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

May 25, 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 the modulus in a ciphertext. However all the existing bootstrapping approaches for CKKS rely on polynomial approximation of modulus reduction function and consequently, the quality of the message deteriorates due to polynomial approximation errors. We propose the first bootstrapping approach for the CKKS scheme without polynomial approximation of the modulus reduction function. Instead, our procedure adopts blind rotation technique from FHEW-type schemes and as a result, our approach introduces only an error that is comparable to a rescaling error. We also present several optimizations including compact representation of public keys required for bootstrapping and modified blind rotation technique for the case of sparse secret key. We demonstrate that our bootstrapping procedure can be generalized to the BGV and BFV schemes with minor modifications in the proposed algorithms.

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 the secret key. Most HE schemes rely on Learning with Errors [1] (LWE) or Ring Learning with Errors [2] (RLWE) problems, and their ciphertexts contain small "noise" which ensures security. However the noise grows during computations and eventually can destroy the message, and thus we are limited in number of operations which can be computed in encrypted form. Since the first construction of the Fully Homomorphic Encryption (FHE) scheme by Gentry [3], a considerable progress has been made in the direction of research on HE. Gentry's celebrated idea of bootstrapping allows one to refresh the noise in ciphertexts and to be able to do more computations on ciphertexts.

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 the so-called blind rotation technique [6, 7]. BGV/BFV-type schemes [8, 9, 10] are commonly designed to perform 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 only differs in encodings, the existing bootstrapping algorithms for all three schemes are very different.

All the previous bootstrapping algorithms for the CKKS scheme such as [13, 14, 15, 16, 17, 18, 19] are based on polynomial approximation of the modulus reduction function. In CKKS, a noise is considered a part of the message so that it is crucial to keep the noise small during the computations. Therefore, the size of the noise added after bootstrapping has a significant impact on the quality of the message. Besides, previous bootstrapping algorithms consume significant amount of levels due to the approximate polynomial evaluation [19]. Hence, the modulus of the ciphertext after bootstrapping is much smaller than that of a fresh ciphertext so that huge RLWE parameters are needed to make bootstrapping feasible. In practice the ring size of RLWE is set to 2^{16} or higher for security purpose.

1.1 Contribution

In this paper, we propose a new bootstrapping technique for the CKKS scheme. Our technique differs greatly from previous bootstrapping methods which use polynomial approximation for the modulus reduction function. Our approach takes advantage of the blind rotation technique used in the FHEW-type bootstrapping algorithm [4, 5] and repacking technique for RLWE ciphertexts [20]. The error size after our bootstrapping increases only by a rescaling error which could be considered negligible compared to the size of the message. Also, unlike previous techniques, our approach consumes only one level. As a result we can employ our bootstrapping technique using RLWE ring size 2^{14} or even less, while preserving the same security level.

We show that our new bootstrapping technique is also applicable to BGV and BFV schemes with only minor modifications in the proposed algorithm. We also provide a technique of reconstructing most of the public keys on the computational side instead of generating and transferring them from the secret key holder side. The reconstruction of the public keys also could be done on the fly without storing all of them on the computational side. Finally we provide a modified blind rotation technique for a sparse secret key with reduced computational complexity.

1.2 Related Works

Both BGV and BFV are exact schemes and the error accumulated during the computations can eventually destroy the message. All known bootstrapping techniques for BGV and BFV schemes and their RNS variants [21, 22, 23] are performed by extracting bits from decrypted result. This is available since decryption algorithm outputs the result where message part and error part are separated.

Meanwhile, in CKKS an error is considered as a part of a message so that the bit extraction technique cannot be applied. Cheon et al. [13] proposed the first bootstrapping procedure for CKKS based on the polynomial approximation of the modular reduction function in decryption algorithm. Subsequent studies have focused on approximating the modulus function more precisely to improve accuracy [14, 15, 16, 17, 18, 19]. However, the accuracy remains a major concern of bootstrapping in CKKS.

Ducas and Micciancio [4] introduced a blind rotation technique based on RGSW [24] and achieved the bootstrapping time in less than a second for evaluation of boolean operation 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, 25] proposed a TFHE scheme over the torus and several optimization methods for FHEW. Recently, Micciancio and Polyakov [26] generalized it to unify the original and extended variants of both FHEW and TFHE.

Several recent studies [27, 28, 29, 30, 31] applied blind rotation in combination with other FHE schemes. In [27, 28], it is shown that conversions between LWE and RLWE in combination with blind rotation can be an efficient basis for evaluation of non-polynomial functions for FHE such as sign function and other neural network activation functions. However, none of them considered it for 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 for our bootstrappings and in Section 4, we introduce several optimizations. In Section 5, we describe our 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 included in Appendices A and B. 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.

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 [32], the bound $\delta_\mathcal{R} = 2\sqrt{N}$ is close to what we observe experimentally.

We use $\boldsymbol{a} \leftarrow \mathsf{S}$ to denote uniform sampling from the set S. For a distribution χ, we denote sampling according to it by $\boldsymbol{a} \leftarrow \chi$. $\chi_{\mathtt{key}}$ denotes ternary distribution such that each coefficient is chosen from $\{-1, 0, 1\}$. $\chi_{\mathtt{err}}$ 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 will use base power gadget vectors $(1, B^1, \ldots, B^{d-1})$ with a power of two B. We will 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 [26]. 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

The key switching operation converts ciphertext $\mathsf{RLWE}_{\boldsymbol{s}_1}(\boldsymbol{m})$ encrypted using secret key $\boldsymbol{s}_1$ to 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 [33] for more details. We focus on BV key switching type [34] and on its RNS variant [35], as they fit our approach. We define the following key switch generation and key switch algorithms:

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

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

The value $\mathsf{RLWE}'_{\boldsymbol{s}_2}(\boldsymbol{s}_1)$ generated by `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 will 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

- $\texttt{EvalAuto}(\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}), t)$: 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:

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

For $\texttt{ct} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{m})$ we have $\texttt{Rescale}(\texttt{ct}, q) = \texttt{ct}_{\texttt{rs}} = \mathsf{RLWE}_{Q/q,\boldsymbol{s}}(\frac{1}{q}\boldsymbol{m})$. The rescaling procedure also divides the error of $\texttt{ct}$ by q, but introduces additional rescaling error $\boldsymbol{e}_{\texttt{rs}}$. The rescaling error $\boldsymbol{e}_{\texttt{rs}}$ however is small [36] and is bounded by $\frac{1}{2}(1 + \delta_{\mathcal{R}})$ for ternary secret key.

2.5 RLWE based schemes

We will briefly mention encryption procedures for three most common FHE schemes based on RLWE. The main difference of encryption in all of 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:

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

3 Scaled Modulus Raising

In this section, we present `ScaledMod` - the core algorithm used in our bootstrapping in Section 5. The algorithm transforms $\mathsf{RLWE}^0_{2N,\boldsymbol{s}}(\boldsymbol{u})$, where $\|\boldsymbol{u}\| \leq c$ for some $c \in \mathbb{Z}$, to $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\Delta \cdot \boldsymbol{u})$ for a scaling factor Δ and a large modulus Q.

We first extract $\mathsf{LWE}^0_{2N,\vec{s}}(u_i)$ ciphertexts from an $\mathsf{RLWE}^0_{2N,\boldsymbol{s}}(\boldsymbol{u})$ ciphertext. Then for each extracted LWE ciphertext, we perform the blind rotation with the initial function $\texttt{f} = -\sum_{j=-c}^{c} \Delta \cdot j \cdot X^j \in \mathcal{R}_Q$, 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 $\Delta \cdot \boldsymbol{u}$.

3.1 `ScaledMod` procedure

The flow of the proposed `ScaledMod` procedure is as follows:

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

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

We describe each part of the `ScaledMod` algorithm in detail.

- `Extract`

Let us start with a pair $(\boldsymbol{a}, \boldsymbol{b}) = \mathsf{RLWE}^0_{2N,\boldsymbol{s}}(\boldsymbol{u})$. Because 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 follows.

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

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

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

- `BlindRotate`

For an $\mathsf{LWE}^0_{2N,\vec{s}}(u) = (\vec{\alpha}, \beta)$ where $|u| < c$, we start a blind rotation with $\texttt{ACC} = \mathsf{RLWE}^0_{Q,\boldsymbol{s}}(\mathtt{f} \cdot X^{\beta})$. We generate blind rotation public keys $\texttt{brk} = \left\{\mathsf{RGSW}_{Q,\boldsymbol{s}}(s_i^+), \mathsf{RGSW}_{Q,\boldsymbol{s}}(s_i^-)\right\}$, where

$$\begin{cases} s_i^+ = 1, & s_i = 1 \\ s_i^+ = 0, & \text{otherwise} \end{cases}, \quad \begin{cases} s_i^- = 1, & s_i = -1 \\ s_i^- = 0, & \text{otherwise} \end{cases} \quad \text{for } i \in [0, N-1].$$

Then we iteratively compute

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

and update `ACC` as

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

The equation (1) 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

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

As we have chosen $\mathtt{f} = -\sum_{j=-c}^{c} \Delta \cdot j \cdot X^j$, the polynomial $\boldsymbol{u}_{\mathtt{f}}$ has $\Delta \cdot u$ as its constant term. The full algorithm is described in Algorithm 1.

We apply blind rotation algorithm to all $\mathsf{LWE}^0_{2N,\vec{s}}(u_i)$ and obtain

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

Algorithm 1 Blind rotation

procedure $\texttt{BlindRotate}(\mathtt{f}, (\vec{\alpha}, \beta), \mathtt{brk})$
  $\mathtt{ACC} \leftarrow \left(0, \mathtt{f} \cdot X^{\beta}\right)$
  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

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 $|u| < c$, 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, $6\sigma_{\mathtt{err}}$. The error bound of each RLWE element of RGSW encryption in equation (1) 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. So the total error after blind rotation is bounded by $\|\boldsymbol{e}_{\mathtt{br}}\| < E_{\mathtt{br}} = 4dNBE_{\mathtt{err}}$.

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 [35, 37]. Instead, we can postpone rescaling to the end to reduce the complexity.*

- `Repack`

Let n be the smallest power of two satisfying $n > 2c$. Given $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i)})$ for $i \in [0, N-1]$, `Repack` algorithm is performed in two steps.

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

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

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

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

In a similar way, we pack subsets $\left\{\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i+nk)})\right\}$ for $i \in [1, n-1]$ into $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i,n)})$ where $\boldsymbol{u}^{(i,n)}$ has coefficients $\Delta \cdot u_{i+nk}$ at X^{nk}.

For the second step we adapt the repacking technique from [20]. Consider a pair $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(0,n)})$ and $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(n/2,n)})$. Notice that the automorphism $\psi_{1+2N/n}$ applied to $\boldsymbol{u}^{(0,n)}$ preserves all

coefficients at X^{nk}, for $k \in [0, N/n-1]$, changes the sign of coefficients at $X^{nk+n/2}$, and shuffles the other coefficients with possible changes in sign. We focus on the coefficients of X^{nk} and $X^{nk+n/2}$ 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 $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(n/2,n)})$ as follows

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

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

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

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

Algorithm 2 Repacking

procedure REPACK($\{\mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i)})\}_{i=0}^{N-1}$, power of two n such that $N > n > 2c$)
  for $i = 0, \ldots, n-1$ **do**
    $C^{(i,n)} \leftarrow \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{u}^{(i)})$
    for $j = 1, \ldots, \frac{N}{n} - 1$ **do**
      $C^{(i,n)} \leftarrow C^{(i,n)} + X^{nj} \cdot C^{(i+nj)}$
  while $n \geq 1$ **do**
    for $i = 0, \ldots, \frac{n}{2} - 1$ **do**
      $C^{(i,n/2)} \leftarrow C^{(i,n)} + X^{n/2} \cdot C^{(i+n/2,n)}$
      $C^{\mathtt{rot}} \leftarrow \mathtt{EvalAuto}\left(C^{(i,n)} - X^{n/2} \cdot C^{(i+n/2,n)}, 1 + \frac{2N}{n}\right)$
      $C^{(i,n/2)} \leftarrow C^{(i,n/2)} + C^{\mathtt{rot}}$
    $n \leftarrow n/2$
return $C^{(0,1)} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(n \cdot \Delta \cdot \boldsymbol{u})$

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' multiplication, 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. *Note that 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 we accumulate the error during the blind rotations and repacking procedures. By modifying the initial state, we can address the first issue. We start with $[n]_Q^{-1}\Delta$ when Q is coprime with n. When Q is a power of two, we start with RLWE modulus $Q \cdot n$ and then rescale by n. For the second problem we use the 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. For this 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 Optimizations

The huge size of the blind rotation keys causes the following two problems. First, it is difficult for the secret key holder to transfer the heavy blind rotation keys to the computational party which performs all the computations. Second, the computational party also can have a limitation in storing all the blind rotation keys.

In this section, we provide several optimization methods for blind rotation. 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 on the secret key holder side. We also present a method that the computational party can generate and perform blind rotations on the fly, without storing the whole keys. Finally, we provide an alternative blind rotation to reduce the number of multiplications for sparse secret key.

4.1 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 3. 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:

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

As a result, we can reconstruct $\mathsf{RGSW}_{\boldsymbol{s}}(s_i^\pm) = \left(\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^\pm), \mathsf{RLWE}'_{\boldsymbol{s}}(s_i^\pm \cdot \boldsymbol{s})\right)$ from $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^\pm)$ and $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$.

Algorithm 3 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**
      $T_i^\pm \leftarrow \mathtt{EvalAuto}(S_i^\pm, n+1)$
      $S_i^\pm = S_i^\pm + T_i^\pm$
      $S_{i+n/2}^\pm = X^{-n/2} \cdot (S_i^\pm - T_i^\pm)$
return $\left\{S_i^\pm\right\} = \left\{\mathsf{RLWE}'(N \cdot s_i^\pm)\right\}$

In Algorithm 3, 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 will require larger auxiliary modulus p' to make the error negligible.

4.2 Performing blind rotations on the fly

When the computational party is limited in storage for every blind rotation step i, we can reconstruct only $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$, perform the blind rotation step on the fly, and discard $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$. 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 4 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 4.1.

Algorithm 4 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 4.1, or evaluate the blind rotation step i on the fly using only $\mathsf{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$ and $\mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{\pm})$. For the latter case, we first evaluate

$$\mathsf{RLWE}'_{\boldsymbol{s}}(X^{\alpha_i \cdot s_i}) = \mathsf{RLWE}'^{0}(1) + (X^{\alpha_i} - 1) \cdot \mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{+}) + (X^{-\alpha_i} - 1) \cdot \mathsf{RLWE}'_{\boldsymbol{s}}(s_i^{-}). \tag{2}$$

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 (2) 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 $2(N+1)dBE_{\mathtt{rc}} + \frac{dB}{2}E_{\mathtt{err}}$.

4.3 Blind rotations for sparse ternary secret key

For sparse ternary secret key with hamming weight 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 generate blind rotation public keys

$$\mathtt{brk}' = \left\{\mathsf{RGSW}_{Q,\boldsymbol{s}}\left(\delta^+(i_l, j)\right), \mathsf{RGSW}_{Q,\boldsymbol{s}}\left(\delta^-(i_l, j)\right)\right\},$$

where

$$\begin{cases}\delta^+(i_l,j)=1, & i_l = j \ \&\ s_{i_l}=1\\ \delta^+(i_l,j)=0, & \text{otherwise}\end{cases}, \begin{cases}\delta^-(i_l,j)=1, & i_l = j \ \&\ s_{i_l}=-1\\ \delta^-(i_l,j)=0, & \text{otherwise}\end{cases} \text{ for } \begin{cases} j \in [0, N-1]\\ l \in [1, h].\end{cases}$$

Then we iteratively compute

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

and update $\mathtt{ACC}$ as

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

The equation (3) is correct as only one of summands is non-zero. This alternative blind rotation decreases number of $\mathsf{RLWE} \odot \mathsf{RGSW}$ multiplications from N to h, but increases the blind rotation key size from $2N$ to $2hN$ of RGSW encryptions. We can also apply compact representation of blind rotation keys, however we require $2h$ of RLWE' ciphertexts instead of 2.

Algorithm 5 Blind rotation for sparse secret key

procedure BLINDROTATE'($\mathtt{f}, (\vec{\alpha}, \beta), \mathtt{brk}'$)
  $\mathtt{ACC} \leftarrow \left(0, \mathtt{f} \cdot X^{\beta}\right)$
  for $l = 1, ..., h$ **do**
    $\mathtt{ACC} \leftarrow \mathtt{ACC} \odot \left(\sum_{j=0}^{N-1} X^{\alpha_j} \cdot \mathsf{RGSW}\left(\delta^+(i_l, j)\right) + \sum_{j=0}^{N-1} X^{-\alpha_j} \cdot \mathsf{RGSW}\left(\delta^-(i_l, j)\right)\right)$
return ACC

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

5 Bootstrapping

In this section, we describe 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 explain the bootstrapping technique to BGV and BFV schemes as subsequent results.

5.1 Bootstrapping for CKKS

The main ingredients of the existing CKKS bootstrapping are homomorphic linear transformations and the evaluation of approximating polynomials for modulus reduction functions. Given a ciphertext $\mathtt{ct} = \mathsf{RLWE}(\boldsymbol{m}) = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q$, one can represent it in a higher modulus Q. Then the decryption query becomes

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

for some small polynomial $\boldsymbol{v}$. After a homomorphic linear transformation, one can obtain ciphertext with slots $m_i + e_i + q \cdot v_i$. After using polynomial approximation of the modulus reduction function, one removes the $q \cdot v_i$ part. Using another homomorphic linear transformation, one finally obtains the ciphertext $\mathtt{ct}'$ encrypting $\boldsymbol{m}$ in a higher modulus Q'. A major drawback of this method is that it is hard to approximate a modulus reduction function with a polynomial so that the ciphertext after the bootstrapping usually adds a large noise and it reduces the quality of the encrypted message.

We propose a completely new technique of bootstrapping which only adds a noise, comparable with rescaling noise, and thus almost preserves the quality of the encrypted message. As a first step, the ciphertext is preprocessed to obtain 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.

5.1.1 Original CKKS

We start with original CKKS scheme [11], where ciphertext modulus q is a power of two. Let us have a ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ where

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

Let $q' = q/2N$ and $\|\boldsymbol{m} + \boldsymbol{e}\|_\infty < q'$. Consider $\mathtt{ct}' = \mathtt{ct} \pmod{q'} = \left([\boldsymbol{a}]_{q'}, [\boldsymbol{b}]_{q'}\right) \in \mathcal{R}_{q'}^2$ where

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

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 preprocessed $\mathtt{ct}_{\mathtt{prep}} = \mathsf{RLWE}_{2N,\boldsymbol{s}}^0(-\boldsymbol{u})$, so we can evaluate $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q', Q)$ and obtain $\mathtt{ct}_{\mathtt{sm}} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(-q' \cdot \boldsymbol{u})$ with an error $\boldsymbol{e}_{\mathtt{sm}}$. Finally we add it with $\mathtt{ct}'$ and obtain the ciphertext $\mathtt{ct}_{\mathtt{boot}} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{m})$ with noise $\boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}}$. The full bootstrapping algorithm is described in Algorithm 6.

Sparsely Packed Ciphertext Let $\mathtt{ct}$ be an encryption of a sparsely packed plaintext $\boldsymbol{m}$ of n values. Then the bootstrapping complexity can be reduced. We firstly prepare $\mathtt{ct}'$ and $\mathtt{ct}_{\mathtt{prep}}$ as previously. Then we do additional preprocessing for $\mathtt{ct}'$ and apply a variant of `ScaledMod` to $\mathtt{ct}_{\mathtt{prep}}$.

The additional preprocessing is zeroizing certain coefficients of $\boldsymbol{u}$. We take a similar approach used in the original CKKS bootstrapping [12] which described in Algorithm 7. We increase the modulus of $\mathtt{ct}'$ to Q and then apply automorphisms and additions.

Algorithm 6 Bootstrapping for CKKS

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

$\mathtt{Preprocess}$: $\mathtt{ct}' \leftarrow \mathtt{ct} \pmod{q'}$, $\mathtt{ct}_{\mathtt{prep}} \leftarrow \left(\frac{\mathtt{ct}-\mathtt{ct}'}{q'}\right)$

- $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{a}\boldsymbol{s} + \boldsymbol{b} = \boldsymbol{m} + \boldsymbol{e} + q\boldsymbol{v} \in \mathcal{R}$
- $\mathtt{ct}'(\boldsymbol{s}) = [\boldsymbol{a}]_{q'}\,\boldsymbol{s} + [\boldsymbol{b}]_{q'} = \boldsymbol{m} + \boldsymbol{e} + q'\boldsymbol{u} \in \mathcal{R}$
- $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{prep}}\boldsymbol{s} + \boldsymbol{b}_{\mathtt{prep}} = -\boldsymbol{u} \pmod{2N}$

$\mathtt{ScaledMod}$: $\mathtt{ct}_{\mathtt{sm}} \leftarrow \mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q', Q)$

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

$\mathtt{Combine}$: $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}'$

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

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

Algorithm 7 Zeroizing Coefficients

procedure ZEROIZECOEFFS($\mathtt{ct}', n$)

  for $k = N; k > n; k = k/2$ **do**

    $\mathtt{ct}' \leftarrow \mathtt{EvalAuto}(\mathtt{ct}', k+1) + \mathtt{ct}'$

$\mathtt{ct}' \leftarrow \mathtt{Rescale}(\mathtt{ct}', \frac{N}{2n})$

return $\mathtt{ct}'$

After zeroizing coefficients of $\mathtt{ct}'$, we have the decryption query as

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

where $Q' = Q \cdot \frac{2n}{N}$ and $\boldsymbol{u}'$ has same coefficients as $\boldsymbol{u}$ at degrees that are multiples of $N/2n$, and other coefficients of $\boldsymbol{u}$ are zero. Notice that as $\boldsymbol{m}$ was sparse-packed plaintext, it 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 input of $\mathtt{ScaledMod}$, we can evaluate the blind rotations only for $u_{(N/2n)\cdot i}$ instead of evaluating for every coefficient of $\boldsymbol{u}$. It reduces the number of $\mathsf{LWE}^0_{2N,\vec{s}}(u_i)$ and $\mathsf{RLWE}_{Q,\boldsymbol{s}}(\mathtt{f} \cdot X^{u_i})$ to $2n$ and also reduces the number of iterations of $\mathtt{Repack}$ algorithm. The output of the variant $\mathtt{ScaledMod}$ is $\mathtt{ct}_{\mathtt{sm}} = \mathsf{RLWE}_{Q',\boldsymbol{s}}(-q'\boldsymbol{u}')$ and the final step is $\mathtt{ct}_{\mathtt{boot}} = \mathtt{ct}' + \mathtt{ct}_{\mathtt{sm}}$.

5.1.2 RNS-CKKS

In RNS-CKKS scheme [12], the modulus q is not a power of two but a product of primes. So our preprocessing steps are different. We again start with $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b})$ such that

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

We assume that $\|\boldsymbol{m} + \boldsymbol{e}\|_\infty < q/2N$ and consider $\mathtt{ct}' = 2N \cdot \mathtt{ct} \pmod{q} = ([2N\boldsymbol{a}]_q, [2N\boldsymbol{b}]_q) \in \mathcal{R}_q^2$. We have

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

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

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

Again it is easy to see that for preprocessed $\mathtt{ct}_{\mathtt{prep}} = \mathsf{RLWE}^0_{2N,\boldsymbol{s}}(-\boldsymbol{u})$ we can evaluate $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q, Qp)$ and obtain $\mathtt{ct}_{\mathtt{sm}} = \mathsf{RLWE}_{Qp,\boldsymbol{s}}(-q \cdot \boldsymbol{u})$, where p is an auxiliary prime that we will rescale by later. Now we add $\mathtt{ct}_{\mathtt{sm}}$ and $\mathtt{ct}'$, multiply by $\frac{p}{2N}$, and rescale the result by p: $\mathtt{ct}_{\mathtt{boot}} = \mathtt{Rescale}(\frac{p}{2N} \cdot (\mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}'), p)$.

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

The full bootstrapping algorithm is described in Algorithm 8.

Algorithm 8 Bootstrapping for RNS-CKKS

procedure BOOTSTRAP-RNS-CKKS($\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$)
- `Preprocess`: $\mathtt{ct}' \leftarrow 2N\mathtt{ct} \pmod{q}$, $\mathtt{ct}_{\mathtt{prep}} \leftarrow \frac{1}{q}(2N\mathtt{ct} - \mathtt{ct}')$
  - $\mathtt{ct}'(\boldsymbol{s}) = [2N\boldsymbol{a}]_q\, \boldsymbol{s} + [2N\boldsymbol{b}]_q = 2N\boldsymbol{m} + 2N\boldsymbol{e} + q\boldsymbol{u} \in \mathcal{R}$
  - $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{prep}}\boldsymbol{s} + \boldsymbol{b}_{\mathtt{prep}} = -\boldsymbol{u} + 2N\boldsymbol{v} \in \mathcal{R}$
- `ScaledMod`: $\mathtt{ct}_{\mathtt{sm}} \leftarrow \mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q, Qp)$
  - $\mathtt{ct}_{\mathtt{sm}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{sm}}\boldsymbol{s} + \boldsymbol{b}_{\mathtt{sm}} = -q\boldsymbol{u} + \boldsymbol{e}_{\mathtt{sm}} \pmod{Qp}$
- `Combine`: $\mathtt{ct}_{\mathtt{boot}} \leftarrow \mathtt{Rescale}\left(\frac{p}{2N}(\mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}'), p\right)$
  - $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{boot}}\boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} = \boldsymbol{m} + \boldsymbol{e} + \frac{1}{2N}\boldsymbol{e}_{\mathtt{sm}} + \boldsymbol{e}_{\mathtt{rs}} \pmod{Q}$

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

5.2 Bootstrapping for BGV and BFV

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

5.2.1 BGV

The decryption query of 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 original CKKS with the only difference that keys are generated with errors of the form $t \cdot \boldsymbol{e}$ instead of $\boldsymbol{e}$ and the automorphisms and rescales 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, to obtain encryption of $\boldsymbol{m}$.

5.2.2 BFV

The decryption query of BFV scheme is

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

The goal of bootstrapping for BFV is to reduce the accumulated error instead of increasing the modulus size. During the procedure, the original big error is removed and replaced with a small refreshed error generated from `ScaledMod` and rescaling.

6 Conclusion

We proposed the first bootstrapping procedure for the CKKS scheme without approximating the modulus reduction function, and extended it to BGV and BFV schemes. Our bootstrapping procedure uses a blind rotation technique and it introduces small rescaling errors instead of big approximation errors as in previous CKKS bootstrapping methods.

We also 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. Finally, we modified the blind rotation for a sparse secret key to reduce computational complexity.

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 our technique on different computational platforms including systems with limited memory storage. We also consider continuing our research on possible optimizations of our new bootstrapping algorithms.

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] 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.

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

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

[16] 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.

[17] 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 (to appear) Springer.

[18] 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 (to appear) Springer.

[19] 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**.

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

[21] 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.

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

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

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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

[30] 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.

[31] Micciancio, 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.

[32] 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.

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

[34] 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.

[35] 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.

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

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

[38] 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 5, with the difference that the blind rotation keys $\mathsf{RGSW}_{Q,\boldsymbol{s}}(s_i^{\pm})$ are generated with errors of type $t\boldsymbol{e}$ instead of $\boldsymbol{e}$.
Let us have a ciphertext $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_q^2$ where

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

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 may assume that $\|\boldsymbol{m} + t\boldsymbol{e}\|_\infty < q'$ with high probability. Hence,

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

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

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

After applying $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q', Q)$, we obtain $\mathtt{ct}_{\mathtt{sm}} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(-q' \cdot \boldsymbol{u})$ with an error $t\boldsymbol{e}_{\mathtt{sm}}$. Finally we add it with $\mathtt{ct}'$ and obtain the ciphertext $\mathtt{ct}_{\mathtt{boot}} = \mathsf{RLWE}_{Q,\boldsymbol{s}}(\boldsymbol{m})$ with noise $t\,(\boldsymbol{e} + \boldsymbol{e}_{\mathtt{sm}})$. The full algorithm for bootstrapping in BGV is described in Algorithm 9.

Algorithm 9 Bootstrapping for BGV

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

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

A.2 RNS-BGV

Our bootstrapping in RNS-BGV is also similar to RNS-CKKS bootstrapping in Section 5, with the only difference at `Combine` step, where division is done by $2N$ modulo t. We obtain encryption of $\boldsymbol{m}$ with error $[2N^{-1}]_t(2Nt\boldsymbol{e} + t\boldsymbol{e}_{\mathtt{sm}} + t\boldsymbol{r})$, where

$$\boldsymbol{r} = \frac{1}{t}\left(2N\boldsymbol{m} - [2N\boldsymbol{m}]_t\right).$$

The algorithm is described in Algorithm 10. 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 [33, 38].

Algorithm 10 Bootstrapping for RNS-BGV

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

$\mathtt{Preprocess}$: $\mathtt{ct}' \leftarrow 2N \cdot \mathtt{ct} \pmod q$, $\mathtt{ct}_{\mathtt{prep}} \leftarrow \frac{1}{q}(2N\mathtt{ct} - \mathtt{ct}')$

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

$\mathtt{ScaledMod}$: $\mathtt{ct}_{\mathtt{sm}} \leftarrow \mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, q, Q)$

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

$\mathtt{Combine}$: $\mathtt{ct}_{\mathtt{boot}} \leftarrow (\mathtt{ct}_{\mathtt{sm}} + \mathtt{ct}') \cdot ([2N^{-1}]_t)$

- $\mathtt{ct}_{\mathtt{boot}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{boot}}\boldsymbol{s} + \boldsymbol{b}_{\mathtt{boot}} = \boldsymbol{m} + [2N^{-1}]_t(2Nt\boldsymbol{e} + t\boldsymbol{e}_{\mathtt{sm}} + t\boldsymbol{r}) = \boldsymbol{m} + t\boldsymbol{e}' \pmod Q$

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

B Bootstrapping for BFV

B.1 Multiprecision BFV

For multiprecision BFV scheme with a power of two Q, we start with $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b})$ 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 $\mathtt{ct}' = t \cdot \mathtt{ct} \pmod Q$, so we have

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

Let $Q' = Q/2N$, and $\mathtt{ct}'' = \mathtt{ct}' \pmod{Q'}$, then we have

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

Now we obtain $\mathtt{ct}_{\mathtt{prep}} = \frac{\mathtt{ct}' - \mathtt{ct}''}{Q'}$ with

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

After applying $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q', Qt)$, we have $\mathtt{ct}_{\mathtt{sm}}$ with an error $\boldsymbol{e}_{\mathtt{sm}}$. Finally we add $\mathtt{ct}_{\mathtt{sm}}$ with $t \cdot \mathtt{ct} - \mathtt{ct}''$ and rescale the result by t, then we obtain $\mathtt{ct}_{\mathtt{boot}}$. The algorithm is described in Algorithm 11.

B.2 RNS-BFV

In RNS-BFV, we again start with $\mathtt{ct} = (\boldsymbol{a}, \boldsymbol{b})$ such that

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

Consider $\mathtt{ct}' = t \cdot \mathtt{ct} \pmod Q$ and $\mathtt{ct}'' = 2N \cdot \mathtt{ct}' \pmod Q$ as

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

and

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

Algorithm 11 Bootstrapping for BFV

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

$\mathtt{Preprocess}$: $\mathtt{ct}' \leftarrow t \cdot \mathtt{ct} \pmod{Q}$, $\mathtt{ct}'' \leftarrow \mathtt{ct}' \pmod{Q'}$, $\mathtt{ct}_{\mathtt{prep}} \leftarrow \frac{\mathtt{ct}' - \mathtt{ct}''}{Q'}$

- $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{a}\boldsymbol{s} + \boldsymbol{b} = \boldsymbol{e} + \frac{Q}{t}\boldsymbol{m} \pmod{Q}$
- $\mathtt{ct}'(\boldsymbol{s}) = [t\boldsymbol{a}]_Q \boldsymbol{s} + [t\boldsymbol{b}]_Q = t\boldsymbol{e} + Q\boldsymbol{v} \in \mathcal{R}$
- $\mathtt{ct}''(\boldsymbol{s}) = [t\boldsymbol{a}]_{Q'} \boldsymbol{s} + [t\boldsymbol{b}]_{Q'} = t\boldsymbol{e} + Q'\boldsymbol{u} \in \mathcal{R}$
- $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{prep}}\boldsymbol{s} + \boldsymbol{b}_{\mathtt{prep}} = -\boldsymbol{u} + 2N\boldsymbol{v} \in \mathcal{R}$

$\mathtt{ScaledMod}$: $\mathtt{ct}_{\mathtt{sm}} \leftarrow \mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q', Qt)$

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

$\mathtt{Combine}$: $\mathtt{ct}_{\mathtt{boot}} = \mathtt{Rescale}\left((\mathtt{ct}_{\mathtt{sm}} + t \cdot \mathtt{ct} - \mathtt{ct}''), t\right)$

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

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

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

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

After applying $\mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q, Qpt)$ with auxiliary prime p, we have $\mathtt{ct}_{\mathtt{sm}}$ with an error $\boldsymbol{e}_{\mathtt{sm}}$. Finally we add $\mathtt{ct}_{\mathtt{sm}}$ and $2Nt \cdot \mathtt{ct} - \mathtt{ct}''$, multiply it by $\frac{p}{2N}$ and rescale the result by pt, then we obtain $\mathtt{ct}_{\mathtt{boot}}$. The algorithm is described in Algorithm 12.

Algorithm 12 Bootstrapping for RNS-BFV

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

$\mathtt{Preprocess}$: $\mathtt{ct}' \leftarrow t \cdot \mathtt{ct} \pmod{Q}$, $\mathtt{ct}'' \leftarrow 2N \cdot \mathtt{ct}' \pmod{Q}$, $\mathtt{ct}_{\mathtt{prep}} \leftarrow \frac{2N \cdot \mathtt{ct}' - \mathtt{ct}''}{Q}$

- $\mathtt{ct}(\boldsymbol{s}) = \boldsymbol{a}\boldsymbol{s} + \boldsymbol{b} = \boldsymbol{e} + \frac{Q}{t}\boldsymbol{m} \pmod{Q}$
- $\mathtt{ct}'(\boldsymbol{s}) = [t\boldsymbol{a}]_Q \boldsymbol{s} + [t\boldsymbol{b}]_Q = t\boldsymbol{e} + Q\boldsymbol{v} \in \mathcal{R}$
- $\mathtt{ct}''(\boldsymbol{s}) = [2Nt\boldsymbol{a}]_Q \boldsymbol{s} + [2Nt\boldsymbol{b}]_Q = 2Nt\boldsymbol{e} + Q\boldsymbol{u} \in \mathcal{R}$
- $\mathtt{ct}_{\mathtt{prep}}(\boldsymbol{s}) = \boldsymbol{a}_{\mathtt{prep}}\boldsymbol{s} + \boldsymbol{b}_{\mathtt{prep}} = -\boldsymbol{u} + 2N\boldsymbol{v} \in \mathcal{R}$

$\mathtt{ScaledMod}$: $\mathtt{ct}_{\mathtt{sm}} = \mathtt{ScaledMod}(\mathtt{ct}_{\mathtt{prep}}, -Q, Qpt)$

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

$\mathtt{Combine}$: $\mathtt{ct}_{\mathtt{boot}} = \mathtt{Rescale}\left(\frac{p}{2N}\left(\mathtt{ct}_{\mathtt{sm}} + 2Nt \cdot \mathtt{ct} - \mathtt{ct}''\right), pt\right)$

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

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