beyond the limit," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 223–234.
- [24] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Cham, Germany: Springer-Verlag, 2021, pp. 648–677.
- [25] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (ring) LWE ciphertexts," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, Cham, Germany: Springer-Verlag, 2021, pp. 460–479.
- [26] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Annu. Cryptol. Conf.*, Berlin, Germany: Springer-Verlag, 2013, pp. 75–92.
- [27] C. Boura, N. Gama, M. Georgieva, and D. Jetchev, "CHIMERA: Combining ring-LWE-based fully homomorphic encryption schemes," *J. Math. Cryptol.*, vol. 14, no. 1, pp. 316–338, 2020.
- [28] W.-j. Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu, "PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption," in *Proc. IEEE Symp. Secur. Privacy (SP)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 1057–1073.
- [29] Z. Liu, D. Micciancio, and Y. Polyakov, "Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Berlin, Germany: Springer-Verlag, 2022, pp. 130–160.
- [30] Z. Liu and Y. Wang, "Amortized functional bootstrapping in less than 7ms, with  $\tilde{O}(1)$  polynomial multiplications," *Cryptology ePrint Archive*, Paper 2023/910, 2023. [Online]. Available: <https://eprint.iacr.org/2023/910>
- [31] N. Genise, D. Micciancio, and Y. Polyakov, "Building an efficient lattice gadget toolkit: Subgaussian sampling and more," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Cham, Germany: Springer-Verlag, 2019, pp. 655–684.
- [32] D. Micciancio and Y. Polyakov, "Bootstrapping in FHEW-like cryptosystems," in *Proc. 9th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, 2021, pp. 17–28.
- [33] A. Kim, Y. Polyakov, and V. Zucca, "Revisiting homomorphic encryption schemes for finite fields," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Cham, Germany: Springer-Verlag, 2021, pp. 608–639.
- [34] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Proc. Annu. Cryptol. Conf.*, Berlin, Germany: Springer-Verlag, 2011, pp. 505–524.
- [35] S. Halevi, Y. Polyakov, and V. Shoup, "An improved RNS variant of the BFV homomorphic encryption scheme," in *Proc. Cryptographers' Track RSA Conf.*, Cham, Germany: Springer-Verlag, 2019, pp. 83–105.
- [36] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Cham, Germany: Springer-Verlag, 2017, pp. 377–408.
- [37] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM J. Comput.*, vol. 43, no. 2, pp. 831–871, 2014.
- [38] R. Geelen and F. Vercauteren, "Bootstrapping for BGV and BFV revisited," *Cryptology ePrint Archive*, 2022. [Online]. Available: <https://eprint.iacr.org/2022/1363>
- [39] M. Albrecht et al., "Homomorphic encryption standard," in *Protecting Privacy Through Homomorphic Encryption*. Cham, Germany: Springer-Verlag, 2021, pp. 31–62.
- [40] C. Gentry, S. Halevi, C. Peikert, and N. P. Smart, "Field switching in BGV-style homomorphic encryption," *J. Comput. Secur.*, vol. 21, no. 5, pp. 663–684, 2013.
- [41] V. Shoup et al., "NTL: A library for doing number theory." GitHub. Accessed: Jun. 24, 2021. [Online]. Available: <https://github.com/libnt/ntl>
- [42] F. Boemer, S. Kim, G. Seifu, F. DM de Souza, and V. Gopal, "Intel HEXL: Accelerating homomorphic encryption with intel AVX512-IFMA52," in *Proc. 9th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, 2021, pp. 57–62.
- [43] A. Kim, Y. Lee, M. Deryabin, J. Eom, and R. Choi, "LFHE: Fully homomorphic encryption with bootstrapping key size less than a megabyte," *Cryptology ePrint Archive*, Paper 2023/767, 2023. [Online]. Available: <https://eprint.iacr.org/2023/767>



**Andrej Kim** received the B.S. and M.S. degrees in mathematics from Lomonosov Moscow State University, Moscow, Russia, in 2011 and the Ph.D. degree in mathematical sciences from Seoul National University, Seoul, South Korea, in 2019. He was a Postdoctoral Researcher with the New Jersey Institute of Technology, Newark, NJ, USA, in 2020. He has been with the Samsung Advanced Institute of Technology since 2020.



**Maxim Deryabin** received the B.S. and M.S. degrees in mathematics and the Ph.D. degree in computer science from North Caucasus Federal University, Stavropol, Russia, in 2011, 2013, and 2016, respectively. He held various research and teaching positions at North Caucasus Federal University, from 2013 to 2020. Currently, he is a Staff Researcher with the Samsung Advanced Institute of Technology, Suwon, South Korea. His current research interests include lattice-based cryptography, homomorphic encryption, and computational algebra.



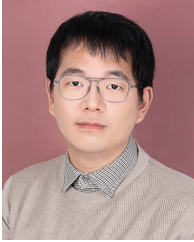
**Jieun Eom** received the B.S. degree in mathematics and the M.S. and Ph.D. degrees in information security from Korea University, Seoul, in 2010, 2012, and 2019, respectively. She is currently working as a Staff Researcher with the Samsung Advanced Institute of Technology. Her research interests include cryptography, information security, and homomorphic encryption.



**Rakyong Choi** received the B.S. and M.S. degrees in mathematical sciences and the Ph.D. degree from the School of Computing, KAIST, Daejeon, Korea, in 2011, 2013, and 2019, respectively. He is currently working as a Staff Researcher with the Samsung Advanced Institute of Technology. His research interests include post quantum cryptography, homomorphic encryption, and privacy preserving machine learning.



**Yongwoo Lee** received the B.S. degree in electrical engineering and computer science from Gwangju Institute of Science and Technology, Korea, in 2015, and the M.S. and Ph.D. degrees in electrical and computer engineering from Seoul National University, Korea, in 2017 and 2021, respectively. Before joining Inha University as an Associate Professor, he served as a Staff Researcher with the Samsung Advanced Institute of Technology. His primary research interest lies in privacy-enhancing technologies.



**Whan Ghang** received the B.S. degree in mathematics from Massachusetts Institute of Technology, Cambridge, MA, USA, and the Ph.D. degree in mathematics from Harvard University, Cambridge, MA, USA, in 2013 and 2019, respectively. He has been with the Samsung Advanced Institute of Technology since 2020.



**Donghoon Yoo** received the M.S. and Ph.D. degrees in information engineering from GIST, in 1999 and 2005, respectively. He is currently the Chief Research Officer and the Director of the Research Team with Desilo Inc., South Korea. Before joining Desilo, he worked as the Research Master and the Project Leader of the Privacy Preserving Computing team with the Samsung Advanced Institute of Technology, South Korea. His current research interest is privacy-enhancing technologies, including homomorphic encryption, multiparty computation, differential privacy, trusted execution environment, their applications, and hardware accelerations.