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

July 14, 2021

Abstract

An approximate homomorphic encryption scheme called CKKS (Cheon-Kim-Kim-Song) is considered one of the most promising fully homomorphic encryption (FHE) schemes since it enables computations on real and complex numbers in encrypted form. Several bootstrapping approaches were proposed for CKKS to refresh a modulus in a ciphertext. However all the existing bootstrapping approaches for CKKS rely on polynomial approximation of a modular reduction function and consequently the quality of a message deteriorates due to errors produced by the polynomial approximation. Also, the polynomial approximation usually consumes a huge number of multiplicative levels. We propose the first bootstrapping approach for the CKKS scheme without polynomial approximation on the modular reduction function. Instead, we adopt a blind rotation technique from FHEW-type schemes and as a result our approach introduces an error which is comparable to a rescaling error while consuming only one multiplicative level. We demonstrate that our bootstrapping procedure can be generalized to the BGV and BFV schemes with minor modifications in the proposed algorithms. We also present several optimizations including a compact representation of public keys required for bootstrapping and a modified blind rotation technique for the case of sparse secret key.

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 [1] (LWE) or Ring Learning with Errors [2] (RLWE) 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 bits from a decrypted result. This is possible since a decryption algorithm outputs a result where message and error parts are separated. Meanwhile, an error is considered part of a 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 for 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 [22] and it causes the modulus of a ciphertext to be much smaller than that of the ciphertext after bootstrapping. 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].

In this paper, we propose a new bootstrapping technique that does not use the 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 [24] to avoid limitations of existing techniques. It produces an error as small as a rescaling error and consumes only one level. As a result we can employ our bootstrapping technique using RLWE ring dimension of 2^{14} or even less, while preserving the same security level.

We show that the new bootstrapping technique is also applicable to BGV and BFV schemes with only minor modifications on the proposed algorithm. This is possible because all 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, which 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 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 $\mathtt{ct}_{\mathtt{boot}} = (\boldsymbol{a}_{\mathtt{boot}}, \boldsymbol{b}_{\mathtt{boot}})$ 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 $\mathtt{ct}_{\mathtt{prep}}$ 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 [24]. This procedure that we call `ScaledMod` gives an encryption $\mathtt{ct}_{\mathtt{sm}}$ 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 [25] 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, 26] proposed a TFHE scheme over the torus and several optimization methods for FHEW. Recently, Micciancio and Polyakov [27] generalized it to unify the original and extended variants of both FHEW and TFHE.

Several recent studies [28, 29, 30, 31, 32] 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 [28, 29]. 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 the full bootstrapping algorithm for CKKS and briefly discuss how it could be generalized to BGV and BFV. The full algorithms of bootstrapping for BGV and BFV are shown in Appendices A and B. In Section 5, we present a compact representation of public keys and how to reconstruct them. We conclude in Section 6.

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 [33].

We use $\boldsymbol{a} \leftarrow \mathsf{S}$ to denote uniform sampling from the set S. We denote sampling according to a distribution χ by $\boldsymbol{a} \leftarrow \chi$. $\chi_{\mathtt{key}}$ denotes ternary distribution such that each coefficient is chosen from $\{-1, 0, 1\}$. $\chi_{\mathtt{key},h}$ denotes ternary distribution with Hamming weight h. $\chi_{\mathtt{err}}$ denotes a discrete Gaussian distribution with a standard deviation $\sigma_{\mathtt{err}}$.

2.1 Basic Lattice-based Encryption

For positive integers q and n, basic LWE encryption of $m \in \mathbb{Z}$ under the secret key $\vec{s}$ 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 $\boldsymbol{s}$ 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. Decryption of ciphertext $\mathtt{ct} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{m}) = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_Q^2$ is done 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 base power gadget vectors $(1, B^1, \ldots, B^{d-1})$ 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 $\hat{q}_j = \prod_{i \neq j} q_i$ 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 [27]. 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

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

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 $\boldsymbol{t}$. 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

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

and

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

The result obtained in the previous equation 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 $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}_1) \odot \mathsf{RGSW}_{\boldsymbol{s}}(\boldsymbol{m}_2) \approx \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2)$, we need the error term $\boldsymbol{e}_1 \cdot \boldsymbol{m}_2$ to be small. This can be achieved by monomials $\pm X^v$ for $\boldsymbol{m}_2$. The multiplication between $\mathsf{RLWE} \odot \mathsf{RGSW}$ is naturally extended to $\mathsf{RLWE}' \odot \mathsf{RGSW}$ by applying $\mathsf{RLWE} \odot \mathsf{RGSW}$ 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 key switching techniques used in the literature and readers can consult the literature such as [34] for more details. We focus on BV key switching type [35] and on its RNS variant [36], as they fit our approach. We define the following key switch generation and key switch algorithms:

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

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

The value $\mathsf{RLWE}'_{\boldsymbol{s}_2}(\boldsymbol{s}_1)$ generated by $\mathtt{KeySwitchGen}$ can be understood as a public key switching key. The key switching error is equal to the error of $\mathcal{R} \times \mathsf{RLWE}'$ multiplication.

Remark. *Key switching usually requires another auxiliary modulus to manage the error. 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

- $\mathtt{EvalAuto}(\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}), t)$: For encryption $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}(X)) = (\boldsymbol{a}(X), \boldsymbol{b}(X))$ of $\boldsymbol{m}$, we apply ψ_t to $\boldsymbol{a}(X)$ and $\boldsymbol{b}(X)$ and obtain $(\boldsymbol{a}(X^t), \boldsymbol{b}(X^t))$, an RLWE encryption of $\boldsymbol{m}(X^t)$ under the secret key $\boldsymbol{s}(X^t)$. Then we perform key switching from $\boldsymbol{s}(X^t)$ to $\boldsymbol{s}(X)$ and output $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}(X^t)) = \mathsf{RLWE}_{\boldsymbol{s}}(\psi_t(\boldsymbol{m}))$.

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. In this paper, we only consider rescaling of $\mathsf{RLWE}_{Q,\boldsymbol{s}}$ instance by q that divides Q. For a $\mathsf{RLWE}_{Q,\boldsymbol{s}}$ instance $(\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_Q^2$, the rescaling by $q|Q$ is as follows:

- $\mathtt{Rescale}((\boldsymbol{a}, \boldsymbol{b}), q) = \left(\left\lfloor \frac{\boldsymbol{a}}{q} \right\rceil, \left\lfloor \frac{\boldsymbol{b}}{q} \right\rceil \right) \in \mathcal{R}_{Q/q}^2$.

For $\mathtt{ct} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{m})$ we have $\mathtt{Rescale}(\mathtt{ct}, q) = \mathtt{ct}_{\mathtt{rs}} = \mathsf{RLWE}_{Q/q,\boldsymbol{s}}(\frac{1}{q}\boldsymbol{m})$. The rescaling procedure also divides the error of $\mathtt{ct}$ by q, but introduces additional rescaling error $\boldsymbol{e}_{\mathtt{rs}}$. The rescaling error $\boldsymbol{e}_{\mathtt{rs}}$ however is small [37] and it's norm $\|\boldsymbol{e}_{\mathtt{rs}}\|_\infty$ is bounded by $\frac{1}{2}(1+\delta_{\mathcal{R}})$ for ternary secret key.

2.5 RLWE based schemes

In this subsection 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:

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

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:

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

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:

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

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 $\|\boldsymbol{u}\|_\infty \le c < N/2$, 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 $\|\boldsymbol{u}\|_\infty \le c < N/2$ 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. `Extract`

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. `BlindRotate`

`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 $\|\boldsymbol{u}\|_\infty \le c < N/2$. 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 `ACC` as

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

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 BLINDROTATE($\mathtt{f}, (\vec{\alpha}, \beta), \mathtt{brk}$)
  $\mathtt{ACC} \leftarrow \left(0, \mathtt{f}\cdot X^\beta\right)$
  for $i = 0, ..., N-1$ **do**
    $\mathtt{ACC} \leftarrow \mathtt{ACC} \odot \left(\mathsf{RGSW}^0(1) + (X^{\alpha_i} - 1)\cdot \mathsf{RGSW}(s_i^+) + (X^{-\alpha_i} - 1)\cdot \mathsf{RGSW}(s_i^-)\right)$
return `ACC`

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

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

Error Analysis Let $E_{\mathtt{err}}$ denote the high-probability upper bound of error in an RLWE encryption, for instance, $6\sigma_{\mathtt{err}}$ [11]. The error bound of each RLWE element of the RGSW encryption in equation (2) can be bounded by $4E_{\mathtt{err}}$, so each blind rotation step introduces an additive error that is bounded by $4dBE_{\mathtt{err}}$, where B is a digit bound, and d is a number of digits of gadget decomposition. Therefore the total error after blind rotations is bounded by $\|\boldsymbol{e}_{\mathtt{br}}\|_\infty < E_{\mathtt{br}} = 4dNBE_{\mathtt{err}}$.

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 rescale every time as in usual key-switching in [36, 38]. Instead, we can postpone rescaling to the end to reduce the complexity.*

Step 2′. `BlindRotate` **for sparse ternary secret key**

For a sparse ternary secret $\boldsymbol{s} \leftarrow \chi_{\mathtt{key},h}$, we can take advantage of the knowledge that only h coefficients of $\boldsymbol{s}$ are non-zero and decrease the computational complexity of blind rotation. Suppose $(s_{i_1}, \ldots, s_{i_h})$ are non-zero coefficients of $\boldsymbol{s}$. We initially generate blind rotation public keys

$$\mathtt{brk}' = \left\{ \mathsf{RGSW}_{Q,\boldsymbol{s}}\left(\delta^+_{\ell,j}\right), \mathsf{RGSW}_{Q,\boldsymbol{s}}\left(\delta^-_{\ell,j}\right) \right\}_{(\ell,j)\in[1,h]\times[0,N-1]}$$

where

$$\begin{cases} \delta^+_{\ell,j} = 1, & \text{if } i_\ell = j \ \&\ s_{i_\ell} = 1 \\ \delta^+_{\ell,j} = 0, & \text{otherwise} \end{cases}, \quad \begin{cases} \delta^-_{\ell,j} = 1, & \text{if } i_\ell = j \ \&\ s_{i_\ell} = -1 \\ \delta^-_{\ell,j} = 0, & \text{otherwise} \end{cases}$$

for $(\ell, j) \in [1, h] \times [0, N-1]$.

In the blind rotation, we iteratively compute

$$\mathsf{RGSW}(X^{\alpha_{i_\ell} \cdot s_{i_\ell}}) = \sum_{j=0}^{N-1} X^{\alpha_j} \cdot \mathsf{RGSW}\left(\delta^+_{\ell,j}\right) + \sum_{j=0}^{N-1} X^{-\alpha_j} \cdot \mathsf{RGSW}\left(\delta^-_{\ell,j}\right) \tag{3}$$

and update `ACC` as

$$\mathtt{ACC} \leftarrow \mathtt{ACC} \odot \mathsf{RGSW}_{Q,\boldsymbol{s}}(X^{\alpha_{i_\ell} \cdot s_{i_\ell}}).$$

The equation (3) is correct since only one of summands is non-zero. This alternative blind rotation decreases the number of RLWE $\odot$ RGSW multiplications from N to h, but increases the size of the blind rotation keys from $2N$ to $2hN$ of RGSW encryptions.

Error Analysis The error bound of each RLWE element of RGSW encryption in equation (3) can be bounded by $2NE_{\mathtt{err}}$, so each blind rotation step introduces an additive error that is bounded by $2dBNE_{\mathtt{err}}$. The total error after blind rotations is bounded by $2dBhNE_{\mathtt{err}}$.

Algorithm 2 Blind rotation for sparse secret key

procedure $\texttt{BlindRotate}'(\texttt{f}, (\vec{\alpha}, \beta), \texttt{brk}')$

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

for $l = 1, ..., h$ **do**

$\texttt{ACC} \leftarrow \texttt{ACC} \odot \left(\sum_{j=0}^{N-1} X^{\alpha_j} \cdot \mathsf{RGSW}\left(\delta_{\ell,j}^{+}\right) + \sum_{j=0}^{N-1} X^{-\alpha_j} \cdot \mathsf{RGSW}\left(\delta_{\ell,j}^{-}\right)\right)$

return ACC

Step 3. `Repack`

After `BlindRotate`, we receive N 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 be the smallest power of two satisfying $n > 2c$ for some c defined in the initial function $\texttt{f}$. 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 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 [24]. 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 focus on the coefficients of X^{nk} and $X^{nk+\frac{n}{2}}$ and we 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 follows

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

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 3.

Algorithm 3 Repacking

procedure $\textsc{Repack}(\{\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i)})\}_{i\in[0,N-1]}, n)$ ▷ n is a power of two such that $2c < n \le N$
  for $i = 0, \dots, n-1$ **do**
    $\mathtt{ct}^{(i,n)} \leftarrow \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i)})$
    for $j = 1, \dots, \frac{N}{n} - 1$ **do**
      $\mathtt{ct}^{(i,n)} \leftarrow \mathtt{ct}^{(i,n)} + X^{nj} \cdot \mathtt{ct}^{(i+nj)}$
  while $n > 1$ **do**
    for $i = 0, \dots, \frac{n}{2} - 1$ **do**
      $\mathtt{ct}^{(i,\frac{n}{2})} \leftarrow \mathtt{ct}^{(i,n)} + X^{\frac{n}{2}} \cdot \mathtt{ct}^{(i+\frac{n}{2},n)}$
      $\mathtt{ct}^{\mathtt{rot}} \leftarrow \mathtt{EvalAuto}\left(\mathtt{ct}^{(i,n)} - X^{\frac{n}{2}} \cdot \mathtt{ct}^{(i+\frac{n}{2},n)}, 1 + \frac{2N}{n}\right)$
      $\mathtt{ct}^{(i,\frac{n}{2})} \leftarrow \mathtt{ct}^{(i,\frac{n}{2})} + \mathtt{ct}^{\mathtt{rot}}$
    $n \leftarrow \frac{n}{2}$
  return $\mathtt{ct}^{(0,1)} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(n \cdot \Delta \cdot \boldsymbol{u})$

Error Analysis The first step of repacking adds the errors from the blind rotations, so we can bound the error of each $\mathsf{RLWE}_{Q,\boldsymbol{s}}\left(\boldsymbol{u}^{(i,n)}\right)$ by $\frac{N}{n}E_{\mathtt{br}}$. For the second step every EvalAuto introduces new error from $\mathcal{R}$ by RLWE' multiplications, which is bounded by $\frac{dB}{2}E_{\mathtt{err}}$. The total error after repacking is bounded by $E_{\mathtt{sm}} = NE_{\mathtt{br}} + (n-1)\frac{dB}{2}E_{\mathtt{err}}$.

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 the bootstrapping technique for the BGV and BFV schemes as subsequent results.

4.1 Bootstrapping for CKKS

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

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

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. As we mentioned before, a major drawback of this approach is that it is hard to approximate a modular reduction function with a polynomial. The ciphertext after bootstrapping usually obtains a large noise, which reduces the quality of the encrypted message.

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 $q \cdot \boldsymbol{v}$. Our bootstrapping uses the `ScaledMod` as a sub-algorithm and it only adds a noise comparable with a rescaling noise, which preserves the quality of the encrypted message. As a first step, the ciphertext is preprocessed to contains 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 $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ for a small modulus q, the decryption query is described as

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

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

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

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 $\mathtt{ct}_{\mathtt{prep}} = \mathsf{RLWE}_{2N,\boldsymbol{s}}^0(-\boldsymbol{u})$, 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 `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 4.

Algorithm 4 Bootstrapping for CKKS

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

$\mathtt{Preprocess}(\mathtt{ct}) \to \mathtt{ct}', \mathtt{ct}_{\mathtt{prep}}$ ▷ $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + q \cdot \boldsymbol{v} \in \mathcal{R}$

- $\mathtt{ct}' \leftarrow \mathtt{ct} \pmod{q'}$ ▷ $\mathtt{ct}'(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + q' \cdot \boldsymbol{u} \in \mathcal{R}$
- $\mathtt{ct}_{\mathtt{prep}} \leftarrow \left(\frac{\mathtt{ct} - \mathtt{ct}'}{q'}\right)$ ▷ $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = -\boldsymbol{u} \pmod{2N}$

$\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q', Q) \to \mathtt{ct}_{\mathtt{sm}}$ ▷ $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = -q' \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Q}$

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

- $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}' \pmod{Q}$ ▷ $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Q}$

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 values where $n < \frac{N}{2}$. The main idea is to reduce the number of coefficients of $\boldsymbol{u}$ which will be inputs of the blind rotations in `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 `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 5. 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 $N/2n$ 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 `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 `ScaledMod`, we can evaluate the blind rotations only for the subset of coefficients $\left\{u_{(N/2n)\cdot i}\right\}$ 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 `Repack` algorithm. The output of the variant `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

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

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

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

Algorithm 5 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 $\mathtt{ct}''$

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

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

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 6.

4.2 Bootstrapping for BGV and BFV

Now we explain how our technique can be applied to BGV and BFV schemes. Since BGV, BFV, and CKKS schemes have a similar cryptographic structure and their encryption only differs in encoding, the `Preprocess` and `Combine` procedures are performed with slight modifications. In this subsection, we only describe the different parts briefly and the full algorithms are presented in Appendices A and B. Both BGV and BFV schemes have a plaintext space $\mathcal{R}_t$ for some t which is normally taken as a prime power and in our cases, we take t to be coprime with 2 for simplicity.

Algorithm 6 Bootstrapping for RNS-CKKS

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

$\mathtt{Preprocess}(\mathtt{ct}) \to \mathtt{ct}', \mathtt{ct}_{\mathtt{prep}}$ ▷ $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + q \cdot \boldsymbol{v} \in \mathcal{R}$

- $\mathtt{ct}' \leftarrow 2N \cdot \mathtt{ct} \pmod q$ ▷ $\mathtt{ct}'(\boldsymbol{s}) = 2N \cdot \boldsymbol{m} + 2N \cdot \boldsymbol{e} + q \cdot \boldsymbol{u} \in \mathcal{R}$
- $\mathtt{ct}_{\mathtt{prep}} \leftarrow \left(\frac{2N \cdot \mathtt{ct} - \mathtt{ct}'}{q}\right)$ ▷ $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = -\boldsymbol{u} \pmod{2N}$

$\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q, Qp) \to \mathtt{ct}_{\mathtt{sm}}$ ▷ $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = -q \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Qp}$

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

- $\mathtt{ct}'' \leftarrow \mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}' \pmod{Qp}$ ▷ $\mathtt{ct}''(\boldsymbol{s}) = 2N \cdot \boldsymbol{m} + 2N \cdot \boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Qp}$
- $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{Rescale}\left(\frac{p}{2N} \cdot \mathtt{ct}'', p\right)$ ▷ $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{m} + \boldsymbol{e} + \frac{1}{2N} \cdot \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{e}_{\mathtt{rs}} \pmod{Q}$

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

4.2.1 BGV

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 multiprecision CKKS with the only difference that secret and public keys are generated with errors of the form $t \cdot \boldsymbol{e}$ instead of $\boldsymbol{e}$ and the automorphism and rescale in `Repack` procedure are evaluated in accordance with BGV style. The bootstrapping algorithm for RNS-BGV is also similar to that for RNS-CKKS, with the only difference at `Combine` step, where division by $2N$ is done modulo t.

4.2.2 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 `ScaledMod` and rescaling.

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 dimension of ring 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 party which performs all the computations. The computational party also can have a limitation in storing 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 party reconstructs the keys and performs blind rotations on the fly, without storing the whole keys.

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

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

The reconstruction of $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^\pm)$ can be done in parallel by using divide and conquer algorithm described in Algorithm 7. The reconstruction of $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^\pm \cdot \boldsymbol{s})$ can be done by observing that for each $\mathsf{RLWE}_{\boldsymbol{s}}(g_j \cdot s_i^\pm) = (\boldsymbol{a}_{i,j}, \boldsymbol{b}_{i,j})$, the reconstruction of $\mathsf{RLWE}_{\boldsymbol{s}}(g_j \cdot s_i^\pm \cdot \boldsymbol{s})$ is as follows:

$$\boldsymbol{a}_{i,j} \odot \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2) + \boldsymbol{b}_{i,j} \cdot (1,0) = \mathsf{RLWE}_{\boldsymbol{s}}\left(\boldsymbol{a}_{i,j} \cdot \boldsymbol{s}^2 + \boldsymbol{b}_{i,j} \cdot \boldsymbol{s}\right) = \mathsf{RLWE}_{\boldsymbol{s}}(g_j \cdot s_i^\pm \cdot \boldsymbol{s}). \tag{4}$$

After repeating this procedure for all $j \in [0, d-1]$ and $i \in [0, N-1]$, 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 from $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^\pm)$ and $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$.

Algorithm 7 Reconstruct and store

procedure RECONSTRUCTION($\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^\pm)$)
  $S_0^\pm \leftarrow \mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^\pm)$
  for $(n = N; n > 1; n = n/2)$ **do**
    for $(i = 0; i < N; i = i + n)$ **do**
      $T_i^\pm \leftarrow \texttt{EvalAuto}(S_i^\pm, n+1)$
      $S_i^\pm = S_i^\pm + T_i^\pm$
      $S_{i+n/2}^\pm = X^{-N/n} \cdot (S_i^\pm - T_i^\pm)$
return $\left\{S_i^\pm\right\} = \left\{\mathsf{RLWE}'(N \cdot s_i^\pm)\right\}$

$\texttt{EvalAuto}(\mathsf{RLWE}'(\cdot), \cdot)$ in Algorithm 7 denotes the operation of performing the same $\texttt{EvalAuto}$ for all the RLWE elements in $\mathsf{RLWE}'(\cdot)$. $\texttt{EvalAuto}$ 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 coefficient, which is an encryption of $s_i^\pm$.

In Algorithm 7, 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.

Error Analysis The error bound of each RLWE element of RLWE' encryption after reconstruction can be bounded by $E_{\mathtt{rc}} = NE_{\mathtt{err}} + (N-1)\frac{dB}{2}E_{\mathtt{err}}$. As the errors of reconstructed $\mathsf{RGSW}_{Q,\boldsymbol{s}}(s_i^\pm)$ are bigger than fresh errors of $\mathsf{RGSW}_{Q,\boldsymbol{s}}(s_i^\pm)$, we can use larger auxiliary modulus to make the error negligible.

5.2 Performing blind rotations on the fly

To prevent storing all blind rotation keys on computational side, we can reconstruct blind rotation key for each blind rotation step i 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 8 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 8 Reconstruct on the fly

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

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

$$\mathsf{RLWE}'_{\boldsymbol{s}}(X^{\alpha_i \cdot s_i}) = \mathsf{RLWE}'^{0}(1) + (X^{\alpha_i} - 1) \cdot \mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{+}) + (X^{-\alpha_i} - 1) \cdot \mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{-}). \tag{5}$$

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 $\mathtt{ACC}$

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

Error Analysis The error of each RLWE element of RLWE' encryption in equation (5) can be bounded by $4E_{\mathtt{rc}}$. Hence, each $\mathcal{R}$ by RLWE' for $\boldsymbol{a}$ and $\boldsymbol{b}$ adds an error which is bounded by $2dBE_{\mathtt{rc}}$, each blind rotation step introduces additive error bounded by at most $2(N+1)dBE_{\mathtt{rc}} + \frac{dB}{2}E_{\mathtt{err}}$.

6 Conclusion

We proposed the first bootstrapping procedure for the CKKS scheme without approximating the modular reduction function, and extended it to BGV and BFV schemes. Our bootstrapping procedure uses the blind rotation technique and it introduces small rescaling errors instead of big approximation errors as in previous CKKS bootstrapping methods. We also modified the blind rotation algorithm for a sparse secret key to reduce computational complexity.

Due to the large size of blind rotation keys, we introduced a method of extracting all blind rotation keys on a computational side rather than generating and transferring all of them from a secret key holder side. In addition, we proposed a method of evaluating the blind rotation on the fly without storing all the keys on the computational side.

In contrast to previous bootstrapping methods, our bootstrapping requires only one rescaling and thus it can be implemented with smaller parameters for the same security level. We plan to implement and experiment our approach on different computing platforms including systems with limited memory. We will also continue research on possible optimizations of our new bootstrapping approach.

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

[25] 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.

[26] 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.

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

[28] 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.

[29] 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) (to appear) IEEE.

[30] Chillotti, I., Joye, M., and Paillier, P. (2021) Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. IACR Cryptol. ePrint Arch., **2021/091**.

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

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

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

[39] Gentry, C., Halevi, S., and Smart, N. P. (2012) Homomorphic evaluation of the AES circuit. In Advances in Cryptology – CRYPTO 2012 Springer pp. 850–867.

A Bootstrapping for BGV

A.1 Multiprecision BGV

Our bootstrapping for multiprecision BGV is quite similar to multiprecision CKKS bootstrapping in Section 4, with the difference that the blind rotation keys $\mathsf{RGSW}_{Q,\boldsymbol{s}}(s_i^{\pm})$ are generated with errors of type $t \cdot \boldsymbol{e}$ instead of $\boldsymbol{e}$.

Assume that we have a ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ 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 consider $\mathtt{ct}' = \mathtt{ct} \pmod{q'} = \left([\boldsymbol{a}]_{q'}, [\boldsymbol{b}]_{q'}\right) \in \mathcal{R}_{q'}^2$. Similar to CKKS, we assume that $\|\boldsymbol{m} + t \cdot \boldsymbol{e}\|_\infty \le \gamma < \frac{q}{4} - \frac{q'}{2} \cdot (\delta_{\mathcal{R}} + 1)$ for some γ. Hence,

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

where

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

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

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

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

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

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 9.

Algorithm 9 Bootstrapping for BGV

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

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

return $\mathtt{ct}_{\mathtt{boot}}$

A.2 RNS-BGV

Bootstrapping for RNS-BGV is also similar to RNS-CKKS bootstrapping in Section 4, with the only difference at `Combine` step, where division is done by $2N$ modulo t. We again start with $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ such that

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

We assume that $\|\boldsymbol{m} + t \cdot \boldsymbol{e}\|_\infty \leq \gamma < \frac{q}{4} - \frac{q}{4N} \cdot (\delta_\mathcal{R} + 1)$ 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

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

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

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

For preprocessed $\mathtt{ct}_{\mathtt{prep}} = \mathsf{RLWE}^0_{2N,\boldsymbol{s}}(-\boldsymbol{u})$ 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

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

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

$$\mathtt{ct}''(\boldsymbol{s}) = \boldsymbol{a}'' \cdot \boldsymbol{s} + \boldsymbol{b}'' = 2N \cdot \boldsymbol{m} + 2Nt \cdot \boldsymbol{e} + q \cdot \boldsymbol{u} - q \cdot \boldsymbol{u} + t \cdot \boldsymbol{e}_{\mathtt{sm}} = [2N \cdot \boldsymbol{m}]_t + t \cdot \boldsymbol{r} + 2Nt \cdot \boldsymbol{e} + t \cdot \boldsymbol{e}_{\mathtt{sm}} \pmod Q,$$

where $\boldsymbol{r} = \frac{1}{t} \cdot (2N \cdot \boldsymbol{m} - [2N \cdot \boldsymbol{m}]_t)$. To get rid of scaling factor $2N$ from the message, we multiply $\mathtt{ct}''$ by $[(2N)^{-1}]_t$, and obtain $\mathtt{ct}_{\mathtt{boot}}$ satisfying

$$\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{m} + t \cdot [(2N)^{-1}]_t \cdot (2N \cdot \boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{r}) \pmod Q.$$

Let $e' = [(2N)^{-1}]_t \cdot (2N \cdot \boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{r})$ and $\mathtt{ct}_{\mathtt{boot}}$ is an RNS-BGV encryption of $\boldsymbol{m}$ with noise $t \cdot \boldsymbol{e}'$ and modulus Q. The full algorithm is described in Algorithm 10.

Remark. *The latest division by $2N$ modulo t is optional, otherwise we can update the scaling factor of the message in the ciphertext by $2N$ instead [34, 39].*

Algorithm 10 Bootstrapping for RNS-BGV

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

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

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

B Bootstrapping for BFV

B.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 $t \cdot \boldsymbol{e} \leq \gamma < \frac{Q}{4} - \frac{Q'}{2} \cdot (\delta_{\mathcal{R}} + 1)$ 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 preprocess ciphertext $\mathtt{ct}_{\mathtt{prep}} = \frac{1}{Q'} \cdot (\mathtt{ct}' - \mathtt{ct}'')$ with

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

After applying $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q', Qt)$ for $c = \left\lfloor \frac{1}{2}(1 + \delta_{\mathcal{R}}) + \frac{\gamma}{Q'} \right\rfloor$, we have $\mathtt{ct}_{\mathtt{sm}}$ satisfying

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

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 11.

B.2 RNS-BFV

Bootstrapping for RNS-BFV scheme 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 $\mathtt{ct}'' = 2N \cdot \mathtt{ct}' \pmod{Q}$, then we have

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

Algorithm 11 Bootstrapping for BFV

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

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

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

$\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q', Qt) \to \mathtt{ct}_{\mathtt{sm}}$ ▷ $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = Q' \cdot \boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Qt}$

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

- $\mathtt{ct}''' \leftarrow \mathtt{ct}_{\mathtt{sm}} + t \cdot \mathtt{ct} - \mathtt{ct}'' \pmod{Qt}$ ▷ $\mathtt{ct}'''(\boldsymbol{s}) = \boldsymbol{e}_{\mathtt{sm}} + Q \cdot \boldsymbol{m} \pmod{Qt}$
- $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{Rescale}(\mathtt{ct}''', t)$ ▷ $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \frac{1}{t} \cdot \boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{e}_{\mathtt{rs}} + \frac{Q}{t} \cdot \boldsymbol{m} \pmod{Q}$

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

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

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

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

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

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

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

Finally we multiply $\mathtt{ct}'''$ by $\frac{p}{2N}$, and rescale the result by p, then obtain $\mathtt{ct}_{\mathtt{boot}} = \mathtt{Rescale}(\frac{p}{2N} \cdot \mathtt{ct}''', pt)$ 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 12.

Algorithm 12 Bootstrapping for RNS-BFV

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

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

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

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

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

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

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