General Bootstrapping Approach for RLWE-based Homomorphic Encryption

Andrey Kim[1], Maxim Deryabin[1], Jieun Eom[1], Rakyong Choi[1], Yongwoo Lee[1], Whan Ghang[1], and Donghoon Yoo[1]

[1] Samsung Advanced Institute of Technology, Suwon, Republic of Korea
{andrey.kim, max.deriabin, jieun.eom, rakyong.choi, yw0803.lee, whan.ghang, say.yoo}@samsung.com

September 27, 2021

Abstract

We propose a new bootstrapping approach that works for all three Brakerski-Gentry-Vaikuntanathan(BGV), Brakerski/Fan-Vercauteren (BFV), and Cheon-Kim-Kim-Song (CKKS) schemes. This approach adopts a blind rotation technique from FHEW-type schemes. For BGV and BFV, our bootstrapping does not have any restrictions on plaintext modulus unlike typical cases of the previous methods. For CKKS, our approach introduces an error comparable to a rescaling error which enables more than 70 bits of precision after bootstrapping while consuming only 1-2 levels. Due to the high precision of the proposed bootstrapping algorithm, it is the first bootstrapping resistant to the security vulnerability of CKKS found by Li and Micciancio (Eurocrypt 2021). In addition, we introduce methods to reduce the size of public keys required for blind rotations generated by a secret key holder.

Keywords: Bootstrapping, Fully Homomorphic Encryption.

Contents

1 Introduction

Homomorphic encryption (HE) is a form of encryption that enables computations on encrypted data without access to a secret key. Most HE schemes rely on Learning with Errors (LWE) [1] or Ring Learning with Errors (RLWE) [2] problem, and their ciphertexts contain small "noise" which ensures security. However, the noise grows during computations and eventually can destroy the message, and thus the number of operations which can be computed in encrypted form is limited. Since the first construction of fully homomorphic encryption (FHE) scheme by Gentry [3], significant progress has been made in the direction of research on HE. Gentry's celebrated idea of bootstrapping allows refreshing the noise of a ciphertext and more computations to be performed on the ciphertext.

The most common FHE schemes can be categorized into FHEW-type, BGV/BFV-type and CKKS-type. FHEW-type schemes (such as FHEW [4] and TFHE schemes [5]) are based on LWE and primarily work with boolean circuits. The core idea of their bootstrapping procedure is a so-called blind rotation technique [5, 6, 7]. BGV/BFV-type schemes [8, 9, 10] are commonly designed for computations over finite rings, and CKKS-type schemes [11, 12] are designed for computations over real and complex numbers. While BGV, BFV, and CKKS are all based on RLWE and their encryption differs only in encoding, the existing bootstrapping algorithms for these three schemes are different.

Both BGV and BFV are exact schemes and an error accumulated during computations can eventually destroy the message. All known bootstrapping techniques for BGV and BFV schemes and their RNS variants [13, 14, 15] are performed by extracting digits from a decrypted result homomorphically. However, the complexity of digit extraction increases with the size of plaintext modulus and thus significant limitations should be applied to the plaintext modulus to make bootstrapping secure and practical.

Meanwhile, an error is considered as a part of the message in CKKS so that the bit extraction technique cannot be applied. Cheon et al. [16] proposed the first bootstrapping procedure for CKKS based on the polynomial approximation to a modular reduction function in a decryption algorithm. Subsequent studies have focused on approximating the modular reduction function more precisely to improve accuracy [17, 18, 19, 20, 21, 22].

However, such an approximation approach produces additional errors that have a significant impact on the quality of a message. Besides, to achieve a better quality of approximations, the previous methods consume a huge number of multiplicative levels and it causes the modulus of a ciphertext after bootstrapping to be much smaller than that of the fresh ciphertext. It leads to the fact that large RLWE parameters are required to make this bootstrapping procedure feasible. In practice, the ring dimension of RLWE is set to 2^{16} or higher for security purpose [16, 18, 20, 23].

Moreover, Li and Micciancio [24] recently discovered a vulnerability of CKKS scheme based on possible leaks of information about secret keys through noise analysis of the decrypted message. Their attack shows that the traditional formulation of IND-CPA security (or indistinguishability under chosen plaintext attacks) achieved by CKKS does not adequately capture the security of CKKS against passive adversaries, and that a different and stronger definition of IND-CPA$^{\mathsf{D}}$ (or indistinguishability under chosen plaintext attacks with decryption oracles) is required to evaluate the security of CKKS. The only known workarounds such as noise flooding techniques [3, 24] in the decryption state, reduce the precision of the message. In practice, the precision of the message is reduced by about 30 bits after the workaround. This arises security concerns of existing bootstrapping methods. Previous bootstrapping approaches preserve only about 25-35 bits of

precision of a message [18, 20]. More levels and large scaling factors are required to guarantee additional 30 bits precision and is hardly possible in practice to resist the vulnerability in [24].

In this paper, we propose a new bootstrapping technique that does not use polynomial approximation to evaluate the modular reduction function. We combine the blind rotation technique used in bootstrapping for FHEW-type schemes [4, 5] and the RLWE ciphertext repacking technique [25] to avoid limitations of existing techniques. It produces an error as small as a rescaling error and consumes only 1-2 levels. Compared to previous CKKS bootstrapping techniques, our technique almost does not lose the precision of the message and is suitable for employing noise flooding workarounds to resist the vulnerability in [24]. We implement our bootstrapping technique using RLWE ring dimension of 2^{13}, while preserving more than 70 bits of precision.

We show that the new bootstrapping technique is also applicable to BGV and BFV schemes with only minor modifications on the proposed algorithm. Unlike the previous bootstrapping, our technique does not have restrictions on the plaintext modulus and allows to perform bootstrapping for smaller ring dimension. This is possible because all the procedures in our bootstrapping technique are almost independent of message size and plaintext space.

We also provide a compact representation technique for reducing the size of public keys. It enables a secret key holder to generate a smaller size of public keys. This brings the effect of reducing the communication overhead. A computational party should reconstruct the public keys for performing homomorphic operations. We present the additional technique to reconstruct the public keys and perform blind rotations on-the-fly without storing all of them.

1.1 Technical Overview

Let N be the ring dimension and q be the ciphertext modulus. $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$ denotes the $2N$-th cyclotomic ring and its quotient ring is denoted by $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$. Given an RLWE ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$, the decryption query is defined as

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

where $\boldsymbol{s}$ is a secret key and $\boldsymbol{e}$ is a small error. When the modulus q is small and the additional homomorphic operations can destroy the message $\boldsymbol{m}$, it is required to increase this small modulus to a bigger modulus $Q > q$ and produce a new ciphertext $\mathtt{ct}_{\mathtt{boot}} = (\boldsymbol{a}_{\mathtt{boot}}, \boldsymbol{b}_{\mathtt{boot}}) \in \mathcal{R}_Q^2$ such that

$$\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{boot}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} = \boldsymbol{m} + \boldsymbol{e}_{\mathtt{boot}} \in \mathcal{R}_Q.$$

The previous CKKS bootstrapping methods start from the ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ represented in a higher modulus Q. Then the decryption query becomes

$$\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{b} = \boldsymbol{m} + \boldsymbol{e} + q \cdot \boldsymbol{u} \in \mathcal{R}_Q \tag{1}$$

for some small polynomial $\boldsymbol{u}$. After a homomorphic linear transformation, the ciphertext contains $m_i + e_i + q \cdot u_i$ in each slot, where m_i, e_i and u_i are i-th coefficients of polynomials $\boldsymbol{m}$, $\boldsymbol{e}$ and $\boldsymbol{u}$, respectively. A polynomial approximation of the modular reduction function removes $q \cdot u_i$ parts from the ciphertext. The final ciphertext, which encrypts $\boldsymbol{m}$ in a higher modulus Q' where $q < Q' < Q$, is obtained through another homomorphic linear transformation. As mentioned above, a major difficulty of this method is accurate polynomial approximation of the modular reduction function.

Our bootstrapping technique greatly differs from the previous bootstrapping methods. To remove $q \cdot \boldsymbol{u}$ part from the equation (1), we compute encryption of $-q \cdot \boldsymbol{u}$ by the blind rotation technique used in the FHEW-type bootstrapping algorithm [4, 5]. The main idea of our approach is to preprocess the ciphertext $\mathtt{ct}$ and extract the encryption of a small polynomial $\boldsymbol{u}$ modulo $2N$ to obtain a ciphertext $\mathtt{ct}_{\mathtt{prep}} = \left(\boldsymbol{a}_{\mathtt{prep}}, \boldsymbol{b}_{\mathtt{prep}}\right) \in \mathcal{R}_{2N}^2$ satisfying

$$\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{prep}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{prep}} = -\boldsymbol{u} \in \mathcal{R}_{2N}.$$

Then we apply the blind rotation technique to $\mathtt{ct}_{\mathtt{prep}}$ and repack the results by using the RLWE ciphertext repacking technique [25]. This procedure that we call `ScaledMod` gives an encryption $\mathtt{ct}_{\mathtt{sm}} = (\boldsymbol{a}_{\mathtt{sm}}, \boldsymbol{b}_{\mathtt{sm}}) \in \mathcal{R}_Q^2$ of the scaled value $q \cdot \boldsymbol{u}$ and it satisfies

$$\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{sm}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{sm}} = -q \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \in \mathcal{R}_Q.$$

Finally, we add the ciphertext $\mathtt{ct}$ and $\mathtt{ct}_{\mathtt{sm}}$ modulo Q to eliminate $q \cdot \boldsymbol{u}$ term from $\mathtt{ct}$ and increase the size of the modulus from q to Q. Since the error grows linearly during `ScaledMod` procedure, we can adjust the error $\boldsymbol{e}_{\mathtt{sm}}$ to be comparable to the rescaling error with only a single rescaling.

1.2 Related Works

Ducas and Micciancio [4] introduced a blind rotation technique based on RGSW [26] and achieved bootstrapping time less than a second for evaluation of boolean operations in encrypted form. One of the main advantages of the blind rotation technique is that it introduces only small controllable additive error. Chillotti et al. [5, 27] proposed a TFHE scheme over the torus and several optimization methods for FHEW. Recently, Micciancio and Polyakov [28] generalized it to unify the original and extended variants of both FHEW and TFHE.

Several recent studies [29, 30, 31, 32, 33] have applied the blind rotation technique in conjunction with other FHE 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 sign functions and other neural network activation functions [29, 30]. However, none of them considered applying this technique to bootstrapping of BGV, BFV, and CKKS schemes.

1.3 Organization

This paper is organized as follows. In Section 2, we start with some preliminaries on lattice-based structures and operations with them. In Section 3, we present the core algorithm `ScaledMod` for our bootstrapping. In Section 4, we describe full bootstrapping algorithms for CKKS, BGV and BFV. In Section 5, we present a compact representation of public keys and how to reconstruct them. In Section 6, we provide complexity and error analysis and compare our bootstrapping precision with previous methods. We conclude in Section 7.

2 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. $\boldsymbol{a} = \boldsymbol{a}(X)$. 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 $\boldsymbol{a} = a_0 + a_1 \cdot X + \cdots + a_{N-1} \cdot X^{N-1} \in \mathcal{R}$, we denote the ℓ_∞ norm of $\boldsymbol{a}$ as $\|\boldsymbol{a}\|_\infty = \max_{0 \le i < N} \{|a_i|\}$. There exists a constant $\delta_\mathcal{R}$ such that $\|\boldsymbol{a} \cdot \boldsymbol{b}\|_\infty \le \delta_\mathcal{R} \|\boldsymbol{a}\|_\infty \|\boldsymbol{b}\|_\infty$ for any $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R}$. For $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, we use the bound $\delta_\mathcal{R} = 2\sqrt{N}$ as shown in [34].

We use $\boldsymbol{a} \leftarrow \mathsf{S}$ to denote uniform sampling from the set S. We denote sampling according to a distribution χ by $\boldsymbol{a} \leftarrow \chi$. $\chi_{\mathtt{err}}$ denotes a discrete Gaussian distribution with a standard deviation $\sigma_{\mathtt{err}}$. $\chi_{\mathtt{key}}$ denotes ternary distribution such that each coefficient is chosen from $\{-1, 0, 1\}$ and is used for secret key generation. We do not consider sparse keys as they are a subject of security concerns [35, 36] and are also not supported by the Homomorphic Encryption standardization document [37].

2.1 Basic Lattice-based Encryption

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

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

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

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

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

where $\boldsymbol{a} \leftarrow \mathcal{R}_Q$, and $e_i \leftarrow \chi_{\mathtt{err}}$ for each coefficient e_i of $\boldsymbol{e}$, $i \in [0, N-1]$. As with LWE, we will occasionally drop subscripts Q and $\boldsymbol{s}$. We also use the notation $\mathsf{RLWE}^0_{Q,\boldsymbol{s}}(\boldsymbol{m})$ if the error $\boldsymbol{e}$ is zero. A ciphertext $\mathtt{ct} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{m}) = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_Q^2$ is decrypted by computing

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

We use shorthand notation $\mathtt{ct}(\boldsymbol{s}) := \mathsf{RLWE}^{-1}_{Q,\boldsymbol{s}}(\mathtt{ct})$.

We assume that $(\boldsymbol{t}_0, \cdots, \boldsymbol{t}_{d-1})$ is a gadget decomposition of $\boldsymbol{t} \in \mathcal{R}_Q$ if $\boldsymbol{t} = \sum_{i=0}^{d-1} g_i \cdot \boldsymbol{t}_i$ where $\vec{g} = (g_0, \ldots, g_{d-1})$ is a gadget vector. For a power of two modulus Q, we use a base power gadget vector $(1, B^1, \ldots, B^{d-1})$ with a power of two B. We use the RNS gadget vector $([\hat{q}_j^{-1}]_{q_j} \cdot \hat{q}_j)_{0 \le j < d}$, where $\hat{q}_j = \prod_{i \ne j} q_i$ and the modulus Q is chosen as the product $Q = \prod_{0 \le j < d} q_j$ of different primes.

We adapt the definitions of RLWE' and RGSW from [28]. For a gadget vector $\vec{g}$, we define

$$\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{m}) := (\mathsf{RLWE}_{\boldsymbol{s}}(g_0 \cdot \boldsymbol{m}), \mathsf{RLWE}_{\boldsymbol{s}}(g_1 \cdot \boldsymbol{m}), \cdots, \mathsf{RLWE}_{\boldsymbol{s}}(g_{d-1} \cdot \boldsymbol{m})) \in \mathcal{R}_Q^{2d}$$

and

$$\mathsf{RGSW}_{\boldsymbol{s}}(\boldsymbol{m}) := \big(\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s} \cdot \boldsymbol{m}), \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{m})\big) \in \mathcal{R}_Q^{2 \times 2d}.$$

The scalar multiplication between an element in $\mathcal{R}_Q$ and RLWE' ciphertext is defined as

$$\odot : \mathcal{R}_Q \times \mathsf{RLWE}' \to \mathsf{RLWE}$$

using the following rule

$$\begin{aligned}\boldsymbol{t} \odot \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{m}) &= \langle (\boldsymbol{t}_0, \cdots, \boldsymbol{t}_{d-1}), (\mathsf{RLWE}_{\boldsymbol{s}}(g_0 \cdot \boldsymbol{m}), \cdots, \mathsf{RLWE}_{\boldsymbol{s}}(g_{d-1} \cdot \boldsymbol{m})) \rangle \\ &= \sum_{i=0}^{d-1} \boldsymbol{t}_i \cdot \mathsf{RLWE}_{\boldsymbol{s}}(g_i \cdot \boldsymbol{m}) = \mathsf{RLWE}_{\boldsymbol{s}}\left(\sum_{i=0}^{d-1} g_i \cdot \boldsymbol{t}_i \cdot \boldsymbol{m}\right) \\ &= \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{t} \cdot \boldsymbol{m}) \in \mathcal{R}_Q^2,\end{aligned}$$

where $(\boldsymbol{t}_0, \cdots, \boldsymbol{t}_{d-1})$ is a gadget decomposition of $\boldsymbol{t} \in \mathcal{R}_Q$. For each error $\boldsymbol{e}_i$ in $\mathsf{RLWE}_{\boldsymbol{s}}(g_i \cdot \boldsymbol{m})$, the error after multiplication is equal to $\sum_{i=0}^{d-1} \boldsymbol{t}_i \cdot \boldsymbol{e}_i$ which is small as $\boldsymbol{t}_i$ and $\boldsymbol{e}_i$ are small.

The multiplication between RLWE and RGSW ciphertexts is defined as

$$\circledast : \mathsf{RLWE} \times \mathsf{RGSW} \to \mathsf{RLWE}$$

and specifically,

$$\begin{aligned}\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}_1) \circledast \mathsf{RGSW}_{\boldsymbol{s}}(\boldsymbol{m}_2) &= (\boldsymbol{a}, \boldsymbol{b}) \circledast \left(\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s} \cdot \boldsymbol{m}_2), \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{m}_2)\right) \\ &= \boldsymbol{a} \odot \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s} \cdot \boldsymbol{m}_2) + \boldsymbol{b} \odot \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{m}_2) \\ &= \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{a} \cdot \boldsymbol{s} \cdot \boldsymbol{m}_2) + \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{b} \cdot \boldsymbol{m}_2) \\ &= \mathsf{RLWE}_{\boldsymbol{s}}((\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{b}) \cdot \boldsymbol{m}_2) \\ &= \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \boldsymbol{e}_1 \cdot \boldsymbol{m}_2) \in \mathcal{R}_Q^2.\end{aligned}$$

This result represents an RLWE encryption of the product $\boldsymbol{m}_1 \cdot \boldsymbol{m}_2$ with an additional error term $\boldsymbol{e}_1 \cdot \boldsymbol{m}_2$. In order to have $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}_1) \circledast \mathsf{RGSW}_{\boldsymbol{s}}(\boldsymbol{m}_2) \approx \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2)$, it is necessary to make the error term $\boldsymbol{e}_1 \cdot \boldsymbol{m}_2$ to be small. It can be achieved by using monomials for $\boldsymbol{m}_2$ as $\boldsymbol{m}_2 = \pm X^v$.The multiplication between RLWE $\circledast$ RGSW is naturally extended to $\mathsf{RLWE}' \circledast \mathsf{RGSW}$ by applying RLWE $\circledast$ RGSW to each component of RLWE'. Note that $\mathsf{RGSW}^0(1) := I_2 \otimes \vec{g}$ is a trivial RGSW encryption of 1 under any key $\boldsymbol{s}$, where I_2 is a 2×2 identity matrix and $\otimes$ is a tensor product.

2.2 Key Switching in RLWE

Key switching operation converts a ciphertext $\mathsf{RLWE}_{\boldsymbol{s}_1}(\boldsymbol{m})$ encrypted by a secret key $\boldsymbol{s}_1$ to a ciphertext $\mathsf{RLWE}_{\boldsymbol{s}_2}(\boldsymbol{m})$ encrypted by a new secret key $\boldsymbol{s}_2$. There are different variants of the key switching technique and readers can refer to literature such as [38] for more details. We focus on the BV key switching type [39] and its RNS variants [40] that fit our approach.

- $\texttt{KeySwitchGen}(\boldsymbol{s}_1, \boldsymbol{s}_2)$: Outputs $\mathsf{RLWE}'_{\boldsymbol{s}_2}(\boldsymbol{s}_1)$.
- $\texttt{KeySwitch}_{\boldsymbol{s}_1 \to \boldsymbol{s}_2}(\mathsf{RLWE}_{\boldsymbol{s}_1}(\boldsymbol{m}))$: Given $\mathsf{RLWE}_{\boldsymbol{s}_1}(\boldsymbol{m}) = (\boldsymbol{a}, \boldsymbol{b})$, it evaluates

$$\mathsf{RLWE}_{\boldsymbol{s}_2}(\boldsymbol{m}) = \boldsymbol{a} \odot \mathsf{RLWE}'_{\boldsymbol{s}_2}(\boldsymbol{s}_1) + (0, \boldsymbol{b}) \pmod{Q}.$$

$\mathsf{RLWE}'_{\boldsymbol{s}_2}(\boldsymbol{s}_1)$ generated by `KeySwitchGen` is a public key switching key. The key switching error is equal to the error of $\mathcal{R} \odot \mathsf{RLWE}'$ multiplication.

Remark. *Key switching usually requires another 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.*

2.3 Automorphism in RLWE

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

- $\texttt{EvalAuto}(\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}), t)$: Given $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}(X)) = (\boldsymbol{a}(X), \boldsymbol{b}(X))$, it applies ψ_t to $\boldsymbol{a}(X)$ and $\boldsymbol{b}(X)$ and obtains $(\boldsymbol{a}(X^t), \boldsymbol{b}(X^t))$ which is an RLWE encryption of $\boldsymbol{m}(X^t)$ under the secret key $\boldsymbol{s}(X^t)$. Then it performs key switching from $\boldsymbol{s}(X^t)$ to $\boldsymbol{s}(X)$ and outputs $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}(X^t)) = \mathsf{RLWE}_{\boldsymbol{s}}(\psi_t(\boldsymbol{m}))$.

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

2.4 Rescaling in RLWE

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

- $\texttt{Rescale}(\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{m}), q)$: Given $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{m}) = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_Q^2$, it outputs

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

The rescaling procedure also divides the error of $\texttt{ct}$ by q, but introduces additional rescaling error $\boldsymbol{e}_{\texttt{rs}}$. The rescaling error $\boldsymbol{e}_{\texttt{rs}}$ is small [41] and it's norm $\|\boldsymbol{e}_{\texttt{rs}}\|_\infty$ is bounded by $\frac{1}{2}(1+\delta_{\mathcal{R}})$ for ternary secret key.

2.5 RLWE based Schemes

We briefly present the encryption procedures for the three most common FHE schemes based on RLWE. The main difference of encryption in all these schemes is in the message representation and encoding procedures.

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

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

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

$$\texttt{Enc}_{\text{BFV}}(\boldsymbol{m}) = \left(\boldsymbol{a}, -\boldsymbol{a}\cdot\boldsymbol{s} + \boldsymbol{e} + \left\lfloor \frac{Q}{t}\cdot\boldsymbol{m} \right\rceil\right).$$

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

$$\texttt{Enc}_{\text{CKKS}}(\boldsymbol{m}) = (\boldsymbol{a}, -\boldsymbol{a}\cdot\boldsymbol{s} + \boldsymbol{e} + \boldsymbol{m}).$$

All encryption algorithms described above assume that the message is already encoded into the polynomial $\boldsymbol{m}$. Encoding techniques for BGV and BFV can be found in [8, 9, 10] and for CKKS in [11, 12].

3 Scaled Modulus Raising

In this section, we present the core algorithm `ScaledMod` used in our bootstrapping in Section 4. The algorithm transforms $\mathsf{RLWE}^0_{2N,\boldsymbol{s}}(\boldsymbol{u})$, where $\|\boldsymbol{u}\|_\infty < N/2$, to $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\Delta\cdot\boldsymbol{u})$ for a scaling factor Δ and a large modulus Q.

3.1 `ScaledMod` Procedure

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

- $\mathtt{ScaledMod}(\mathsf{RLWE}^0_{2N,\boldsymbol{s}}(\boldsymbol{u}),\Delta,Q)$: Outputs $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\Delta\cdot\boldsymbol{u})$

$$\mathsf{RLWE}^0_{2N,\boldsymbol{s}}(\boldsymbol{u}) \xrightarrow{\mathtt{Extract}} \{\mathsf{LWE}^0_{2N,\vec{s}}(u_i)\} \xrightarrow{\mathtt{BlindRotate}} \{\mathsf{RLWE}_{Q,\boldsymbol{s}}(\mathtt{f}\cdot X^{u_i})\} \xrightarrow{\mathtt{Repack}} \mathsf{RLWE}_{Q,\boldsymbol{s}}(\Delta\cdot\boldsymbol{u})$$

Now we describe each part of the `ScaledMod` algorithm in detail.

Step 1. Extraction.

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

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

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

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

Step 2. Blind Rotation.

`BlindRotate` procedure transforms a single LWE ciphertext $\mathsf{LWE}^0_{2N,\vec{s}}(u) = (\vec{\alpha},\beta)$ obtained from `Extract` into RLWE encryption of $\mathtt{f}\cdot X^u$ where $\mathtt{f} = -\sum_{j=-c}^{c}\Delta\cdot j\cdot X^j$ for $\|\boldsymbol{u}\|_\infty \le c < \frac{N}{2}$. The result will be accumulated into the RLWE ciphertext that we call `ACC`. Blind rotation public keys $\mathtt{brk} = \left\{\mathsf{RGSW}_{Q,\boldsymbol{s}}(s_i^+), \mathsf{RGSW}_{Q,\boldsymbol{s}}(s_i^-)\right\}_{i\in[0,N-1]}$ where

$$\begin{cases} s_i^+ = 1, & \text{if } s_i = 1 \\ s_i^+ = 0, & \text{otherwise} \end{cases}, \quad \begin{cases} s_i^- = 1, & \text{if } s_i = -1 \\ s_i^- = 0, & \text{otherwise} \end{cases}$$

for $i\in[0,N-1]$ must be generated in advance.

We initialize `ACC` as $\mathtt{ACC} = (0,\mathtt{f}\cdot X^\beta) = \mathsf{RLWE}^0_{Q,\boldsymbol{s}}(\mathtt{f}\cdot X^\beta)$. Then we iteratively compute

$$\mathsf{RGSW}(X^{\alpha_i\cdot s_i}) = \mathsf{RGSW}^0(1) + (X^{\alpha_i}-1)\cdot\mathsf{RGSW}(s_i^+) + (X^{-\alpha_i}-1)\cdot\mathsf{RGSW}(s_i^-) \tag{2}$$

and update $\texttt{ACC}$ as

$$\texttt{ACC} \leftarrow \texttt{ACC} \circledast \mathsf{RGSW}_{Q,\boldsymbol{s}}(X^{\alpha_i \cdot s_i}).$$

The equation (2) is correct as for each $s_i \in \{-1, 0, 1\}$, at least one of s_i^+ and s_i^- is zero. The result of the blind rotation is

$$\mathsf{RLWE}_{Q,\boldsymbol{s}}(\mathtt{f} \cdot X^{\beta+\alpha_0 s_0+\cdots+\alpha_{N-1}s_{N-1}}) = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\mathtt{f} \cdot X^u) = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}_{\mathtt{f}}).$$

Due to the initial function $\mathtt{f}$ and the boundary of $\|\boldsymbol{u}\|_\infty$, the polynomial $\boldsymbol{u}_{\mathtt{f}}$ has $\Delta \cdot u$ as its constant term. The full algorithm is described in Algorithm 1.

Algorithm 1 Blind rotation

procedure $\textsc{BlindRotate}(\mathtt{f}, (\vec{\alpha}, \beta), \mathtt{brk} = \{\mathsf{RGSW}(s_i^{\pm})\})$ ▷ $\beta + \langle \vec{\alpha}, \vec{s} \rangle = u \in \mathbb{Z}_{2N}$

  $\texttt{ACC} \leftarrow \left(0, \mathtt{f} \cdot X^\beta\right)$

  for $(i = 0; i < N; i = i + 1)$ **do**

    $\texttt{ACC} \leftarrow \texttt{ACC} \circledast \left(\mathsf{RGSW}^0(1) + (X^{\alpha_i} - 1) \cdot \mathsf{RGSW}(s_i^+) + (X^{-\alpha_i} - 1) \cdot \mathsf{RGSW}(s_i^-)\right)$

return $\texttt{ACC} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\mathtt{f} \cdot X^u)$

We apply the blind rotation to each $\mathsf{LWE}^0_{2N,\vec{s}}(u_i)$ for all $i \in [0, N-1]$ and obtain

$$\mathsf{RLWE}_{Q,\boldsymbol{s}}(\mathtt{f} \cdot X^{u_i}) := \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i)}) := (\boldsymbol{a}_i, \boldsymbol{b}_i) \in \mathcal{R}_Q^2$$

such that

$$\boldsymbol{a}_i \cdot \boldsymbol{s} + \boldsymbol{b}_i = \boldsymbol{u}^{(i)} + \boldsymbol{e}_{\mathtt{br}} = \Delta \cdot u_i + * \cdot X + * \cdot X^2 + \cdots + * \cdot X^{N-1} + \boldsymbol{e}_{\mathtt{br}} \pmod{Q},$$

where $*$ denotes some value in $\mathbb{Z}_Q$. As $|u_i| \leq c$, most coefficients of $\boldsymbol{u}^{(i)}$ are zeros. More precisely, we have

$$\boldsymbol{u}^{(i)} = \Delta \cdot u_i + * \cdot X + \cdots + * \cdot X^{2c} + 0 \cdot X^{2c+1} + \cdots + 0 \cdot X^{N-2c-2} + * \cdot X^{N-2c-1} + \cdots + * \cdot X^{N-1}.$$

Remark. *It is worth noting that all the 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 in [40, 42]. Instead, we can postpone rescaling to the end to reduce the complexity.*

Step 3. Repacking.

After `BlindRotate`, we receive N RLWE ciphertexts which encrypt polynomials $\boldsymbol{u}^{(i)}$ introduced earlier. Only constant coefficients of the encrypted polynomials contain useful information. Thus, other coefficients of these polynomials must be removed. The goal of `Repack` procedure is to combine all constant coefficients of encryption of $\boldsymbol{u}^{(i)}$ 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 $\mathtt{f}$. Given $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i)})$ for $i \in [0, N-1]$, `Repack` algorithm is performed in the following

two steps. First, we consider a subset of the given ciphertexts as $\left\{\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(nk)})\right\}_{k\in[0,\frac{N}{n}-1]}$. We pack these ciphertexts into the following n RLWE ciphertext

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

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

$$\boldsymbol{u}^{(0,n)}=\Delta\cdot u_0+*\cdot X+\cdots+*\cdot X^{n-1}+\Delta\cdot u_n\cdot X^n+\cdots+\Delta\cdot u_{2n}\cdot X^{2n}+\cdots+*\cdot X^{N-1}$$

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

Second, we adapt the repacking technique from [25]. We consider a pair $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(0,n)})$ and $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(\frac{n}{2},n)})$. Notice that the automorphism $\psi_{1+\frac{2N}{n}}$ applied to $\boldsymbol{u}^{(0,n)}$ 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 operates on the other coefficients. We can merge $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(0,n)})$ and $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(\frac{n}{2},n)})$ as

$$\begin{aligned}\mathsf{RLWE}(2\boldsymbol{u}^{(0,\frac{n}{2})})=\mathsf{RLWE}(\boldsymbol{u}^{(0,n)})+X^{\frac{n}{2}}\cdot\mathsf{RLWE}(\boldsymbol{u}^{(\frac{n}{2},n)})\\+\mathtt{EvalAuto}\left(\mathsf{RLWE}(\boldsymbol{u}^{(0,n)})-X^{\frac{n}{2}}\cdot\mathsf{RLWE}(\boldsymbol{u}^{(\frac{n}{2},n)}),1+\frac{2N}{n}\right),\end{aligned}$$

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

$$\mathsf{RLWE}_{Q,\boldsymbol{s}}(n\cdot\boldsymbol{u}^{(0,1)})=\mathsf{RLWE}_{Q,\boldsymbol{s}}(n\cdot\Delta\cdot\boldsymbol{u}).$$

The full `Repack` algorithm is described in Algorithm 2.

Remark. *We obtain $\mathsf{RLWE}_{Q,\boldsymbol{s}}(n\cdot\Delta\cdot\boldsymbol{u})$ instead of $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\Delta\cdot\boldsymbol{u})$ during the `ScaledMod` procedure and accumulate errors during the blind rotations and repacking procedures. By modifying the initial state, we can address the first issue. We start with $[n]_Q^{-1}\cdot\Delta$ instead of Δ when Q and n are coprime. When Q is a power of two, we start with RLWE modulus $Q\cdot n$ and then rescale by n. For the second issue we use an auxiliary modulus $p>E_{sm}$ and do all the computations modulo $Q\cdot p$ instead of Q and rescale the result by p in the end. To do that, we also start with $\Delta\cdot p$ instead of Δ. Finally we obtain $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\Delta\cdot\boldsymbol{u})$ with only rescaling error $\boldsymbol{e}_{sm}=\boldsymbol{e}_{rs}$.*

4 Bootstrapping

In this section, we present the whole procedure of the new bootstrapping technique which uses `ScaledMod` algorithm as a core functionality. We mainly deal with the CKKS scheme and then briefly describe how to adopt our bootstrapping technique for the BGV and BFV schemes.

Algorithm 2 Repacking

procedure $\texttt{REPACK}(\{\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i)})\}_{i\in[0,N-1]}, n)$ ▷ power of two n s.t. $n \le N$
  for $(i = 0; i < n; i = i + 1)$ **do**
    $\texttt{ct}^{(i,n)} \leftarrow \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i)})$
    for $(j = 1; j < \frac{N}{n}; j = j + 1)$ **do**
      $\texttt{ct}^{(i,n)} \leftarrow \texttt{ct}^{(i,n)} + X^{nj} \cdot \texttt{ct}^{(i+nj)}$
  for $(k = n; k > 1; k = \frac{k}{2})$ **do**
    for $(i = 0; i < \frac{k}{2}; i = i + 1)$ **do**
      $\texttt{ct}^{(i,\frac{k}{2})} \leftarrow \texttt{ct}^{(i,k)} + X^{\frac{k}{2}} \cdot \texttt{ct}^{(i+\frac{k}{2},k)}$
      $\texttt{ct}^{\texttt{rot}} \leftarrow \texttt{EvalAuto}\left(\texttt{ct}^{(i,k)} - X^{\frac{k}{2}} \cdot \texttt{ct}^{(i+\frac{k}{2},k)}, 1 + \frac{2N}{k}\right)$
      $\texttt{ct}^{(i,\frac{k}{2})} \leftarrow \texttt{ct}^{(i,\frac{k}{2})} + \texttt{ct}^{\texttt{rot}}$
return $\texttt{ct}^{(0,1)} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(n \cdot \Delta \cdot \boldsymbol{u})$

4.1 Bootstrapping for CKKS

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

$$\texttt{ct}(\boldsymbol{s}) = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{b} = \boldsymbol{m} + \boldsymbol{e} + q \cdot \boldsymbol{v} \in \mathcal{R}_Q$$

for some small polynomial $\boldsymbol{v}$. To remove the $q \cdot \boldsymbol{v}$ part, existing CKKS bootstrapping methods mainly use homomorphic linear transformations and evaluation of approximating polynomials for modular reduction functions.

On the other hand, our new bootstrapping technique makes use of a blind rotation technique to remove $q \cdot \boldsymbol{v}$ part instead of using polynomial approximation and homomorphic linear transformations. The `ScaledMod` algorithm is presented in the previous section which uses the blind rotation technique as a sub-algorithm and can be used to compute $q \cdot \boldsymbol{v}$. As a first step, the ciphertext is preprocessed to get a ciphertext suitable for `ScaledMod`. After obtaining the result of `ScaledMod`, we add it with the other preprocessed ciphertext and the final result will be a bootstrapped ciphertext. The preprocessing procedure is different depending on the structure of the scheme and the detailed descriptions are given in the following subsections.

4.1.1 Multiprecision CKKS.

In the multiprecision CKKS scheme [11], the ciphertext modulus q is a power of two. Given a ciphertext $\texttt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ for a small modulus q, the decryption query is described as

$$\texttt{ct}(\boldsymbol{s}) = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{b} = \boldsymbol{m} + \boldsymbol{e} \pmod q = \boldsymbol{m} + \boldsymbol{e} + q \cdot \boldsymbol{v}.$$

Let $q' = q/2N$ and $\|\boldsymbol{m} + \boldsymbol{e}\|_\infty \le \gamma < \frac{q}{4} - \frac{q'}{2} \cdot (\delta_{\mathcal{R}} + 1)$ for some γ. Firstly, we compute $\texttt{ct}' = \texttt{ct} \pmod{q'} = \left([\boldsymbol{a}]_{q'}, [\boldsymbol{b}]_{q'}\right) \in \mathcal{R}_{q'}^2$ and obtain

$$\texttt{ct}'(\boldsymbol{s}) = [\boldsymbol{a}]_{q'} \cdot \boldsymbol{s} + [\boldsymbol{b}]_{q'} = \boldsymbol{m} + \boldsymbol{e} \pmod{q'} = \boldsymbol{m} + \boldsymbol{e} + q' \cdot \boldsymbol{u}.$$

Due to $q' \cdot \boldsymbol{u} = [\boldsymbol{a}]_{q'} \cdot \boldsymbol{s} + [\boldsymbol{b}]_{q'} - (\boldsymbol{m} + \boldsymbol{e})$, we have

$$\|\boldsymbol{u}\|_\infty < \frac{1}{2}(\delta_{\mathcal{R}} + 1) + \frac{\gamma}{q'} < \frac{N}{2}$$

for ternary secret key. Now both $\boldsymbol{a} - [\boldsymbol{a}]_{q'}$ and $\boldsymbol{b} - [\boldsymbol{b}]_{q'}$ are divisible by q', thus we can obtain a ciphertext

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

It is easy to see that the preprocessed ciphertext $\mathtt{ct}_{\mathtt{prep}} = \mathsf{RLWE}^0_{2N,\boldsymbol{s}}(-\boldsymbol{u})$, so we can evaluate $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q', Q)$ by setting $c = \left\lfloor \frac{1}{2}(\delta_{\mathcal{R}} + 1) + \frac{\gamma}{q'} \right\rfloor$ for the initial function $\mathtt{f}$ and obtain $\mathtt{ct}_{\mathtt{sm}} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(-q' \cdot \boldsymbol{u})$ with an error $\boldsymbol{e}_{\mathtt{sm}}$ such that

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

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

$$\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{boot}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} = \boldsymbol{m} + \boldsymbol{e} + q' \cdot \boldsymbol{u} - q' \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} = \boldsymbol{m} + \boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Q}.$$

The full bootstrapping algorithm is described in Algorithm 3.

Algorithm 3 Bootstrapping for CKKS

procedure BOOTSTRAP-CKKS($\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$)

- $\mathtt{Preprocess}(\mathtt{ct}) \to \mathtt{ct}', \mathtt{ct}_{\mathtt{prep}}$ ▷ $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + q \cdot \boldsymbol{v} \in \mathcal{R}$
  - $\mathtt{ct}' \leftarrow \mathtt{ct} \pmod{q'}$ ▷ $\mathtt{ct}'(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + q' \cdot \boldsymbol{u} \in \mathcal{R}$
  - $\mathtt{ct}_{\mathtt{prep}} \leftarrow \left(\frac{\mathtt{ct} - \mathtt{ct}'}{q'}\right)$ ▷ $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = -\boldsymbol{u} \in \mathcal{R}_{2N}$
- $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q', Q) \to \mathtt{ct}_{\mathtt{sm}}$ ▷ $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = -q' \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \in \mathcal{R}_Q$
- $\mathtt{Combine}(\mathtt{ct}_{\mathtt{sm}}, \mathtt{ct}') \to \mathtt{ct}_{\mathtt{boot}}$
  - $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}' \pmod{Q}$ ▷ $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}} \in \mathcal{R}_Q$

return $\mathtt{ct}_{\mathtt{boot}} = (\boldsymbol{a}_{\mathtt{boot}}, \boldsymbol{b}_{\mathtt{boot}}) \in \mathcal{R}_Q^2$

Sparsely Packed Ciphertext The bootstrapping complexity can be reduced with sparse packing [12]. Let $\mathtt{ct}$ be an encryption of a sparsely packed plaintext $\boldsymbol{m}$ which encodes $n = n_s$ values where $n \leq \frac{N}{2}$. The main idea is to reduce the number of coefficients of $\boldsymbol{u}$ which will be inputs of the blind rotations in $\mathtt{ScaledMod}$ procedure. We firstly prepare $\mathtt{ct}'$ and $\mathtt{ct}_{\mathtt{prep}}$ as previously and execute an additional preprocessing for $\mathtt{ct}'$ to obtain $\mathtt{ct}''$. Then we apply a variant of $\mathtt{ScaledMod}$ to $\mathtt{ct}_{\mathtt{prep}}$ and combine it to $\mathtt{ct}''$.

The additional preprocessing for $\mathtt{ct}'$ is zeroizing certain coefficients of $\boldsymbol{u}$. We take a similar approach used in the original CKKS bootstrapping [16] which is presented in Algorithm 4. It increases the modulus of $\mathtt{ct}'$ from q' to Q and then applies automorphisms and additions to $\mathtt{ct}'$. After the zeroizing procedure, we obtain a ciphertext $\mathtt{ct}''$ which is described as

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

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

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

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

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

$$\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{boot}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} = \boldsymbol{m} + \boldsymbol{e}'' + q' \cdot \boldsymbol{u}' - q' \cdot \boldsymbol{u}' + \boldsymbol{e}'_{\mathtt{sm}} = \boldsymbol{m} + \boldsymbol{e}'' + \boldsymbol{e}'_{\mathtt{sm}} \pmod{Q'}.$$

Algorithm 4 Zeroizing coefficients

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

4.1.2 RNS-CKKS.

In the RNS-CKKS scheme [12], the modulus q is not a power of two but a product of primes, so the preprocessing steps are different. We start from the decryption query for the given ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ described as

$$\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{b} = \boldsymbol{m} + \boldsymbol{e} + q \cdot \boldsymbol{v} \in \mathcal{R}.$$

Let $\|\boldsymbol{m} + \boldsymbol{e}\|_\infty \leq \gamma < \frac{q}{4} - \frac{q}{4N} \cdot (\delta_{\mathcal{R}} + 1)$ for some γ. We first compute $\mathtt{ct}' = 2N \cdot \mathtt{ct} \pmod{q} = ([2N \cdot \boldsymbol{a}]_q, [2N \cdot \boldsymbol{b}]_q) \in \mathcal{R}_q^2$ to obtain

$$\mathtt{ct}'(\boldsymbol{s}) = [2N \cdot \boldsymbol{a}]_q \cdot \boldsymbol{s} + [2N \cdot \boldsymbol{b}]_q = 2N \cdot \boldsymbol{m} + 2N \cdot \boldsymbol{e} + q \cdot \boldsymbol{u} \in \mathcal{R},$$

where

$$\|\boldsymbol{u}\|_\infty < \frac{1}{2}(\delta_{\mathcal{R}} + 1) + \frac{2N}{q} \cdot \gamma < \frac{N}{2}.$$

Now both $2N \cdot \boldsymbol{a} - [2N \cdot \boldsymbol{a}]_q$ and $2N \cdot \boldsymbol{b} - [2N \cdot \boldsymbol{b}]_q$ are divisible by q, thus we can obtain a ciphertext

$$\mathtt{ct}_{\mathtt{prep}} = (\boldsymbol{a}_{\mathtt{prep}}, \boldsymbol{b}_{\mathtt{prep}}) = \left(\frac{2N \cdot \boldsymbol{a} - [2N \cdot \boldsymbol{a}]_q}{q}, \frac{2N \cdot \boldsymbol{b} - [2N \cdot \boldsymbol{b}]_q}{q} \right) \in \mathcal{R}_{2N}^2.$$

Again for the preprocessed $\mathtt{ct}_{\mathtt{prep}} = \mathsf{RLWE}^0_{2N,\boldsymbol{s}}(-\boldsymbol{u})$, we can evaluate $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q, Qp)$ by setting $c = \left\lfloor \frac{1}{2}(\delta_{\mathcal{R}} + 1) + \frac{\gamma}{q'} \right\rfloor$ for the initial function $\mathtt{f}$ and obtain a ciphertext $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = (\boldsymbol{a}_{\mathtt{sm}}, \boldsymbol{b}_{\mathtt{sm}})$ which satisfies

$$\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{sm}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{sm}} = -q \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Qp},$$

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

$$\begin{aligned}\mathtt{ct}''(\boldsymbol{s}) &= \boldsymbol{a}'' \cdot \boldsymbol{s} + \boldsymbol{b}'' = 2N \cdot \boldsymbol{m} + 2N \cdot \boldsymbol{e} + q \cdot \boldsymbol{u} - q \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \\ &= 2N \cdot \boldsymbol{m} + 2N \cdot \boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Qp}.\end{aligned}$$

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

$$\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{boot}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} = \boldsymbol{m} + \boldsymbol{e} + \frac{1}{2N} \cdot \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{e}_{\mathtt{rs}} \pmod{Q}.$$

The full bootstrapping algorithm is described in Algorithm 5.

Algorithm 5 Bootstrapping for RNS-CKKS

procedure BOOTSTRAP-RNS-CKKS($\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$)

- $\mathtt{Preprocess}(\mathtt{ct}) \to \mathtt{ct}', \mathtt{ct}_{\mathtt{prep}}$ ▷ $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + q \cdot \boldsymbol{v} \in \mathcal{R}$
  - $\mathtt{ct}' \leftarrow 2N \cdot \mathtt{ct} \pmod{q}$ ▷ $\mathtt{ct}'(\boldsymbol{s}) = 2N \cdot \boldsymbol{m} + 2N \cdot \boldsymbol{e} + q \cdot \boldsymbol{u} \in \mathcal{R}$
  - $\mathtt{ct}_{\mathtt{prep}} \leftarrow \left(\frac{2N \cdot \mathtt{ct} - \mathtt{ct}'}{q}\right)$ ▷ $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = -\boldsymbol{u} \in \mathcal{R}_{2N}$
- $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q, Qp) \to \mathtt{ct}_{\mathtt{sm}}$ ▷ $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = -q \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \in \mathcal{R}_{Qp}$
- $\mathtt{Combine}(\mathtt{ct}_{\mathtt{sm}}, \mathtt{ct}') \to \mathtt{ct}_{\mathtt{boot}}$
  - $\mathtt{ct}'' \leftarrow \mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}' \pmod{Qp}$ ▷ $\mathtt{ct}''(\boldsymbol{s}) = 2N \cdot \boldsymbol{m} + 2N \cdot \boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}} \in \mathcal{R}_{Qp}$
  - $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{Rescale}\left(\frac{p}{2N} \cdot \mathtt{ct}'', p\right)$ ▷ $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + \frac{1}{2N} \cdot \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{e}_{\mathtt{rs}} \in \mathcal{R}_Q$

return $\mathtt{ct}_{\mathtt{boot}} = (\boldsymbol{a}_{\mathtt{boot}}, \boldsymbol{b}_{\mathtt{boot}}) \in \mathcal{R}_Q^2$

4.2 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

$$\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{b} = \boldsymbol{m} + t \cdot \boldsymbol{e} + q \cdot \boldsymbol{v} \in \mathcal{R}.$$

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 $t \cdot \boldsymbol{e}$ for multiprecision BGV and $2Nt \cdot \boldsymbol{e}$ for RNS-BGV instead of $\boldsymbol{e}$ and the automorphism and rescale in `Repack` procedure are evaluated in accordance with BGV style.

4.2.1 Multiprecision BGV.

Assume that we have a BGV ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ for a small modulus q where

$$\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{b} = \boldsymbol{m} + t \cdot \boldsymbol{e} \pmod{q} = \boldsymbol{m} + t \cdot \boldsymbol{e} + q \cdot \boldsymbol{v}.$$

Let $q' = q/2N$ and $\|\boldsymbol{m} + t \cdot \boldsymbol{e}\|_\infty \le \gamma < \frac{q}{4} - \frac{q'}{2} \cdot (\delta_{\mathcal{R}} + 1)$ for some γ. We compute $\mathtt{ct}' = \mathtt{ct} \pmod{q'} = \left([\boldsymbol{a}]_{q'}, [\boldsymbol{b}]_{q'}\right) \in \mathcal{R}_{q'}^2$ and have

$$\mathtt{ct}'(\boldsymbol{s}) = [\boldsymbol{a}]_{q'} \cdot \boldsymbol{s} + [\boldsymbol{b}]_{q'} = \boldsymbol{m} + t \cdot \boldsymbol{e} \pmod{q'} = \boldsymbol{m} + t \cdot \boldsymbol{e} + q' \cdot \boldsymbol{u},$$

where

$$\|\boldsymbol{u}\|_\infty < \frac{1}{2}(\delta_{\mathcal{R}}+1) + \frac{\gamma}{q'} < \frac{N}{2}$$

for ternary secret key. Both $\boldsymbol{a}-[\boldsymbol{a}]_{q'}$ and $\boldsymbol{b}-[\boldsymbol{b}]_{q'}$ are divisible by q' and thus, we obtain a ciphertext

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

We set $c = \left\lfloor \frac{1}{2}(\delta_{\mathcal{R}}+1) + \frac{\gamma}{q'} \right\rfloor$ and apply $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q', Q)$ to obtain $\mathtt{ct}_{\mathtt{sm}} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(-q'\cdot\boldsymbol{u})$ with an error $t\cdot\boldsymbol{e}_{\mathtt{sm}}$. Finally, we add $\mathtt{ct}_{\mathtt{sm}}$ and $\mathtt{ct}'$ modulo Q and have the ciphertext $\mathtt{ct}_{\mathtt{boot}}$ satisfying

$$\begin{aligned}\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) &= \boldsymbol{a}_{\mathtt{boot}}\cdot\boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} \\ &= \boldsymbol{m} + t\cdot\boldsymbol{e} + q'\cdot\boldsymbol{u} - q'\cdot\boldsymbol{u} + t\cdot\boldsymbol{e}_{\mathtt{sm}} \\ &= \boldsymbol{m} + t\cdot(\boldsymbol{e}+\boldsymbol{e}_{\mathtt{sm}}) \pmod{Q},\end{aligned}$$

It shows that $\mathtt{ct}_{\mathtt{boot}}$ is a BGV encryption of $\boldsymbol{m}$ with noise $t\cdot(\boldsymbol{e}+\boldsymbol{e}_{\mathtt{sm}})$ and modulus Q. The full algorithm for bootstrapping in BGV is described in Algorithm 6.

Algorithm 6 Bootstrapping for BGV

procedure BOOTSTRAP-BGV($\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$)

- $\mathtt{Preprocess}(\mathtt{ct}) \to \mathtt{ct}', \mathtt{ct}_{\mathtt{prep}}$ ▷ $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{m} + t\cdot\boldsymbol{e} + q\cdot\boldsymbol{v} \in \mathcal{R}$
  - $\mathtt{ct}' \leftarrow \mathtt{ct} \pmod{q'}$ ▷ $\mathtt{ct}'(\boldsymbol{s}) = \boldsymbol{m} + t\cdot\boldsymbol{e} + q'\cdot\boldsymbol{u} \in \mathcal{R}$
  - $\mathtt{ct}_{\mathtt{prep}} \leftarrow \left(\frac{\mathtt{ct}-\mathtt{ct}'}{q}\right)$ ▷ $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = -\boldsymbol{u} \in \mathcal{R}_{2N}$
- $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q', Q) \to \mathtt{ct}_{\mathtt{sm}}$ ▷ $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = -q'\cdot\boldsymbol{u} + t\cdot\boldsymbol{e}_{\mathtt{sm}} \in \mathcal{R}_Q$
- $\mathtt{Combine}(\mathtt{ct}_{\mathtt{sm}}, \mathtt{ct}') \to \mathtt{ct}_{\mathtt{boot}}$
  - $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}' \pmod{Q}$ ▷ $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{m} + t\cdot(\boldsymbol{e}+\boldsymbol{e}_{\mathtt{sm}}) \in \mathcal{R}_Q$

return $\mathtt{ct}_{\mathtt{boot}} = (\boldsymbol{a}_{\mathtt{boot}}, \boldsymbol{b}_{\mathtt{boot}}) \in \mathcal{R}_Q^2$

4.2.2 RNS-BGV.

We start from the decryption query for a given RNS-BGV ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ described as

$$\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{a}\cdot\boldsymbol{s} + \boldsymbol{b} = \boldsymbol{m} + t\cdot\boldsymbol{e} + q\cdot\boldsymbol{v} \in \mathcal{R}.$$

Assume that $\|\boldsymbol{m}+t\cdot\boldsymbol{e}\|_\infty \le \gamma < \frac{q}{4} - \frac{q}{4N}\cdot(\delta_{\mathcal{R}}+1)$ for some γ. We compute $\mathtt{ct}' = 2N\cdot\mathtt{ct} \pmod{q} = ([2N\cdot\boldsymbol{a}]_q, [2N\cdot\boldsymbol{b}]_q) \in \mathcal{R}_q^2$ and have

$$\mathtt{ct}'(\boldsymbol{s}) = [2N\cdot\boldsymbol{a}]_q\cdot\boldsymbol{s} + [2N\cdot\boldsymbol{b}]_q = 2N\cdot\boldsymbol{m} + 2Nt\cdot\boldsymbol{e} + q\cdot\boldsymbol{u} \in \mathcal{R},$$

where

$$\|\boldsymbol{u}\|_\infty < \frac{1}{2}(\delta_{\mathcal{R}}+1) + \frac{2N}{q}\cdot\gamma < \frac{N}{2}.$$

Both $2N\cdot\boldsymbol{a} - [2N\cdot\boldsymbol{a}]_q$ and $2N\cdot\boldsymbol{b} - [2N\cdot\boldsymbol{b}]_q$ are divisible by q and thus, we obtain a ciphertext

$$\mathtt{ct}_{\mathtt{prep}} = (\boldsymbol{a}_{\mathtt{prep}}, \boldsymbol{b}_{\mathtt{prep}}) = \left(\frac{2N\cdot\boldsymbol{a}-[2N\cdot\boldsymbol{a}]_q}{q}, \frac{2N\cdot\boldsymbol{b}-[2N\cdot\boldsymbol{b}]_q}{q}\right) \in \mathcal{R}_{2N}^2.$$

For the preprocessed ciphertext $\mathtt{ct}_{\mathtt{prep}} = \mathsf{RLWE}^0_{2N,\boldsymbol{s}}(-\boldsymbol{u})$, we can evaluate $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q, Q)$ by setting $c = \left\lfloor \frac{1}{2}(1+\delta_{\mathcal{R}}) + \frac{2N}{q} \cdot \gamma \right\rfloor$ and obtain $\mathtt{ct}_{\mathtt{sm}}$ satisfying

$$\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{sm}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{sm}} = -q \cdot \boldsymbol{u} + 2Nt \cdot \boldsymbol{e}_{\mathtt{sm}} \pmod{Q}.$$

Now we add $\mathtt{ct}'' = \mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}'$ modulo Q and have

$$\begin{aligned}\mathtt{ct}''(\boldsymbol{s}) &= \boldsymbol{a}'' \cdot \boldsymbol{s} + \boldsymbol{b}'' = 2N \cdot \boldsymbol{m} + 2Nt \cdot \boldsymbol{e} + q \cdot \boldsymbol{u} - q \cdot \boldsymbol{u} + 2Nt \cdot \boldsymbol{e}_{\mathtt{sm}} \\ &= 2N \cdot (\boldsymbol{m} + t \cdot \boldsymbol{e} + t \cdot \boldsymbol{e}_{\mathtt{sm}}) \pmod{Q},\end{aligned}$$

To get rid of scaling factor $2N$ from the message, we multiply $\mathtt{ct}''$ by $[(2N)^{-1}]_Q$, and obtain $\mathtt{ct}_{\mathtt{boot}}$ satisfying

$$\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{boot}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} = \boldsymbol{m} + t \cdot (\boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}}) \pmod{Q}.$$

It shows that $\mathtt{ct}_{\mathtt{boot}}$ is a RNS-BGV encryption of $\boldsymbol{m}$ with noise $t \cdot (\boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}})$ and modulus Q. The full algorithm is described in Algorithm 7.

Algorithm 7 Bootstrapping for RNS-BGV

procedure BOOTSTRAP-RNS-BGV($\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$)
- $\mathtt{Preprocess}(\mathtt{ct}) \to \mathtt{ct}', \mathtt{ct}_{\mathtt{prep}}$ ▷ $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{m} + t \cdot \boldsymbol{e} + q \cdot \boldsymbol{u} \in \mathcal{R}$
  - $\mathtt{ct}' \leftarrow 2N \cdot \mathtt{ct} \pmod{q}$ ▷ $\mathtt{ct}'(\boldsymbol{s}) = 2N \cdot \boldsymbol{m} + 2Nt \cdot \boldsymbol{e} + q \cdot \boldsymbol{u} \in \mathcal{R}$
  - $\mathtt{ct}_{\mathtt{prep}} \leftarrow \left(\frac{2N \cdot \mathtt{ct} - \mathtt{ct}'}{q}\right)$ ▷ $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = -\boldsymbol{u} \in \mathcal{R}_{2N}$
- $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q, Q) \to \mathtt{ct}_{\mathtt{sm}}$ ▷ $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = -q \cdot \boldsymbol{u} + 2Nt \cdot \boldsymbol{e}_{\mathtt{sm}} \in \mathcal{R}_Q$
- $\mathtt{Combine}(\mathtt{ct}_{\mathtt{sm}}, \mathtt{ct}') \to \mathtt{ct}_{\mathtt{boot}}$
  - $\mathtt{ct}'' \leftarrow (\mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}') \pmod{Q}$ ▷ $\mathtt{ct}''(\boldsymbol{s}) = 2N \cdot (\boldsymbol{m} + t \cdot \boldsymbol{e} + t \cdot \boldsymbol{e}_{\mathtt{sm}}) \in \mathcal{R}_Q$
  - $\mathtt{ct}_{\mathtt{boot}} \leftarrow ([(2N)^{-1}]_Q) \cdot \mathtt{ct}'' \pmod{Q}$ ▷ $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{m} + t \cdot \boldsymbol{e}' \in \mathcal{R}_Q$

return $\mathtt{ct}_{\mathtt{boot}} = (\boldsymbol{a}_{\mathtt{boot}}, \boldsymbol{b}_{\mathtt{boot}}) \in \mathcal{R}_Q^2$

4.3 Bootstrapping for BFV

The decryption query of the BFV scheme is

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

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 $\boldsymbol{e}$ is removed and replaced with a small refreshed error generated from $\mathtt{ScaledMod}$ and rescaling.

4.3.1 Multiprecision BFV.

Bootstrapping for multiprecision BFV scheme with a power of two Q starts with a ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_Q^2$ such that

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

Let $Q' = Q/2N$ and $t \cdot \boldsymbol{e} \le \gamma < \frac{Q}{4} - \frac{Q'}{2} \cdot (\delta_{\mathcal{R}} + 1)$ for some γ. We first multiply t by $\mathtt{ct}$ as $\mathtt{ct}' = t \cdot \mathtt{ct} \pmod{Q}$ and have

$$\mathtt{ct}'(\boldsymbol{s}) = [t \cdot \boldsymbol{a}]_Q \cdot \boldsymbol{s} + [t \cdot \boldsymbol{b}]_Q = t \cdot \boldsymbol{e} + Q \cdot \boldsymbol{v}.$$

We compute $\mathtt{ct}'' = \mathtt{ct}' \pmod{Q'}$ and have

$$\mathtt{ct}''(\boldsymbol{s}) = [t \cdot \boldsymbol{a}]_{Q'} \cdot \boldsymbol{s} + [t \cdot \boldsymbol{a}]_{Q'} = t \cdot \boldsymbol{e} + Q' \cdot \boldsymbol{u},$$

where

$$\|\boldsymbol{u}\|_\infty < \frac{1}{2}(1 + \delta_{\mathcal{R}}) + \frac{\gamma}{Q'} < \frac{N}{2}.$$

Now we obtain the preprocessed ciphertext $\mathtt{ct}_{\mathtt{prep}} = \frac{1}{Q'} \cdot (\mathtt{ct}' - \mathtt{ct}'')$ with

$$\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{prep}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{prep}} = -\boldsymbol{u} + 2N \cdot \boldsymbol{v} \in \mathcal{R}.$$

For $c = \left\lfloor \frac{1}{2}(1 + \delta_{\mathcal{R}}) + \frac{\gamma}{Q'} \right\rfloor$, $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q', Qt)$ outputs the ciphertext $\mathtt{ct}_{\mathtt{sm}}$ satisfying

$$\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{sm}} \cdot \boldsymbol{s} + \boldsymbol{b} = Q' \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Qt}.$$

We evaluate $\mathtt{ct}''' = \mathtt{ct}_{\mathtt{sm}} + t \cdot \mathtt{ct} - \mathtt{ct}'' \pmod{Qt}$ and have

$$\mathtt{ct}'''(\boldsymbol{s}) = \boldsymbol{a}''' \cdot \boldsymbol{s} + \boldsymbol{b}''' = Q' \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} + t \cdot \boldsymbol{e} + Q \cdot \boldsymbol{m} - t \cdot \boldsymbol{e} - Q' \cdot \boldsymbol{u} = \boldsymbol{e}_{\mathtt{sm}} + Q \cdot \boldsymbol{m} \pmod{Qt}.$$

Finally we rescale $\mathtt{ct}'''$ by t and obtain $\mathtt{ct}_{\mathtt{boot}}$ satisfying

$$\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{boot}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} = \frac{1}{t} \cdot \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{e}_{\mathtt{rs}} + \frac{Q}{t} \cdot \boldsymbol{m} \pmod{Q}.$$

The full algorithm is described in Algorithm 8.

Algorithm 8 Bootstrapping for BFV

procedure BOOTSTRAP-BFV($\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_Q^2$)

- $\mathtt{Preprocess}(\mathtt{ct}) \to \mathtt{ct}', \mathtt{ct}'', \mathtt{ct}_{\mathtt{prep}}$ ▷ $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{e} + \frac{Q}{t} \cdot \boldsymbol{m} \in \mathcal{R}_Q$
  - $\mathtt{ct}' \leftarrow t \cdot \mathtt{ct} \pmod{Q}$ ▷ $\mathtt{ct}'(\boldsymbol{s}) = t \cdot \boldsymbol{e} + Q \cdot \boldsymbol{v} \in \mathcal{R}$
  - $\mathtt{ct}'' \leftarrow \mathtt{ct}' \pmod{Q'}$ ▷ $\mathtt{ct}''(\boldsymbol{s}) = t \cdot \boldsymbol{e} + Q' \cdot \boldsymbol{u} \in \mathcal{R}$
  - $\mathtt{ct}_{\mathtt{prep}} \leftarrow \left(\frac{\mathtt{ct}' - \mathtt{ct}''}{Q'}\right) \pmod{2N}$ ▷ $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = -\boldsymbol{u} \in \mathcal{R}_{2N}$
- $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q', Qt) \to \mathtt{ct}_{\mathtt{sm}}$ ▷ $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = Q' \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \in \mathcal{R}_{Qt}$
- $\mathtt{Combine}(\mathtt{ct}_{\mathtt{sm}}, \mathtt{ct}, \mathtt{ct}'') \to \mathtt{ct}_{\mathtt{boot}}$
  - $\mathtt{ct}''' \leftarrow \mathtt{ct}_{\mathtt{sm}} + t \cdot \mathtt{ct} - \mathtt{ct}'' \pmod{Qt}$ ▷ $\mathtt{ct}'''(\boldsymbol{s}) = \boldsymbol{e}_{\mathtt{sm}} + Q \cdot \boldsymbol{m} \in \mathcal{R}_{Qt}$
  - $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{Rescale}(\mathtt{ct}''', t)$ ▷ $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \frac{1}{t} \cdot \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{e}_{\mathtt{rs}} + \frac{Q}{t} \cdot \boldsymbol{m} \in \mathcal{R}_Q$

return $\mathtt{ct}_{\mathtt{boot}} = (\boldsymbol{a}_{\mathtt{boot}}, \boldsymbol{b}_{\mathtt{boot}}) \in \mathcal{R}_Q^2$

4.3.2 RNS-BFV.

Bootstrapping for RNS-BFV also starts with a ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_Q^2$ such that

$$\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{b} = \boldsymbol{e} + \frac{Q}{t} \cdot \boldsymbol{m} \pmod{Q}.$$

Let $t \cdot \boldsymbol{e} \leq \gamma < \frac{Q}{4} - \frac{Q}{4N} \cdot (\delta_{\mathcal{R}} + 1)$ for some γ. We compute $\mathtt{ct}' = t \cdot \mathtt{ct} \pmod{Q}$ and $\mathtt{ct}'' = 2N \cdot \mathtt{ct}' \pmod{Q}$ and we have

$$\mathtt{ct}'(\boldsymbol{s}) = [t \cdot \boldsymbol{a}]_Q \cdot \boldsymbol{s} + [t \cdot \boldsymbol{b}]_Q = t \cdot \boldsymbol{e} + Q \cdot \boldsymbol{v} \in \mathcal{R}$$

and

$$\mathtt{ct}''(\boldsymbol{s}) = [2Nt \cdot \boldsymbol{a}]_Q \cdot \boldsymbol{s} + [2Nt \cdot \boldsymbol{b}]_Q = 2Nt \cdot \boldsymbol{e} + Q \cdot \boldsymbol{u} \in \mathcal{R}.$$

where

$$\|\boldsymbol{u}\|_\infty < \frac{1}{2}(1 + \delta_{\mathcal{R}}) + \frac{2N}{Q} \cdot \gamma < \frac{N}{2}.$$

Now we obtain a preprocessed ciphertext $\mathtt{ct}_{\mathtt{prep}} = \frac{1}{Q} \cdot (2N \cdot \mathtt{ct}' - \mathtt{ct}'')$ with

$$\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{prep}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{prep}} = -\boldsymbol{u} + 2N \cdot \boldsymbol{v} \in \mathcal{R}.$$

For an auxiliary prime p and $c = \left\lfloor \frac{1}{2}(1 + \delta_{\mathcal{R}}) + \frac{2N}{Q} \cdot \gamma \right\rfloor$, $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q, Qpt)$ outputs $\mathtt{ct}_{\mathtt{sm}}$ satisfying

$$\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{sm}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{sm}} = Q \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Qpt}.$$

We evaluate $\mathtt{ct}''' = \mathtt{ct}_{\mathtt{sm}} + 2Nt \cdot \mathtt{ct} - \mathtt{ct}'' \pmod{Qpt}$ and have

$$\begin{aligned}\mathtt{ct}'''(\boldsymbol{s}) &= \boldsymbol{a}''' \cdot \boldsymbol{s} + \boldsymbol{b}''' = Q \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} + 2Nt \cdot \boldsymbol{e} + 2NQ \cdot \boldsymbol{m} - 2Nt \cdot \boldsymbol{e} - Q \cdot \boldsymbol{u} \\ &= \boldsymbol{e}_{\mathtt{sm}} + 2NQ \cdot \boldsymbol{m} \pmod{Qpt}.\end{aligned}$$

Finally we multiply $\mathtt{ct}'''$ by $\frac{p}{2N}$, and rescale the result by p, then obtain $\mathtt{ct}_{\mathtt{boot}}$ satisfying

$$\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{boot}} \cdot \boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} = \frac{1}{2Nt} \cdot \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{e}_{\mathtt{rs}} + \frac{Q}{t} \cdot \boldsymbol{m} \pmod{Q}.$$

The full algorithm is described in Algorithm 9.

5 Compact Representation of Blind Rotation Keys

As presented in Section 3, blind rotation keys should be precomputed by a secret key holder for `BlindRotate` procedure. As a ring dimension increases, the amount of memory required to represent blind rotation keys increases dramatically. It is difficult for the secret key holder to generate and transfer the heavy blind rotation keys to the computational side which performs all the computations. The computational side is also limited to store all the blind rotation keys.

In this section, we provide a possible way to reduce the communication cost and storage size of blind rotation keys. First, we present a method that the secret key holder generates only a small amount of public keys and the blind rotation keys are reconstructed on the computational side instead of being generated by the secret key holder. We also present a method that the computational side reconstructs the keys and performs blind rotations on the fly, without storing the whole keys.

Algorithm 9 Bootstrapping for RNS-BFV

procedure BOOTSTRAP-RNS-BFV($\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_Q^2$)

$\mathtt{Preprocess}(\mathtt{ct}) \to \mathtt{ct}', \mathtt{ct}'', \mathtt{ct}_{\mathtt{prep}}$ ▷ $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{e} + \frac{Q}{t} \cdot \boldsymbol{m} \in \mathcal{R}_Q$

- $\mathtt{ct}' \leftarrow t \cdot \mathtt{ct} \pmod{Q}$ ▷ $\mathtt{ct}'(\boldsymbol{s}) = t \cdot \boldsymbol{e} + Q \cdot \boldsymbol{v} \in \mathcal{R}$
- $\mathtt{ct}'' \leftarrow 2N \cdot \mathtt{ct}' \pmod{Q}$ ▷ $\mathtt{ct}''(\boldsymbol{s}) = 2Nt \cdot \boldsymbol{e} + Q \cdot \boldsymbol{u} \in \mathcal{R}$
- $\mathtt{ct}_{\mathtt{prep}} \leftarrow \left(\frac{2N \cdot \mathtt{ct}' - \mathtt{ct}''}{Q}\right) \pmod{2N}$ ▷ $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = -\boldsymbol{u} \in \mathcal{R}_{2N}$

$\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q, Qpt) \to \mathtt{ct}_{\mathtt{sm}}$ ▷ $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = Q \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \in \mathcal{R}_{Qpt}$

$\mathtt{Combine}(\mathtt{ct}_{\mathtt{sm}}, \mathtt{ct}, \mathtt{ct}'') \to \mathtt{ct}_{\mathtt{boot}}$

- $\mathtt{ct}''' \leftarrow \mathtt{ct}_{\mathtt{sm}} + 2Nt \cdot \mathtt{ct} - \mathtt{ct}'' \pmod{Qpt}$ ▷ $\mathtt{ct}'''(\boldsymbol{s}) = \boldsymbol{e}_{\mathtt{sm}} + 2NQ\boldsymbol{m} \in \mathcal{R}_{Qpt}$
- $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{Rescale}\left(\frac{p}{2N} \cdot \mathtt{ct}''', pt\right)$ ▷ $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \frac{1}{2Nt} \cdot \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{e}_{\mathtt{rs}} + \frac{Q}{t} \cdot \boldsymbol{m} \in \mathcal{R}_Q$

return $\mathtt{ct}_{\mathtt{boot}} = (\boldsymbol{a}_{\mathtt{boot}}, \boldsymbol{b}_{\mathtt{boot}}) \in \mathcal{R}_Q^2$

5.1 Reconstruction of Blind Rotation Keys

For simplicity, we denote s_i^+ and s_i^- as $s_i^\pm$. First, we notice that $\mathsf{RGSW}_{\boldsymbol{s}}(s_i^\pm)$ can be reconstructed from only $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^\pm)$ and $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$, where

$$\boldsymbol{s}^\pm = \sum_{i=0}^{N-1} s_i^\pm X^i.$$

Given $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^\pm)$, $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^\pm)$ can be reconstructed in parallel by using divide-and-conquer algorithm described in Algorithm 10. For each $\mathsf{RLWE}_{\boldsymbol{s}}(g_j \cdot s_i^\pm) = (\boldsymbol{a}_{i,j}, \boldsymbol{b}_{i,j})$ in $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^\pm)$, the reconstruction of $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^\pm \cdot \boldsymbol{s})$ can be done through the reconstruction of $\mathsf{RLWE}_{\boldsymbol{s}}(g_j \cdot s_i^\pm \cdot \boldsymbol{s})$ which is calculated by

$$\boldsymbol{a}_{i,j} \odot \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2) + \boldsymbol{b}_{i,j} \cdot (1,0) = \mathsf{RLWE}_{\boldsymbol{s}}\left(\boldsymbol{a}_{i,j} \cdot \boldsymbol{s}^2 + \boldsymbol{b}_{i,j} \cdot \boldsymbol{s}\right) = \mathsf{RLWE}_{\boldsymbol{s}}(g_j \cdot s_i^\pm \cdot \boldsymbol{s}). \tag{3}$$

After repeating this procedure for all $i \in [0, N-1]$ and $j \in [0, d-1]$, the keys $\mathsf{RGSW}_{\boldsymbol{s}}(s_i^\pm) = \left(\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^\pm), \mathsf{RLWE}'_{\boldsymbol{s}}(s_i^\pm \cdot \boldsymbol{s})\right)$ can be fully reconstructed.

Algorithm 10 Reconstruct blind rotation keys

procedure RECONST-ALL($\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^\pm)$)

$\mathtt{brk}_0^\pm \leftarrow \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^\pm)$

for $(n = N; n > 1; n = n/2)$ **do**

for $(i = 0; i < N; i = i + n)$ **do**

$\mathtt{tmp}_i^\pm \leftarrow \mathtt{EvalAuto}(\mathtt{brk}_i^\pm, n+1)$

$\mathtt{brk}_i^\pm = \mathtt{brk}_i^\pm + \mathtt{tmp}_i^\pm$

$\mathtt{brk}_{i+n/2}^\pm = X^{-N/n} \cdot (\mathtt{brk}_i^\pm - \mathtt{tmp}_i^\pm)$

return $\left\{\mathtt{brk}_i^\pm\right\} = \left\{\mathsf{RLWE}'(N \cdot s_i^\pm)\right\}$ for $i \in [0, N-1]$

$\mathtt{EvalAuto}(\mathsf{RLWE}'(\cdot), \cdot)$ in Algorithm 10 denotes the operation of performing the same $\mathtt{EvalAuto}$ for all the RLWE elements in $\mathsf{RLWE}'(\cdot)$. $\mathtt{EvalAuto}$ by $2N/2^k + 1$ maintains the $2^k \cdot i$-th coefficients and changes the signs of $2^k \cdot i + 2^{k-1}$-th coefficients for an integer i. When the other coefficients are zeros, we can obtain the ciphertext with only $2^k \cdot i$-th coefficients or $2^k \cdot i + 2^{k-1}$-th coefficients by

addition or subtraction of the original ciphertext and its rotation, respectively. By repeating this procedure, we obtain a ciphertext with only one non-zero coefficient, which is an encryption of $s_i^{\pm}$.

In Algorithm 10, the coefficients that are not removed are doubled after each evaluation of automorphism and addition. Therefore, the target coefficient will eventually be multiplied by N and the algorithm outputs $\mathsf{RLWE}'(N \cdot s_i^{\pm})$. There are two possible ways to remove the additional multiplicand N from the polynomial. If Q is coprime with N, the input ciphertext can be multiplied initially by $N^{-1} \pmod{Q}$, i.e. we start with $\mathsf{RLWE}'_{Q,\boldsymbol{s}}([N]_Q^{-1} \cdot \boldsymbol{s}^{\pm})$. Otherwise, if Q is a power of two we start with QN and rescale $\mathsf{RLWE}'_{QN,\boldsymbol{s}}(N \cdot s_i^{\pm})$ by N.

5.2 Performing Blind Rotations On the Fly

To avoid the computational side storing all blind rotation keys, it can reconstruct the blind rotation key for each blind rotation step on the fly, and discard it after.

To reconstruct $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$ for specific i, we multiply $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^{\pm})$ by X^{-i} to have $s_i^{\pm}$ as the constant term, and then make all other coefficients zeros by applying the sequence of automorphisms and additions. Algorithm 11 sums up the reconstruction of $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^{\pm})$ for a single i. We remove N from the result in a similar way as we did in Section 5.1.

Algorithm 11 Reconstruct for performing blind rotation on the fly

procedure RECONST($\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^{\pm}), i$)
  $\mathtt{brk}_i^{\pm} \leftarrow \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^{\pm}) \cdot X^{-i}$
  for $(n = N; n > 1; n = n/2)$ **do**
    $\mathtt{brk}_i^{\pm} \leftarrow \mathtt{EvalAuto}(\mathtt{brk}_i^{\pm}, n+1) + \mathtt{brk}_i^{\pm}$
return $\mathtt{brk}_i^{\pm} = \mathsf{RLWE}'_{\boldsymbol{s}}(N \cdot s_i^{\pm})$

Furthermore, we can either reconstruct $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm} \cdot \boldsymbol{s})$ as explained in Section 5.1, or evaluate the blind rotation step i using only $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$ and $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$. For the latter case, we first evaluate

$$\mathsf{RLWE}'_{\boldsymbol{s}}(X^{\alpha_i \cdot s_i}) = \mathsf{RLWE}'^{0}(1) + (X^{\alpha_i} - 1) \cdot \mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{+}) + (X^{-\alpha_i} - 1) \cdot \mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{-}). \tag{4}$$

For given $\mathtt{ACC} = \mathsf{RLWE}_{\boldsymbol{s}}(\mathtt{f} \cdot X^{\beta + \alpha_0 s_0 + \cdots + \alpha_{i-1} s_{i-1}}) = (\boldsymbol{a}, \boldsymbol{b})$, we multiply $\boldsymbol{a}$ and $\boldsymbol{b}$ by $\mathsf{RLWE}'_{\boldsymbol{s}}(X^{\alpha_i \cdot s_i})$.

$$\begin{aligned}
\boldsymbol{a} \odot \mathsf{RLWE}'_{\boldsymbol{s}}(X^{\alpha_i \cdot s_i}) &= \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{a} \cdot X^{\alpha_i \cdot s_i}) = (\boldsymbol{a}', \boldsymbol{b}') \\
\boldsymbol{b} \odot \mathsf{RLWE}'_{\boldsymbol{s}}(X^{\alpha_i \cdot s_i}) &= \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{b} \cdot X^{\alpha_i \cdot s_i})
\end{aligned}$$

Then we evaluate $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{a} \cdot \boldsymbol{s} \cdot X^{\alpha_i \cdot s_i})$ as

$$\boldsymbol{a}' \odot \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2) + (\boldsymbol{b}', 0) = \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{a}' \cdot \boldsymbol{s}^2 + \boldsymbol{b}' \cdot \boldsymbol{s}) = \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{a} \cdot \boldsymbol{s} \cdot X^{\alpha_i \cdot s_i}).$$

Finally, we add $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{a} \cdot \boldsymbol{s} \cdot X^{\alpha_i \cdot s_i})$ and $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{b} \cdot X^{\alpha_i \cdot s_i})$ to obtain the updated $\mathtt{ACC}$ as

$$\begin{aligned}
\mathtt{ACC} &\leftarrow \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{a} \cdot \boldsymbol{s} \cdot X^{\alpha_i \cdot s_i}) + \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{b} \cdot X^{\alpha_i \cdot s_i}) \\
&= \mathsf{RLWE}_{\boldsymbol{s}}((\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{b}) \cdot X^{\alpha_i \cdot s_i}) = \mathsf{RLWE}_{\boldsymbol{s}}(\mathtt{f} \cdot X^{\beta + \alpha_0 s_0 + \cdots + \alpha_i s_i}).
\end{aligned}$$

The full algorithm is described in Algorithm 12.

Algorithm 12 Reconstruct key and blind rotation on the fly

procedure RECONST-OTF($\mathtt{f}, (\vec{\alpha}, \beta), \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^{\pm}), \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$) ▷ $\beta + \langle \vec{\alpha}, \vec{s} \rangle = u$

$\mathtt{ACC} = (\mathtt{ACC}_{\boldsymbol{a}}, \mathtt{ACC}_{\boldsymbol{b}}) \leftarrow \left(0, \mathtt{f} \cdot X^{\beta}\right)$

for $(i = 0; i < N; i = i + 1)$ **do**

$\mathtt{brk}_i^{\pm} \leftarrow \mathtt{Reconst}(\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^{\pm}), i)$

$\mathtt{brk}_i^{\pm} \leftarrow \mathtt{Rescale}(\mathtt{brk}_i^{\pm}, N)$

$\mathtt{ct}_i \leftarrow \mathsf{RLWE}'^{0}(1) + (X^{\alpha_i} - 1) \cdot \mathtt{brk}_i^{+} + (X^{-\alpha_i} - 1) \cdot \mathtt{brk}_i^{-}$

$(\mathtt{ct}_{\boldsymbol{a}}, \mathtt{ct}_{\boldsymbol{b}}) \leftarrow \mathtt{ACC}_{\boldsymbol{a}} \odot \mathtt{ct}_i$

$\mathtt{ACC} \leftarrow \mathtt{ct}_{\boldsymbol{a}} \odot \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2) + (\mathtt{ct}_{\boldsymbol{b}}, 0) + \mathtt{ACC}_{\boldsymbol{b}} \odot \mathtt{ct}_i$

return $\mathtt{ACC} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\mathtt{f} \cdot X^{u})$

6 Analysis and Implementation

In this section, we analyze the computational complexity and the size of public keys for each bootstrapping algorithm, and present proof-of-concept implementation results with theoretical error analysis.

6.1 Complexity Analysis

The proposed bootstrapping algorithm consists of a number of multiplications $\odot$ between an element in $\mathcal{R}_Q$ and RLWE'_Q ciphertext, which is a bottleneck operation in both `ScaledMod` and reconstruction algorithms. Thus, it is convenient to analyze the computational complexity by the number of $\odot$ operations. We choose $\odot$ for our analysis rather than lower level operations, since it depends on whether we are in multiprecision or RNS case. We also analyze the public key size by the number of RLWE' ciphertexts generated by the secret key holder and stored by the computational side.

The blind rotation requires $2N$ $\odot$ operations for each coefficient of $\boldsymbol{u}$, which can be easily checked from Algorithms 1. In the repacking procedure, only `EvalAuto` requires $2n_c$ $\odot$ operations, where $n_c \leq N$ is defined initially depending on the function `f`. Therefore, the full bootstrapping algorithm requires $4Nn_s + 2n_c$ $\odot$ operations, where n_s is the number of slots in the message. In this general case, the secret key holder generates $4N$ RLWE' ciphertexts for the blind rotation keys and the computational side should store all keys.

When the compact representation method is applied to the secret key, the computational complexity increases by reconstruction of the blind rotation keys. However, the computational side can reconstruct the keys in the precomputation stage, which does not affect the complexity of bootstrapping itself. Algorithm 10 shows that the reconstruction of $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$ requires d $\odot$ operations for each $i \in [0, N-1]$ and the equation (3) shows that the reconstruction of $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm} \cdot \boldsymbol{s})$ also requires d $\odot$ operations for each $i \in [0, N-1]$. The total number of $\odot$ operations in reconstruction is $2dN$. The secret key holder generates $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^{\pm})$ and $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$ which consists of 3 RLWE' ciphertexts and shifts the responsibility to the computational side to reconstruct the blind rotation keys.

In another way, the blind rotation keys can be reconstructed and computed by blind rotation on-the-fly. The reconstruction of $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$ requires $d \log N$ $\odot$ operations and the blind rotation requires 3 $\odot$ operations for all $i \in [0, N-1]$. For each coefficient of $\boldsymbol{u}$, the total number of $\odot$ operations is $dN \log N + 3N$. Since the blind rotation is replaced to the on-the-fly method, the computational side does not have to precompute and store all the blind rotation keys. It stores the

Table 1: Complexity for each bootstrapping. SH denotes a secret key holder, and CP denotes a computational side.

Bootstrapping Method	Storage in # of RLWE'_Q		Complexity in # of $\mathcal{R}_Q \odot \mathsf{RLWE}'_Q$
	gen by SH	stored by CP	
`General`	$4N$	$4N$	$4Nn_s + 2n_c$
`Reconst-Boot`	3	$4N$	$4Nn_s + 2n_c$
`Reconst-OTF-Boot`	3	3	$(3 + d\log N)Nn_s + 2n_c$

keys sent by the secret key holder and each time reconstructs only $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$ for each $i \in [0, N-1]$. In this case, the bootstrapping requires $(d\log N + 3)Nn_s + 2n_c \odot$ operations. Table 1 shows the storage and computational complexity of each bootstrapping method.

6.2 Error Analysis and Implementation Results

We present theoretical analysis of the error growth to set the proper auxiliary modulus and show the implementation results with concrete parameters.

6.2.1 Error analysis.

As discussed in Remark 3.1, a rescaling procedure by an auxiliary modulus is required to make the error $\boldsymbol{e}_{\mathtt{sm}}$ in the output ciphertext of `ScaledMod` comparable to the rescaling error $\boldsymbol{e}_{\mathtt{rs}}$. We analyze the error growth in detail by observing each step in `ScaledMod` and provide a boundary of the auxiliary modulus. Here, we use the approach in [4] and [28] to write the error in a ciphertext after `ScaledMod`. We note that the error in a fresh ciphertext is a Gaussian with standard deviation $\sigma_{\mathtt{err}}$. Let B be a digit bound and d be the number of digits of gadget decomposition in RLWE' and RGSW encryptions.

The blind rotation starts from the equation (2). The error of each RLWE element of the RGSW encryption in (2) is a Gaussian with variance $4\sigma_{\mathtt{err}}^2$ and each blind rotation introduces an additive error that of variance $4dN\frac{B^2}{6}\sigma_{\mathtt{err}}^2$. As shown in Algorithm 1, we perform the same operation for all $i \in [0, N-1]$ and thus, the total error variance after `BlindRotate` is $\sigma_{\mathtt{br}}^2 = 4dN^2\frac{B^2}{6}\sigma_{\mathtt{err}}^2$. The first step of `Repack` adds the $\frac{N}{n}$ resulting ciphertexts of `BlindRotate`, so the error of each $\mathsf{RLWE}_{Q,\boldsymbol{s}}\left(\boldsymbol{u}^{(i,n)}\right)$ is a Gaussian with variance $\frac{N}{n}\sigma_{\mathtt{br}}^2$. For the second step, every `EvalAuto` introduces $\log n$ additive independent key switching errors of variance $\sigma_{\mathtt{ks}}^2 = dN\frac{B^2}{12}\sigma_{\mathtt{err}}^2$. Finally, the total error of `ScaledMod` is a Gaussian with standard deviation

$$\sigma_{\mathtt{sm}} = \sqrt{n^2\frac{N}{n}\sigma_{\mathtt{br}}^2 + \frac{n^2-1}{3}\sigma_{\mathtt{ks}}^2} \approx \sqrt{nN}\sigma_{\mathtt{br}}.$$

Following [11], we can assume that a Gaussian with standard deviation σ is bounded by 6σ and therefore, the auxiliary modulus should be the size of $6\sigma_{\mathtt{sm}} = \sqrt{24dnN}NB\sigma_{\mathtt{err}}$.

When the compact representation method is applied to the secret key, the error of the blind rotation keys increases during reconstruction. Algorithm 10 works similar to `Repack`, so the error in the blind rotation keys can be found in a similar way. $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^{\pm})$ has a Gaussian error of variance $\sigma_{\mathtt{err}}^2$ and the error in the reconstructed key, $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$ is a Gaussian with variance $\sigma_{\mathtt{rc}}^2 = N\sigma_{\mathtt{err}}^2 + \frac{N^2-1}{3}\sigma_{\mathtt{ks}}^2 \approx \frac{N^2}{3}\sigma_{\mathtt{ks}}^2$. The error in $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm} \cdot \boldsymbol{s})$, which is reconstructed by the

equation (3), is a Gaussian with variance ${\sigma'_{\mathtt{rc}}}^2 = \frac{N}{2}\sigma^2_{\mathtt{rc}} + \sigma^2_{\mathtt{ks}} \approx \frac{N^3}{6}\sigma^2_{\mathtt{ks}}$, as the error in $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$ is multiplied by ternary secret key $\boldsymbol{s}$. The error after blind rotation is a Gaussian with variance ${\sigma'_{\mathtt{br}}}^2 = N \cdot (4dN\frac{B^2}{12}\sigma^2_{\mathtt{rc}} + 4dN\frac{B^2}{12}{\sigma'_{\mathtt{rc}}}^2) \approx 4dN^2\frac{B^2}{12}\frac{N^3}{6}\sigma^2_{\mathtt{ks}}$ and the total error of `ScaledMod` with reconstruction is a Gaussian with standard deviation

$$\sigma'_{\mathtt{sm}} \approx \sqrt{nN}\sigma'_{\mathtt{br}} \approx \sqrt{4dnN^3\frac{B^2}{12}\frac{N^3}{6}}\sigma_{\mathtt{ks}}.$$

The auxiliary modulus should be the size of $6\sigma'_{\mathtt{sm}} = \sqrt{\frac{1}{6}nN}dN^3B^2\sigma_{\mathtt{err}}$.

Another way is to perform blind rotation directly from $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$ and $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$ without finding $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm} \cdot \boldsymbol{s})$. The error of $\mathsf{RLWE}'_{\boldsymbol{s}}(X^{\alpha_i \cdot s_i})$ in the equation (4) is a Gaussian with variance $4\sigma^2_{\mathtt{rc}}$ and the error variances of $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{b} \cdot X^{\alpha_i \cdot s_i})$ and $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{a} \cdot \boldsymbol{s} \cdot X^{\alpha_i \cdot s_i})$ are $4dN\frac{B^2}{12}\sigma^2_{\mathtt{rc}}$ and $4d\frac{N^2}{2}\frac{B^2}{12}\sigma^2_{\mathtt{rc}} + \sigma^2_{\mathtt{ks}}$, respectively. A single accumulation introduces an additive Gaussian error of variance $4d\frac{N^2}{2}\frac{B^2}{12}\sigma^2_{\mathtt{rc}}$. The error after blind rotation on-the-fly is a Gaussian with variance ${\sigma''_{\mathtt{br}}}^2 \approx 4d\frac{N^3}{2}\frac{B^2}{12}\sigma^2_{\mathtt{rc}} \approx 4d\frac{N^3}{2}\frac{B^2}{12}\frac{N^2}{3}\sigma^2_{\mathtt{ks}}$, which is the same as $\sigma'_{\mathtt{br}}$. Therefore, auxiliary modulus should be the size of $6\sigma'_{\mathtt{sm}}$.

6.2.2 Implementation.

We have implemented the version of bootstrapping for multiprecision CKKS described in Section 4 as a proof-of-concept and to demonstrate the error achieved after bootstrapping procedure. It was implemented using C++, NTL library [43] for multiprecision integer and floating-point arithmetic and Intel HEXL library [44] for NTT. We experimented for ring dimensions 2^{12} and 2^{13} the result of which is demonstrated in the Table 2. Our particular implementation is not optimized, but performance improvement is not the main goal of this research. Recent results presented in [45] show good performance for blind rotation.

Table 2 shows comparison of CKKS bootstrapping error between our technique and the state-of-the-art bootstrapping methods [20, 21]. Unlike previous methods, our bootstrapping error is not determined by polynomial approximation and is comparable to rescaling error. Due to the small error, our method enables CKKS bootstrapping to work for smaller parameters such as $N = 2^{12}, 2^{13}$ with a better precision. Especially, in $N = 2^{13}$, our bootstrapping preserves more than 70-bit accuracy which is sufficient to achieve $\mathsf{IND\text{-}CPA^D}$ security. Despite the theoretical error analysis showed that the auxiliary prime $p = 2^{42}$ and 2^{60} are required for $(N, B) = (2^{12}, 2^{16})$ and $(2^{13}, 2^{32})$, respectively, we explored that $p = 2^{34}$ and 2^{49} are enough in practice to make the bootstrapping error comparable to rescaling error.

7 Conclusion

We demonstrate that the blind rotation technique that was previously used only for FHEW/TFHE bootstrapping can be used for BGV, BFV, and CKKS bootstrapping as well. For CKKS, our bootstrapping procedure introduces small rescaling errors instead of big approximation errors as in previous CKKS bootstrapping methods, which makes our bootstrapping to be $\mathsf{IND\text{-}CPA^D}$ secure [24]. For BGV and BFV, unlike previous bootstrapping methods, our bootstrapping does not have restrictions on the plaintext modulus.

Table 2: Comparison of the error performance. ϵ denotes the average of bootstrapping error and Δ_{scale} denotes the scaling factor which is multiplied to the message to keep the message precision.

Method	N	n_s	$\log(Q \cdot p)$	$\log Q$	$\log(\Delta_{\mathsf{scale}})$	$\log(\epsilon^{-1})$	IND-CPA$^{\mathsf{D}}$
[20]	2^{16}	2^{14}	1553	240	45	31.6	X
[21]	2^{16}	2^{14}	1547	533	50	32.6	X
Ours	2^{12}	2^{1}	110	76	34	34.97	X
		2^{10}	110	76	34	29.91	X
	2^{13}	2^{1}	219	170	80	79.04	✓
		2^{5}	219	170	80	75.97	✓
		2^{10}	219	170	80	73.58	✓

Further improvements are required in terms of efficiency. As we proposed the compact representation method, we need to reduce the size of a fairly large public key and improve the computational complexity that increases linearly with the number of slots. We plan to continue research on improving the performance of the proposed bootstrapping technique. We believe that this point of view for designing bootstrapping algorithms could be helpful in the study of fully homomorphic encryption schemes.

Acknowledgments

We thank Yuriy Polyakov and Daniele Micciancio for their careful review, feedback and insightful discussions that helped us to improve the paper.

References

[1] Regev, O. (2009) On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM), **56**(6), 1–40.

[2] Lyubashevsky, V., Peikert, C., and Regev, O. (2013) On ideal lattices and learning with errors over rings. Journal of the ACM (JACM), **60**(6), 1–35.

[3] Gentry, C. (2009) Fully homomorphic encryption using ideal lattices. In Proceedings of the forty-first annual ACM Symposium on Theory of Computing ACM pp. 169–178.

[4] Ducas, L. and Micciancio, D. (2015) FHEW: Bootstrapping homomorphic encryption in less than a second. In Advances in Cryptology – EUROCRYPT 2015 Springer pp. 617–640.

[5] Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. (2020) TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology, **33**(1), 34–91.

[6] Alperin-Sheriff, J. and Peikert, C. (2014) Faster bootstrapping with polynomial error. In Advances in Cryptology – CRYPTO 2014 Springer pp. 297–314.

[7] Gama, N., Izabachene, M., Nguyen, P. Q., and Xie, X. (2016) Structural lattice reduction: generalized worst-case to average-case reductions and homomorphic cryptosystems. In Advances in Cryptology – EUROCRYPT 2016 Springer pp. 528–558.

[8] Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (2014) (Leveled) Fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT), **6**(3), 1–36.

[9] Brakerski, Z. (2012) Fully homomorphic encryption without modulus switching from classical GapSVP. In Advances in Cryptology – CRYPTO 2012 Springer pp. 868–886.

[10] Fan, J. and Vercauteren, F. (2012) Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch., **2012/144**.

[11] Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2017) Homomorphic encryption for arithmetic of approximate numbers. In Advances in Cryptology – ASIACRYPT 2017 Springer pp. 409–437.

[12] Cheon, J. H., Han, K., Kim, A., Kim, M., and Song, Y. (2018) A full RNS variant of approximate homomorphic encryption. In Selected Areas in Cryptography – SAC 2018 Springer pp. 347–368.

[13] Gentry, C., Halevi, S., and Smart, N. P. (2012) Better bootstrapping in fully homomorphic encryption. In Public Key Cryptography – PKC 2012 Springer pp. 1–16.

[14] Halevi, S. and Shoup, V. (2021) Bootstrapping for HElib. Journal of Cryptology, **34**(1), 1–44.

[15] Chen, H. and Han, K. (2018) Homomorphic lower digits removal and improved FHE bootstrapping. In Advances in Cryptology – EUROCRYPT 2018 Springer pp. 315–337.

[16] Cheon, J. H., Han, K., Kim, A., Kim, M., and Song, Y. (2018) Bootstrapping for approximate homomorphic encryption. In Advances in Cryptology – EUROCRYPT 2018 Springer pp. 360–384.

[17] Chen, H., Chillotti, I., and Song, Y. (2019) Improved bootstrapping for approximate homomorphic encryption. In Advances in Cryptology – EUROCRYPT 2019 Springer pp. 34–54.

[18] Han, K. and Ki, D. (2020) Better bootstrapping for approximate homomorphic encryption. In Topics in Cryptology – CT-RSA 2020 Springer pp. 364–390.

[19] Lee, Y., Lee, J.-W., Kim, Y.-S., and No, J.-S. (2020) Near-optimal polynomial for modulus reduction using l2-norm for approximate homomorphic encryption. IEEE Access, **8**, 144321–144330.

[20] Bossuat, J.-P., Mouchet, C., Troncoso-Pastoriza, J., and Hubaux, J.-P. (2021) Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In Advances in Cryptology – EUROCRYPT 2021 Springer.

[21] Lee, J.-W., Lee, E., Lee, Y., Kim, Y.-S., and No, J.-S. (2021) High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function. In Advances in Cryptology – EUROCRYPT 2021 Springer.

[22] Lee, Y., Lee, J., Kim, Y.-S., Kang, H., and No, J.-S. (2020) High-Precision and Low-Complexity Approximate Homomorphic Encryption by Error Variance Minimization. IACR Cryptol. ePrint Arch., **2020/1549**.

[23] Han, K., Hhan, M., and Cheon, J. H. (2019) Improved homomorphic discrete fourier transforms and FHE bootstrapping. IEEE Access, **7**, 57361–57370.

[24] Li, B. and Micciancio, D. (2021) On the security of homomorphic encryption on approximate numbers. In Advances in Cryptology – EUROCRYPT 2021 Springer pp. 648–677.

[25] Chen, H., Dai, W., Kim, M., and Song, Y. (2021) Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In Applied Cryptography and Network Security Springer.

[26] Gentry, C., Sahai, A., and Waters, B. (2013) Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Advances in Cryptology – CRYPTO 2013 Springer pp. 75–92.

[27] Chillotti, I., Gama, N., Georgieva, M., and Izabachene, M. (2017) Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In Advances in Cryptology – ASIACRYPT 2017 Springer pp. 377–408.

[28] Micciancio, D. and Polyakov, Y. (2020) Bootstrapping in FHEW-like Cryptosystems.. IACR Cryptol. ePrint Arch., **2020/86**.

[29] Boura, C., Gama, N., Georgieva, M., and Jetchev, D. (2020) Chimera: Combining Ring-LWE-based fully homomorphic encryption schemes. Journal of Mathematical Cryptology, **14**(1), 316–338.

[30] Lu, W.-j., Huang, Z., Hong, C., Ma, Y., and Qu, H. (2021) PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. In 2021 IEEE symposium on Security and Privacy (S&P) IEEE.

[31] Chillotti, I., Joye, M., and Paillier, P. (2021) Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In International Symposium on Cyber Security Cryptography and Machine Learning Springer pp. 1–19.

[32] Guimarães, A., Borin, E., and Aranha, D. F. (2021) Revisiting the functional bootstrap in TFHE. IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 229–253.

[33] Miccianco, D. and Sorrell, J. (2018) Ring packing and amortized FHEW bootstrapping. In 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018) Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[34] Halevi, S., Polyakov, Y., and Shoup, V. (2019) An improved RNS variant of the BFV homomorphic encryption scheme. In Topics in Cryptology – CT-RSA 2019 Springer pp. 83–105.

[35] Cheon, J. H., Hhan, M., Hong, S., and Son, Y. (2019) A Hybrid of Dual and Meet-in-the-Middle Attack on Sparse and Ternary Secret LWE. IEEE Access,.

[36] Son, Y. and Cheon, J. H. (2019) Revisiting the Hybrid Attack on Sparse Secret LWE and Application to HE Parameters. In Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography Association for Computing Machinery.

[37] Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., and Vaikuntanathan, V., Homomorphic Encryption Security Standard. Technical report, HomomorphicEncryption.org Toronto, Canada (Nov., 2018).

[38] Kim, A., Polyakov, Y., and Zucca, V. (2021) Revisiting Homomorphic Encryption Schemes for Finite Fields. IACR Cryptol. ePrint Arch., **2021/204**.

[39] Brakerski, Z. and Vaikuntanathan, V. (2011) Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In Advances in Cryptology – CRYPTO 2011 Springer pp. 505–524.

[40] Bajard, J.-C., Eynard, J., Hasan, M. A., and Zucca, V. (2016) A full RNS variant of FV like somewhat homomorphic encryption schemes. In Selected Areas in Cryptography – SAC 2016 Springer pp. 423–442.

[41] Kim, A., Papadimitriou, A., and Polyakov, Y. (2020) Approximate homomorphic encryption with reduced approximation error. IACR Cryptol. ePrint Arch., **2020/1118**.

[42] Brakerski, Z. and Vaikuntanathan, V. (2014) Efficient fully homomorphic encryption from (standard) LWE. SIAM Journal on Computing, **43**(2), 831–871.

[43] Shoup, V. NTL: A library for doing number theory. `https://github.com/libntl/ntl` (2001).

[44] Boemer, F., Kim, S., Seifu, G., de Souza, F. D., and Gopal, V. (2021) Intel HEXL: Accelerating Homomorphic Encryption with Intel AVX512-IFMA52. arXiv preprint arXiv:2103.16400,.

[45] Kluczniak, K. and Schild, L. (2021) FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption. arXiv preprint arXiv:2109.02731,.