On the Hardness of Scheme-Switching Between SIMD FHE Schemes

Karim Eldefrawy
SRI International
karim.eldefrawy@sri.com

Nicholas Genise*
Duality Technologies
ngenise@dualitytech.com

Nathan Manohar*
IBM T.J. Watson Research Center
nmanohar@ibm.com

Abstract

Fully homomorphic encryption (FHE) schemes are either lightweight and can evaluate boolean circuits or are relatively heavy and can evaluate arithmetic circuits on encrypted vectors, i.e., they perform single instruction multiple data operations (SIMD). SIMD FHE schemes can either perform exact modular arithmetic in the case of the Brakerski-Gentry-Vaikuntanathan (BGV) and Brakerski-Fan-Vercauteren (BFV) schemes or approximate arithmetic in the case of the Cheon-Kim-Kim-Song (CKKS) scheme. While one can homomorphically switch between BGV/BFV and CKKS using the computationally expensive bootstrapping procedure, it is unknown how to switch between these schemes without bootstrapping. Finding more efficient methods than bootstrapping of converting between these schemes was stated as an open problem by Halevi and Shoup, Eurocrypt 2015 [HS15, HS14b].

In this work, we provide strong evidence that homomorphic switching between BGV/BFV and CKKS is *as hard as* bootstrapping. In more detail, if one could efficiently switch between these SIMD schemes, then one could bootstrap these SIMD FHE schemes using a single call to a homomorphic scheme-switching algorithm *without applying homomorphic linear transformations*. Thus, one cannot hope to obtain significant improvements to homomorphic scheme-switching without also significantly improving the state-of-the-art for bootstrapping.

We also explore the relative hardness of computing homomorphic comparison in these same SIMD FHE schemes as a secondary contribution. We show that given a comparison algorithm, one can bootstrap these schemes using a few calls to the comparison algorithm for typical parameter settings. While we focus on the comparison function in this work, the overall approach to demonstrate relative hardness of computing specific functions homomorphically extends beyond comparison to other useful functions such as min/max or ReLU.

1 Introduction

Fully homomorphic encryption (FHE) enables a client to encrypt their data, send it to a cloud server, and have the server compute any function on the client's encrypted data without knowledge of the decryption key. This client-server setting for outsourcing computation is especially useful when the client has limited computational capabilities, and the server has significantly more resources. FHE was first proposed in 1978 by Rivest et al. [RAD$^+$78], but it was not until Gentry's breakthrough in 2009 [Gen09b, Gen09a]

*This work was partially done while these authors were at SRI International.

that the cryptographic community saw a plausible, (truly) fully homomorphic[1] encryption scheme. Gentry's breakthrough in 2009 focused on schemes with limited homomorphic capabilities that also possessed a simple decryption circuit. Such schemes can homomorphically evaluate their decryption circuits to re-encrypt, or "bootstrap," their ciphertexts. Gentry showed that a fully homomorphic encryption scheme can be obtained from any scheme that can homomorphically evaluate a NAND gate together with its decryption circuit [Gen09b]. Further, Gentry based his scheme on ideal lattices since decryption in lattice-based cryptosystems boils down to decoding noisy lattice points back to the public lattice (linear operations followed by a rounding step).

FHE has gone through many iterations towards practicality since Gentry's breakthrough in 2009. Today, there are three main FHE cryptoschemes using (ring) lattices: FHEW/TFHE [DM15, CGGI16] which encrypt one bit, or a scalar, at a time and the BGV/BFV or CKKS schemes which pack (up to) thousands of plaintext values into each ciphertext. The former, FHEW/TFHE, schemes are relatively lightweight but can only perform one nonlinear boolean or small-number operations for each lightweight bootstrapping. On the other hand, the packed schemes, BGV/BFV and CKKS, can perform thousands of arithmetic operations in parallel using ciphertext packing and also support computations between ciphertext slots using the underlying field's Galois automorphisms. After a fixed number of multiplications (referred to as the "depth"), these schemes need to perform a more expensive bootstrapping operation to increase the size of the ciphertext modulus (and lower the relative noise level in the ciphertext in the case of BGV/BFV).

In this work, we focus on the SIMD FHE schemes. BGV and BFV are schemes which perform exact modular arithmetic modulo some prime power whereas CKKS performs approximate arithmetic on fixed-point numbers. It is well-known that homomorphic scheme-switching between BGV and BFV can be computed efficiently by scalar multiplications [AP13]. However, it is not known how to efficiently switch between CKKS and BGV/BFV without bootstrapping.

Given the differing plaintext spaces, switching between schemes can be useful in a variety of applications. For example, say the majority of a computation is done in CKKS until the underlying plaintext has lost a set number of precision bits. Then, one could homomorphically convert the CKKS ciphertexts into BGV/BFV ciphertexts to continue the rest of the computation and avoid any further loss in precision. Alternatively, a computation may consist of several stages where some stages are required to be computed exactly and others can be computed approximately. One could compute homomorphically using CKKS for the approximate stages of the computation and use BGV/BFV for the exact stages of the computation. This is a natural setting since CKKS is much more efficient than BGV/BFV for performing real and complex arithmetic computations. Another setting is to develop a general cross-scheme bootstrapping technique where efficient homomorphic scheme-switching is the first step [HS15]. Then, for example, an ASIC on a server for one scheme could be used for multiple FHE schemes.

Recently, there has been a flurry of work in scheme-specific hardware acceleration for SIMD schemes [FSK$^+$21, SFK$^+$21, SFK$^+$22, AdCY$^+$23, GBP$^+$22]. This gives a clear motivation for efficient homomorphic scheme-switching since a server might have a specialized ASIC for one scheme (CKKS) but has many ciphertext stored encrypted in another scheme (BFV). Ideally, the server could homomorphically switch the ciphertexts into the main scheme used in the ASIC without bootstrapping.

It is folklore that one can switch between schemes using bootstrapping. This is done, essentially, by homomorphically evaluating the decryption circuit of one scheme inside the other. Unfortunately, bootstrapping is computationally expensive, and, thus, it is desirable to obtain more efficient methods of scheme-switching. Finding scheme-switching between BGV/BFV and CKKS methods without bootstrapping was explicitly stated as an open problem by Halevi and Shoup Eurocrypt 2015 [HS15, HS14b]. The recent library OpenFHE [BBB$^+$22] and the recent work [DMPS22] also both explicitly mention that scheme-switching between BGV/BFV and CKKS as an open problem.

[1]Many schemes beforehand had limited homomorphic capabilities. They either had ciphertexts grow exponentially with operations [FK94, AS08, MGH10] or they had limited homomorphism [BGN05, IP07]. See Chapter 3 of Gentry's dissertation [Gen09a] for a survey of previous schemes.

1.1 Our Results

We provide strong evidence that homomorphic scheme-switching between the aforementioned SIMD FHE schemes is *as hard as* bootstrapping. In fact, we show that a weak variant of homomorphic scheme-switching between these schemes which ignores the differences in the plaintext encoding can bootstrap these SIMD schemes in one call to such an algorithm *without applying homomorphic linear transformations.* We model this homomorphic scheme-switcher as an oracle.

In more detail, both BGV/BFV and CKKS have ciphertexts of the form $\mathsf{ct} = (c_0, c_1) \in R_Q^2$, where $R := \mathbb{Z}[X]/(X^N + 1)$ is a cyclotomic ring of integers, $Q \in \mathbb{Z}^+$ is a ciphertext modulus, and $R_Q := R/(QR)$. Since converting between BGV and BFV can be done by simply multiplying by a constant [AP13], we will focus on BGV in this work, but the results hold for BFV as well. In both BGV and CKKS, (homomorphic) decryption begins by computing $(c_0 + c_1 \cdot s(X)) \bmod Q$ for a secret $s(X) \in R$. In BGV, this results in $p \cdot e(X) + m(X) \in R$, where p is the plaintext modulus. In CKKS, this gives $\Delta m(X) + e(X) \in R$ for some scaling factor Δ. A weak scheme-switcher from BGV to CKKS takes as input a BGV ciphertext $(c_0, c_1) \in R_Q^2$ where $(c_0 + c_1 \cdot s(X)) \bmod Q = pe(X) + m(X)$ and outputs a CKKS ciphertext $(c_0', c_1') \in R_{Q'}^2$ where $(c_0' + c_1' \cdot s(X)) \bmod Q' = \Delta m(X) + e'(X)$. (A weak homomorphic scheme-switcher from CKKS to BGV is defined analogously.) We refer to such a scheme-switcher as *weak* because it homomorphically switches schemes with respect to the same ring polynomial $m(X)$ without dealing with the fact that BGV and CKKS have different plaintext encodings. Recall, in BGV, the plaintext polynomial $m(X) \in R_p$ is related to $(\mathbb{Z}_p)^k$ or $(\mathbb{F}_{p^r})^k$ via a ring isomorphism [GHS12b], whereas in CKKS, the plaintext polynomial $\Delta m(X) + e(X)$ is embedded into $\mathbb{C}^{N/2}$ via the canonical embedding [CKKS17]. Thus, in order to homomorphically switch between these schemes in practice, one would also need to deal with the differences in plaintext encodings. This is done by applying homomorphic linear transformations where the matrix is represented by plaintext polynomials [HS14a, CHK$^+$18]. However, we show this weaker variant of homomorphic scheme-switching already implies an immediate bootstrapping algorithm *without applying homomorphic linear transformations.* We model the weak scheme-switchers as oracles $\mathcal{O}_{C \hookrightarrow B}$ and $\mathcal{O}_{B \hookrightarrow C}$. Informally, we show the following main results.

Theorem 1 (Informal)**.** *If there exists a weak homomorphic scheme-switching algorithm from BGV to CKKS, $\mathcal{O}_{B \hookrightarrow C}$, then there exists a CKKS bootstrapping algorithm with the same time complexity plus the time to perform one CKKS rescaling operation.*

Theorem 2 (Informal)**.** *If there exists a weak homomorphic scheme-switching algorithm from CKKS to BGV, $\mathcal{O}_{C \hookrightarrow B}$, then there exists a BGV bootstrapping algorithm with the same time complexity plus the time to perform one BGV plaintext multiplication, one plaintext addition, and one modulus switching operation.*

As a secondary contribution, we also explore the relative hardness of computing homomorphic comparison in these SIMD FHE schemes. Prior works [CKK$^+$19, CKK20, IZ21, LLKN21] focused on computing homomorphic comparison (and other related functions such as max/min and ReLU) in these SIMD FHE schemes since these functions are useful to compute for many machine-learning applications. Computing comparisons in these SIMD schemes is difficult since comparison is not easily expressible as a shallow arithmetic circuit. We model the homomorphic comparison functionality as an oracle $\mathcal{O}_{\geq}$ and show how to use several calls to this oracle to bootstrap these SIMD FHE schemes. We focus on comparison in this work, but our approach extends to other related functions such as max/min and ReLU. Unlike our primary contribution on the hardness of homomorphic scheme-switching, this secondary result is somewhat expected because bootstrapping requires some form of digit extraction.

1.2 Technical Overview

A simplified view of a weak scheme-switching oracle, $\mathcal{O}_{B \hookrightarrow C}$, is shown in Figure 1. The main intuition is that any non-trivial bit-wise manipulations on an RLWE encryption's message/error polynomial's coefficients are the most expensive operations performed in bootstrapping. This is seen in BGV's decryption circuit: there is a linear operation on the ciphertext over R_Q, $d := c_0 + c_1 \cdot s \bmod Q$, followed by some nonlinear rounding, $d \bmod p$. In essence, we show the homomorphic nonlinear rounding in either BGV or CKKS can be achieved by (homomorphically) moving around contiguous bits of the message/error polynomial's coefficients, weakly switching the plaintext encoding of the encrypted plaintext. Hence, a weak converter is quite powerful in the FHE setting. We sketch the case from BGV to CKKS using $\mathcal{O}_{B \hookrightarrow C}$

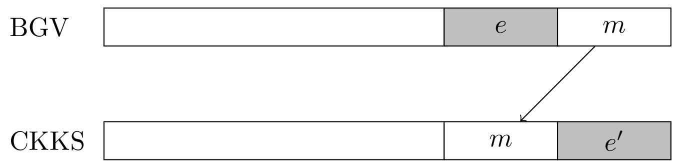


Figure 1: Visualization of a weak scheme-switching oracle from BGV to CKKS. This process is called *weak* because homomorphic scheme-switching using this oracle requires us to call BGV's slots-to-coefficient algorithm, a homomorphic linear transformation, before we call the oracle and CKKS's coefficient-to-slots algorithm, another homomorphic linear transformation, after we call the oracle. We show this "weak" process is surprisingly powerful: it allows for bootstrapping CKKS ciphertexts *without* homomorphic linear transformations.

since it is simpler. Recall the main idea of bootstrapping a CKKS ciphertext [CHK+18]: we have a CKKS ciphertext $\mathsf{ct} = (c_0, c_1) \in R_q^2$ which we implicitly treat as a ciphertext with respect to a larger ciphertext modulus $Q > q$ with $q \mid Q$. That is, we view ct as a pair of elements in R_Q^2. We need to compute the $y(X) \mapsto y(X) \bmod q$ function (where the mod is taken on all the coefficients of $y(X)$) homomorphically on ct to obtain a $\mathsf{ct}' \in R_Q^2$ that decrypts to approximately the same value as ct under the same secret key for some ciphertext modulus $Q' > q$. Observe that $c_0 + c_1 \cdot s(X) = \Delta m(X) + e(X) + I(X)q$ where each coefficient of $\Delta m(X) + e(X)$ is $\ll q$. Notice that bootstrapping in CKKS reduces to clearing the high order bits of the plaintext polynomial's coefficients (the $I(X)q$ term). Standard CKKS bootstrapping moves these coefficients to the plaintext slots via discrete Fourier transform (DFT), computes the mod q function via a polynomial approximation, and then moves the result back to the coefficients.

However, there is a much more efficient algorithm for performing the $y \mapsto y \bmod q$ function on a CKKS ciphertext's coefficients given the oracle in Figure 1. If we simply treat the input CKKS ciphertext $\mathsf{ct} \in R_Q^2$ as a BGV ciphertext with plaintext modulus q, we observe that ct is a valid BGV encryption of $\Delta m(X) + e(X)$ with error $I(X)q$ for some small integer polynomial $I(X)$. The oracle $\mathcal{O}_{B \hookrightarrow C}$ applied to this BGV encryption outputs a CKKS encryption of $\Delta'(\Delta m(X) + e(X)) + e'(X)$ with ciphertext modulus $Q' \gg q$ and some other error $e'(X)$, noting that $q > \Delta m(X)$. Therefore, we perform one CKKS rescaling operation and return the result as the new bootstrapped ciphertext of CKKS encrypting $m(X)$. The CKKS to BGV case is similar.

1.3 Related Works

Switching between BGV/BFV and CKKS without bootstrapping was first mentioned as an open problem in the updated version of [HS15, HS14b] as it pertains to a potential cross-scheme bootstrapping technique.

The problem of switching between FHEW/TFHE and BGV/BFV and CKKS is an orthogonal problem since once you apply homomorphic linear transformations on the latter, the rest is up to extracting the coefficients as separate lightweight FHEW/TFHE ciphertexts. This was done explicitly in [BGGJ20, LHH+21], respectively called Chimera and Pegasus, and was partially done in the original FHEW/TFHE works [DM15, CGGI16] while interfacing the homomorphic accumulator and LWE Regev encryptions [Reg05, Reg09]. Chimera [BGGJ20] and Pegasus [LHH+21] consider methods of scheme-switching between CKKS and TFHE via bootstrapping. Moreover, Chimera [BGGJ20] also considers scheme-switching from BFV to TFHE by bootstrapping. Chimera handles the differing plaintext spaces by using the underlying ring's coefficient packing, similar to our approach. Kim et al. [KDE+21] take this idea further and apply FHEW/TFHE techniques to packed schemes, BGV/BFV and CKKS. Liu, Micciancio, and Polyakov [LMP22] developed a high precision FHEW functional bootstrapping with the CKKS-to-FHEW/TFHE application in mind, building on Chillotti et al. [CLOT21].

Additionally, several works [CKK+19, CKK20, IZ21, LLKN21] have focused on methods of computing comparison (and other related functions such as max/min and ReLU) in these SIMD FHE schemes since these functions are useful to compute for applications, but are not easily expressible as arithmetic circuits.

We note the security model likely changes when going from BGV/BFV to CKKS, since the latter is an approximate FHE scheme [LM21], if the user publishes decryption results. One must apply noise flooding when switching to CKKS in applications where the noisy CKKS decryption value is publicly available or published [LMSS22].

1.4 Organization

In Section 2, we cover the necessary background and preliminaries. Scheme-switching oracles are defined in Section 3. We also show how to transform a weak scheme-switching oracle, which works on coefficients, into a strong scheme-switching oracle, which inputs a packed BGV (resp. CKKS) ciphertext and outputs a packed CKKS (resp. BGV) ciphertext, in this section. In Section 4, we give the main results of our paper in Theorems 5 and 6. In Section 5, we show how one can bootstrap BGV and CKKS with a few calls to comparison oracles. We conclude in Section 6.

2 Preliminaries

We denote the integers as $\mathbb{Z}$, the rationals as $\mathbb{Q}$, the reals as $\mathbb{R}$, and the complex numbers as $\mathbb{C}$. For polynomials or vectors of real numbers, we use the notation $\lceil v \rfloor$ to denote coefficient-wise rounding to the integers. For an integer z, we denote its balanced remainder modulo q as $[z]_q \in [-q/2, q/2)$ and its p-digit decomposition as $z = \sum_i z\langle i \rangle p^i$ where $z\langle i \rangle \in [-p/2, p/2)$. We denote an integer z's k-through-j middle digits as $z\langle j, \dots, k \rangle = \sum_{i=k}^{j} z\langle i \rangle p^i$ for $k < j$. Throughout the paper, we use power of two cyclotomic rings[2] (fields), $R := \mathbb{Z}[X]/(X^N + 1)$ ($F := \mathbb{Q}[X]/(X^N + 1)$) is the cyclotomic ring (field) of order $2N$, where N is a power of two. We often view these rings in their coefficient embedding. That is, $a(X) = \sum_{i=0}^{N-1} a_i X^i \in R$ embeds as $a(X) \leftrightarrow (a_0, \dots, a_{N-1}) \in \mathbb{Z}^N$ and the analogous embedding for the cyclotomic field. We denote the ℓ_∞ norm of a polynomial in $a(X) = \sum_{i=0}^{N-1} a_i X^i \in R$ as $\|a\|_\infty = \max_i |a_i|$. Any two polynomials $f, g \in R$ satisfy $\|fg\|_\infty \leq N\|f\|_\infty\|g\|_\infty$. This worst-case bound is rarely met when the polynomials are distributed as in FHE ciphertexts and $\|fg\|_\infty \leq 2\sqrt{N}\|f\|_\infty\|g\|_\infty$ is often seen in practice with high probability [HPS19].

2.1 RLWE SIMD Schemes

Here we discuss the main SIMD schemes used in practice: CKKS [CKKS17], BGV [BGV12], and BFV [Bra12, FV12] . These schemes are RLWE-based crypto-systems [LPR10] and use ciphertext packing [GHS12b, SV14], which allows a ciphertext to hold up to thousands of plaintext scalars for common parameters. Even though BGV and BFV can encrypt elements from an extension field of $\mathbb{Z}_p$, we focus on the case where they encrypt scalar elements for simplicity. BGV/BFV and CKKS differ greatly in their plaintext spaces even with this simplification. The former have each plaintext scalar in $\mathbb{Z}_p$, and the latter has each in $\mathbb{R}$ (or even $\mathbb{C}$) via fixed-point approximations. Since we can always switch between BFV and BGV with the same plaintext spaces cheaply with scalar multiplications [AP13, KPZ21], we will often focus on BGV in this work, but our results also apply for BFV.

Let $R := \mathbb{Z}[X]/(X^N + 1)$ be the $2N$-th cyclotomic ring for N a power of two and let $R_Q := R/QR$ be the ciphertext space where Q is a positive integer. When sampling ring elements, we refer to them by their coefficient representations in the appropriate space (i.e. $\mathbb{Z}^N, \mathbb{Z}_Q^N, \mathbb{Z}_p^N$). We will work with residue classes in the balanced representation (so elements of $\mathbb{Z}_Q$ are represented by $[-Q/2, Q/2) \cap \mathbb{Z}$).

BGV. A BGV encryption is defined as follows.

Definition 2.1. Fix a ciphertext modulus Q, a plaintext modulus p (typically a prime power), and a ring dimension N. Further, p and Q must be coprime integers. A BGV encryption of $m \in R_p$ for a secret $s \in R$ with small norm, and a noise distribution χ, is a pair of ring elements $ct = (c_0, c_1) \in R_Q^2$ where $c_0 = as + pe + m$, $c_1 = -a$, and $e \leftarrow \chi$. To decrypt, one computes $(c_0 + c_1 s \bmod Q) \bmod p$. For correctness of decryption, we require that $\|pe + m\|_\infty < Q/2$. We denote the set of BGV ciphertexts that decrypt to m, under some fixed s and χ, as $\mathsf{BGV}^s(m)_{p,Q} \subset R_Q^2$.

[2] All of our results apply to arbitrary cyclotomic fields. See [HS20] for the full details of BGV in general cyclotomics.

Note that we leave χ unspecified since the error distribution is affected by homomorphic operations. The appropriate χ for a particular ciphertext will be clear from context.

BFV. A BFV encryption is defined as follows.

Definition 2.2. Fix a ciphertext modulus Q, a plaintext modulus p (typically a prime power), and a ring dimension N. A BFV encryption of $m \in R_p$ for a secret $s \in R$ with small norm, and a noise distribution χ, is a pair of ring elements $ct = (c_0, c_1) \in R_Q^2$ where $c_0 = as + \lceil \Delta m \rfloor + e$, for $\Delta = \frac{Q}{p} \in \mathbb{Q}$, $c_1 = -a$, and $e \leftarrow \chi$. To decrypt, one computes

$$m' = \lceil (c_0 + c_1 s \bmod Q)/\Delta \rfloor .$$

For correctness of decryption, we require that $\|e\|_\infty < Q/(2p) - 1/2 = (\Delta - 1)/2$.

See [KPZ21] for more details on BFV, its optimized version, and its relation to BGV.

A key noise management operation in BGV is modulus-switching, defined as follows. Note, we focus on the case where the moduli, Q and Q', satisfy $Q = DQ'$, as all feasible BGV implementations include a chain of ciphertext moduli satisfying this divisibility requirement, e.g., HELib [HS20].

Definition 2.3 ([BGV12, GHS12b])**.** Let $\mathsf{ct} \in \mathsf{BGV}^s(m)_{p,Q}$ be a BGV ciphertext and $Q = Q'D$ be a positive integer coprime with p, and $Q \bmod p = Q' \bmod p = 1$. Then, the BGV modulus-switching operation is

$$\mathsf{ct}' \leftarrow \lceil (Q'/Q) \cdot (\mathsf{ct} + \delta) \rfloor_p \in R_{Q'}^2,$$

where $\delta = p \cdot ([-c_0/p]_D, [-c_1/p]_D) \in R^2$ and $\lceil \frac{Q'}{Q} z \rfloor_p$ maps an integer $z \in [-Q/2, Q/2)$ to the nearest integer, z', in $[-Q'/2, Q'/2)$ such that $z' \equiv z \mod p$. We write $\mathsf{ct}' \leftarrow \mathsf{BGV.ModSwitch}(\mathsf{ct})_{Q \to Q'} \in \mathsf{BGV}^s(m)_{p,Q'}$ as shorthand for BGV modulus-switching.

Modulus-switching is the main noise-management technique used in BGV besides bootstrapping. The following lemma states its effect on ciphertext noise precisely.

Lemma 2.1 ([BGV12, GHS12b])**.** *If* $\mathsf{ct} \in \mathsf{BGV}^s(m)_{p,Q}$ *is a BGV ciphertext satisfying*

$$\|c_0 + c_1 s \bmod Q\|_\infty \le \frac{Q}{2} - \frac{pD(1 + N\|s\|_\infty)}{2},$$

then $\mathsf{ct}' \leftarrow \mathsf{BGV.ModSwitch}(\mathsf{ct})_{Q \to Q'} \in \mathsf{BGV}^s(m)_{p,Q'}$ *is a BGV ciphertext with error norm* $\|e'\|_\infty$ *at most* $\|e\|_\infty/D + (1 + N\|s\|_\infty)/2$*, where* $Q'D = Q$ *and* $e \in R$ *is the error for the input ciphertext* ct.

CKKS scheme. A CKKS encryption with respect to a scaling parameter $\Delta \in \mathbb{Z}$ is defined as follows.

Definition 2.4. Fix a ciphertext modulus Q, $\Delta \in \mathbb{Z}^+$, and a ring dimension N. A CKKS encryption of $m \in R$ with scaling factor $\Delta \in \mathbb{Z}^+$ is a pair of ring elements $\mathsf{ct} = (c_0, c_1) \in R_Q^2$ where $c_0 = as + e + \Delta m$, $c_1 = -a$. To decrypt, one computes $\lceil (c_0 + c_1 s \bmod Q)/\Delta \rfloor$. For correctness of decryption, we require that $\|\Delta m + e\|_\infty < Q/2$. We denote the set of CKKS ciphertexts, under some fixed s and χ, as $\mathsf{CKKS}^s(m)_{\Delta,Q} \subset R_Q^2$.

The flexibility in choosing Δ, together with ciphertext packing, allows CKKS encryption to encrypt fixed-point approximations of numbers. Next, we describe the analogous noise-management technique in CKKS to BGV's modulus switching.

Definition 2.5 ([CKKS17])**.** Given a CKKS ciphertext $ct \in \mathsf{CKKS}^s(m)_{\Delta,Q}$ with a ciphertext modulus $Q = Q'D$, CKKS rescaling is the following operation:

$$\mathsf{ct}' \leftarrow \lceil (Q'/Q) \cdot \mathsf{ct} \rfloor \in R_{Q'}^2,$$

where multiplication is done over $\mathbb{Q}$. We use $\mathsf{CKKS.Rescale}(\mathsf{ct})_{Q \to Q'}$ as shorthand for the above operation.

Rescaling and modulus switching are the nearly same operation, but the rounding factor in BGV has entries as large as $\pm p/2$ wheres the rounding factor in CKKS has smaller entries in $[\pm 1/2]$. Next, we give the change in ciphertext noise under the infinity metric of the coefficients.

Lemma 2.2 ([CKKS17])**.** *Let* $\mathsf{ct} \in \mathsf{CKKS}^s(m)_{\Delta,Q}$ *be a CKKS ciphertext satisfying*

$$\|c_0 + c_1 s \bmod Q\|_\infty \le Q/2 - D(N\|s\|_\infty + 1)/2$$

and let $Q = Q'D$*. Then, the operation,* $\mathsf{ct}' \leftarrow \mathsf{CKKS.Rescale}(\mathsf{ct})_{Q \to Q'} \in R_{Q'}^2$*, is a CKKS encryption,* $\mathsf{ct}' \in \mathsf{CKKS}^s(m)_{\Delta/D,Q'}$*. Furthermore, if the error term* e *in* $\mathsf{ct} = (as + \Delta m + e, -a)$ *has norm* $\|e\|_\infty$*, then* ct' *has an error norm* $\|e'\|_\infty$ *with norm at most* $\|e\|_\infty/D + (N\|s\|_\infty + 1)/2$.

It is easy to see that if ct satisfies

$$\|c_0 + c_1 s \bmod Q\|_\infty \le Q/2 - D(N\|s\|_\infty + 1)/2,$$

then you can always decrypt after rescaling. The same holds true for BGV modulus-switching if the input ciphertext satisfies

$$\|c_0 + c_1 s \bmod Q\|_\infty \le Q/2 - pD(N\|s\|_\infty + 1)/2.$$

Homomorphic operations. Adding and multiplying ciphertexts in BGV and CKKS are given by the following simple operations: if $\mathsf{ct} = (c_0, c_1) \in \mathsf{BGV}^s(m)_{p,Q}$ and $\mathsf{ct}' = (c_0', c_1') \in \mathsf{BGV}^s(m')_{p,Q}$, then addition over R_Q^2 $(\mathsf{ct} + \mathsf{ct}')$ gives a ciphertext in $\mathsf{BGV}^s(m + m')_{p,Q}$. We can multiply a BGV (CKKS) ciphertext by a plaintext polynomial $\alpha \in R_p$ $(\alpha \in R)$ by simply returning $\mathsf{ct} \leftarrow \alpha \cdot \mathsf{ct} \in R_Q^2$. It is easy to see how this increases the noise from $\|e\|_\infty$ to $\|\alpha\|_\infty \|e\|_\infty$. Multiplying BGV or CKKS ciphertexts is given by $(d_0, d_1, d_2) = (c_0 c_0', c_0 c_1' + c_0' c_1, c_1 c_1') \in R_Q^3$. In BGV, we get the following:

$$\begin{aligned}(c_0 + c_1 s)(c_0' + c_1' s) &= c_0 c_0' + s(c_0 c_1' + c_0' c_1) + s^2 c_1 c_1' \\ &= p(pee' + em' + e'm) + mm' \\ &= mm' \bmod p.\end{aligned}$$

We have the following analogous equations for multiplying two CKKS ciphertexts:

$$\begin{aligned}(c_0 + c_1 s)(c_0' + c_1' s) &= c_0 c_0' + s(c_0 c_1' + c_0' c_1) + s^2 c_1 c_1' \\ &= \Delta^2 mm' + (\Delta m e' + \Delta m' e + ee').\end{aligned}$$

Therefore, we "re-linearize" the ciphertext after multiplication with a key-switching operation to get a ciphertext represented with two polynomials again in R_Q encrypting mm'. For simplicity, we use the Gentry-Halevi-Smart method [GHS12c] for relinearization: The *relinearization key* is an RLWE encryption of s^2 under the original key s under a larger modulus, $Q' = PQ$ where Q is the largest modulus allowed for ciphertexts. That is, the evaluation key is $evk := (-as + Ps^2 + pe, a) \in R_{P \cdot Q}^2$. Then, we relinearize (d_0, d_1, d_2) by returning

$$(d_0, d_1) + \mathsf{BGV.ModSwitch}(d_2 \cdot evk)_{PQ \to Q}.$$

For CKKS, we have the same operations but the evaluation key does not have its RLWE error scaled by p: $evk := (-as + Ps^2 + e, a) \in R_{PQ}^2$.

See [BGV12, GHS12b, CKKS17] for the full details of the BGV and CKKS schemes.

2.2 Useful Lemmas

Here we list some useful lemmas used throughout the paper. First, we list a core lemma to the state of the art in BGV bootstrapping's digit extraction procedure. We list the case simplified to the plaintext space being a prime $p \neq 2$.

Lemma 2.3 ([HS15]). *Let $p > 1$, $r \geq 1$, and $\tilde{q} = p^r + 1$ be integers with p being an odd prime. Let z be an integer such that $|z/q| + |[z]_q| < (q-1)/2$. Then,*

$$[z]_q = z\langle 0 \rangle - z\langle r \rangle \bmod p.$$

Next, we list a lemma which summarizes the complexity to perform homomorphic digit extraction in BFV and BGV. Digit extraction is a crucial step in these schemes' state-of-the-art bootstrapping algorithms.

Lemma 2.4 ([CH18]). *Let p be prime, $v < e$ be positive integers, u be an integer input modulo p^e with digits*

$$u = u\langle e-1, \ldots, 0 \rangle = \sum_{i=0}^{e-1} u_i p^i.$$

Then, there is an algorithm with $\sqrt{2pev}$ multiplications and arithmetic depth $v \log p + \log e$ which returns

$$u\langle e-1, \ldots, v \rangle = \sum_{i=v}^{e-1} u\langle i \rangle p^i.$$

2.3 Bootstrapping Circuits for BGV and CKKS

Here we describe the state of the art in BGV (BFV) and CKKS bootstrapping. The linear portion of the RLWE decryption function is $c_0 + c_1 s \bmod q$ for both BGV and CKKS. However, BGV further performs a modulo p operation, for plaintext p coprime to q, so

$$m = (c_0 + c_1 s \bmod q) \bmod p = (m + pe) \bmod p.$$

CKKS takes a ciphertext at the lowest level, $(c_0, c_1) \in R_q^2$ just like BGV, but implicitly treats the ciphertext as a high level ciphertext in R_Q^2. Then,

$$c_0 + c_1 s \bmod q = \Delta m + e + Iq$$

for some small polynomial $I = I(X) \in R$. Therefore, the majority of CKKS bootstrapping is spent computing the $y \mapsto y \bmod q$ homomorphically

Both CKKS and BGV use plaintext packing which enables SIMD arithmetic. CKKS packing is given by the canonical embedding modulo complex conjugation: $a(X) \in R$ can be represented by $(a(\delta), a(\delta^{i_1}), \dots, a(\delta^{i_{N/2-1}})) \in \mathbb{C}^{N/2}$ where δ is a complex root of $X^N + 1$ and $i_1, \dots, i_{N/2-1}$ are representatives of $\mathbb{Z}_{2N}^* / \{\pm 1\}$. BGV ciphertext packing is given by the analogous representation modulo p: $(a(\zeta^{k_1}), a(\zeta^{k_2}), \dots, a(\zeta^{k_j})) \in \mathbb{F}_{p^d}^j$ where $\zeta \in \mathbb{Z}_p$ is a root of $X^N + 1 \bmod p$, d is the order of p in $\mathbb{Z}_{2N}^*$, and $\{\zeta^{k_j}\}$ are coset representatives of $\mathbb{Z}_{2N}^* / \langle p \rangle$ [HS20]. Switching between coefficient reprensetation and these evaluation representations is done by linear transformations over $\mathbb{C}$ and $\mathbb{Z}_p$, respectively. Both can be performed homomorphically on a ciphertext, where the former is approximate. These linear transformations are a key step in bootstrapping and can be evaluated with constant multiplicative depth [HS14a, CHK+18].

BGV. In more detail, BGV bootstrapping is given by:

1. Modulus-switch to a special ciphertext modulus of the form $\tilde{q} = p^r + 1$ [HS15].
2. Perform a homomorphic inner-product with an encrypted version of the secret key at the highest level (also called the bootstrapping hint). Here, the input ciphertext is treated as a plaintext to the bootstrapping hint.
3. Unpack the ciphertext with a homomorphic linear transformation (constant depth, but time-intensive). This moves the ciphertext coefficients to the ciphertext slots. Depending on parameters, these values may require multiple ciphertexts.
4. Homomorphically compute the function $x \mapsto (x \bmod \tilde{q}) \bmod p$ via the state-of-the-art digit extraction polynomials [CH18].
5. Repack the ciphertext with the inverse of the homomorphic linear transformation from the second step above with respect to the smaller, original plaintext modulus p.

See [GV22] for more details and the state of the art in BGV and BFV bootstrapping.

CKKS. For CKKS, we perform the following:

1. Treat the ciphertext as a larger modulus ciphertext ($Q \gg q$).
2. Unpack the ciphertext with a homomorphic linear transformation. This moves the coefficients into the plaintext slots. If the ciphertext is fully packed, this will need to output 2 ciphertexts to store the N coefficients. If $\leq N/4$ slots are used, then we can fit all the coefficients in the slots of a single ciphertext. See [CHK+18] for more details. This step is referred to as CoeffsToSlots.
3. Compute an approximation of the $y \mapsto y \bmod q$ homomorphically ([CHK+18, CCS19, HK20, LLL+21, JM20, JM22, LLK+22]).
4. Re-pack the ciphertext with a homomorphic linear transformation. This step is referred to as SlotsToCoeffs.

Another optimization in CKKS bootstrapping was recently given in [BCC+22] where they treat bootstrapping as a black box and bootstrap twice in order to reduce the error induced strictly from bootstrapping. Recently, [KPK+22] showed how to save a few ciphertext levels in the overall CKKS bootstrapping procedure by computing the EvalRound function homomorphically instead of EvalMod.

3 Homomorphic Scheme-Switching

In this section, we define variants of scheme-switching oracles. First, we define weak scheme-switching oracles. A weak scheme-switching oracle takes a CKKS (resp. BGV) ciphertext encrypting a ring polynomial $m(X)$ and outputs a BGV (resp. CKKS) ciphertext encrypting the same ring polynomial $m(X)$. We call the scheme-switching oracles *weak* because they fix the encrypted message in its coefficient form, instead of the evaluation representation (slots), and do not handle the fact that BGV and CKKS have different message encodings. In other words, we would still have to call the CKKS (resp. BGV) slots-to-coefficients function before calling the oracle and re-pack the BGV ciphertext homomorphically after calling the oracle. Next, we define strong scheme-switching oracles which switch packed ciphertexts. We conclude with showing how to transform a weak oracle into a strong oracle with homomorphic linear transformations.

3.1 Weak Scheme-Switching Oracles

Here we define two weak scheme-switching oracles needed for our main result: one that takes a BGV ciphertext encrypting some message polynomial $m(X)$ and outputs a CKKS ciphertext for $m(X)$ (with respect to some scaling factor Δ) and another which takes a CKKS ciphertext encrypting some message polynomial $m(X)$ (with respect to some scaling factor Δ) and outputs a BGV ciphertext encrypting $m(X)$. Ideally, these oracles are black boxes parameterized by BGV and CKKS parameters which have the same functionality as decrypting a BGV (resp. CKKS) ciphertext and re-encrypting the message polynomial in CKKS (resp. BGV) without direct access to the secret key.

Regarding noise and ciphertext moduli, these oracles potentially lower the quality (i.e., the noise magnitude to modulus ratio) of the ciphertexts they convert, just as a real FHE computation would. We have the oracles take as input a ciphertext with noise from an error distribution χ_{in} and return a ciphertext with noise from an error distribution χ. Note that an oracle returning a ciphertext with a smaller modulus and the same noise magnitude as the input is analogous to an oracle returning a ciphertext with the same ciphertext modulus but with a larger noise magnitude since we can always modulus-switch or rescale the oracle's output without the secret key.

First, we define the weak scheme-switching oracle from BGV to CKKS:

Definition 3.1. Let $\mathcal{O}_{B\hookrightarrow C}(\mathsf{ct}_{\mathsf{in}}; p, \Delta, Q, Q', \chi_{\mathsf{in}}, \chi)$ denote the *BGV-to-CKKS oracle* that takes as input a BGV ciphertext $\mathsf{ct}_{\mathsf{in}} = (c_0, c_1)$ encrypting some m, i.e., $c_0 + c_1 s \bmod Q = m + pe$ for error distributed as $e \sim \chi_{\mathsf{in}}$, and is parameterized by a BGV plaintext modulus $p \in \mathbb{Z}^+$, a CKKS scaling factor Δ, an input ciphertext modulus Q, an output ciphertext modulus Q', an input noise distribution χ_{in}, and an output noise distribution χ, potentially a randomized function of χ_{in}, $\chi = f(\chi_{\mathsf{in}})$. The oracle returns a CKKS encryption of m, $\mathsf{ct}_{\mathsf{out}} = (c'_0, c'_1) \in R^2_{Q'}$, i.e., $c'_0 + c'_1 s \bmod Q' = \Delta m + e_\chi$, for a potentially smaller modulus $Q' \leq Q$ and an error e_χ, where $e_\chi = f(e)$, under the same secret key as the input ciphertext.

Observe that since we allow f to be a randomized function in the above definition, Def. 3.1 captures instantiations of the oracle where χ is independent of χ_{in}, and f simply ignores the input noise distribution χ_{in}. However, by defining the oracle in this manner, we also capture situations where χ is dependent on χ_{in}, which is also possible depending on the oracle instantiation. The other scheme-switching oracles in this paper are defined analogously for the same reason.

Next, we define the analogous oracle for switching from CKKS to BGV.

Definition 3.2. Let $\mathcal{O}_{C\hookrightarrow B}(\mathsf{ct}_{\mathsf{in}}; \Delta, p, Q, Q', \chi_{\mathsf{in}}, \chi)$ denote the *CKKS-to-BGV oracle* that takes as input a CKKS ciphertext $\mathsf{ct}_{\mathsf{in}} = (c_0, c_1) \in R^2_Q$ encrypting some m, i.e., $c_0 + c_1 s \bmod Q = \Delta m + e$ for error distributed as $e \sim \chi_{\mathsf{in}}$, a CKKS scaling factor Δ, and is parameterized by a a BGV plaintext modulus $p \in \mathbb{Z}^+$, an input ciphertext modulus Q, an output ciphertext modulus Q', an input noise distribution χ_{in}, and an output noise distribution χ, potentially a randomized function of χ_{in}, $\chi = f(\chi_{\mathsf{in}})$. The oracle returns a BGV encryption of m, $\mathsf{ct}_{\mathsf{out}} = (c'_0, c'_1) \in R^2_{Q'}$ where $c'_0 + c'_1 s \bmod Q' = m + pe_\chi$, for a potentially smaller modulus $Q' \leq Q$ and an error e_χ, where $e_\chi = f(e)$, under the same secret key as the input ciphertext.

Observe that in order to preserve $m(X)$, the oracles must be called with appropriate parameters. For example, when calling $\mathcal{O}_{B\hookrightarrow C}$, if $\|e_X\|_\infty > \Delta$, then the least significant bits of $m(X)$ will be destroyed by the error (we would only have an approximation of $m(X)$). This is commonplace in CKKS encryption and why CKKS is often referred to as an "approximate" FHE scheme. When calling $\mathcal{O}_{C\hookrightarrow B}$, the returned ciphertext is a BGV encryption of $m(X) \bmod p$. Thus, the higher-order bits of the coefficients of $m(X)$ will be lost if their magnitude is larger than $p/2$.

3.2 Strong Scheme-Switching Oracles

Here we discuss strong scheme-switching oracles, which take packed ciphertexts as input and return a packed ciphertext in another SIMD scheme. First, we define strong scheme-switching oracles. Then, we show how to transform a weak scheme-switching oracle into a strong one via homomorphic linear transformations.

Recall the BGV plaintext encoding in its simplest form, where p is a prime with $p = 1 \bmod 2N$. Here, the RLWE plaintext space is $\mathbb{Z}_p[X]/(X^N+1)$, and X^N+1 splits modulo p. This means that $\mathbb{Z}_p[X]/(X^N+1) \cong \mathbb{Z}_p^N$ by evaluating a polynomial at the primitive roots of unity modulo p via the discrete Fourier transform (DFT) over the field. Let $\mathbf{D}$ denote this matrix. Then, given an input vector $\mathbf{v} \in \mathbb{Z}_p^N$, one packs it into a polynomial by treating it as the evaluation of some polynomial over the roots of unity and recovers the polynomial by taking the inverse modulo p: $m(X) := \mathbf{D}^{-1}\mathbf{v}$. In general, the cyclotomic polynomial might not split modulo p, but it will factor into degree k polynomials. Then, the plaintext space is $\mathbb{F}_{p^k}^{\ell}$ where $N = k \cdot \ell$, and the packing is similar. See Appendix C of [GHS12b] or [HS20] for the full details. Given a vector $\mathbf{v} \in \mathbb{Z}_p^N$, we denote the BGV encoding algorithm as $m(X) = \mathsf{encode}_{\mathsf{BGV}}(\mathbf{v})$ with $m(X) \in R_p$. The inverse is denoted by $\mathbf{v} = \mathsf{decode}_{\mathsf{BGV}}(m(X))$.

CKKS packing is done analogously but with the primitive complex roots of unity. Here, the roots of X^N+1 over $\mathbb{C}$ are all N primitive roots of unity of order $2N$. Evaluating a polynomial at all of these roots leads to an element in the conjugate space $\mathbb{H} = \{\mathbf{z} = (z_j)_{j\in\mathbb{Z}_{2N}^*} : z_j = \overline{z_{-j}}\} \subset \mathbb{C}^N$ where one half of the vector $\mathbf{z}$ is the conjugate of the other half and $\mathbb{Z}_{2N}^*$ denotes the unit group of $\mathbb{Z}_{2N}$. Call this map τ. CKKS encoding only uses one half of these vectors by projecting down to $\mathbb{C}^{N/2}$. Call this projection π. CKKS encoding is the inverse of this process: given a vector $\mathbf{v} \in \mathbb{C}^{N/2}$, take the inverse DFT corresponding to the projected portion of the symmetric space $\mathbb{H}$. Since this only gives us an element in $\mathbb{R}[X]/(X^N+1)$, CKKS encoding scales and rounds this element to $R = \mathbb{Z}[X]/(X^N+1)$. This is done in the canonical embedding space $\mathbb{H}$ by scaling up by some scaling factor Δ and rounding to the ring R: $m(X) = \lfloor \tau^{-1}(\Delta\pi^{-1}(\mathbf{v}))\rceil$ where the projection π is invertible on the image of $\mathbb{H}$. Further, we often write the scaled CKKS message as $\Delta m(X)$ by linearity and since we can add the rounding error into the RLWE error. Representing the message as $\Delta m(X)$ helps gauge the number of bits left for homomorphic operations, roughly $\log(Q) - \log(\Delta)$ for small m. Given a vector $\mathbf{v} \in \mathbb{C}^{N/2}$, we denote the CKKS encoding algorithm as $m(X) = \mathsf{encode}_{\mathsf{CKKS}}(\mathbf{v}, \Delta)$, $m(X) \in R$. The (lossy) inverse is denoted by $\mathbf{v}' = \mathsf{decode}_{\mathsf{CKKS}}(m(X), \Delta)$.

Given a BGV or CKKS ciphertext encrypting a packed polynomial, one can homomorphically rotate the slots by performing a field automorphism σ to the ciphertext. For example, if $\mathsf{ct} = (c_0, c_1) \in R_Q^2$ is a BGV ciphertext encrypting $m(X) = \mathsf{encode}_{\mathsf{BGV}}(\mathbf{v})$, then $(\sigma_i(c_0), \sigma_i(c_1))$ encrypts

$$m'(X) = \sigma_i(m) = \mathsf{encode}_{\mathsf{BGV}}((v_i, v_{i+1}, \ldots, v_{N-1-i \bmod N}))$$

under the secret key $\sigma_i(s)$. That is, σ_i is the Galois automorphism that cyclically shifts the vector by i positions. We then key-switch back to a ciphertext encrypted under s using an operation similar to relinearization.(Relinearization is a special case of key-switching.)

Now consider a strong homomorphic scheme-switching algorithm. Given a BGV ciphertext $(as+pe+m, -a)$ where $m = m(X) = \mathsf{encode}_{\mathsf{BGV}}(\mathbf{v})$ encodes N elements in $\mathbb{Z}_p$, the goal is to switch to either one or two packed CKKS ciphertexts encrypting the elements of v. We say potentially two ciphertexts since we expect applications to use BGV for exact multiplication before overflow modulo p and the spaces $\mathbb{C}$ and $\mathbb{F}_p^2$ are incompatible in terms of arithmetic operations. For simplicity, assume we want to pack them into one CKKS ciphertext. Then, given $(as+pe+m, -a)$, the output is a ciphertext $(a's+e'+\Delta m', -a')$

where $m' = m'(X) = \mathsf{encode}_{\mathsf{CKKS}}(\mathbf{v}, \Delta)$. This yields the following definitions.

We denote $\Re(y)$ and $\Im(y)$ as the respective real and imaginary part of a complex number, $y \in \mathbb{C}$, and $\Re(\mathbf{y}), \Im(\mathbf{y})$ as the function applied component-wise to a vector $\mathbf{y} \in \mathbb{C}^m$. We define the strong scheme-switching oracles below:

Definition 3.3. Let $\mathcal{O}^{\mathsf{strong}}_{B\hookrightarrow C}(\mathsf{ct}_{\mathsf{in}}; p, \Delta, Q, Q', \chi_{\mathsf{in}}, \chi)$ denote the *strong BGV-to-CKKS oracle* that takes as input a BGV ciphertext $\mathsf{ct}_{\mathsf{in}} = (c_0, c_1)$ encrypting some m, i.e., $c_0 + c_1 s \bmod Q = m + pe$ for $m = m(X) = \mathsf{encode}_{\mathsf{BGV}}(\mathbf{v})$, $\mathbf{v} \in \mathbb{Z}_p^N$ with error distributed as $e \sim \chi_{\mathsf{in}}$, and is parameterized by a BGV plaintext modulus $p \in \mathbb{Z}^+$, a CKKS scaling factor Δ, an input ciphertext modulus Q, an output ciphertext modulus Q', an input noise distribution χ_{in}, and an output noise distribution χ, potentially a randomized function of χ_{in}, $\chi = f(\chi_{\mathsf{in}})$. The oracle returns a CKKS encryption of m, $\mathsf{ct}_{\mathsf{out}} = (c_0', c_1') \in R_{Q'}^2$, i.e., $c_0' + c_1' s \bmod Q' = \Delta m' + e_\chi$ where $m' = \mathsf{encode}_{\mathsf{CKKS}}(\mathbf{v}, \Delta)/\Delta$, for a potentially smaller modulus $Q' \leq Q$ and an error e_χ, where $e_\chi = f(e)$, under the same secret key as the input ciphertext.

Definition 3.4. Let $\mathcal{O}^{\mathsf{strong}}_{C\hookrightarrow B}(\mathsf{ct}_{\mathsf{in}}; \Delta, p, Q, Q', \chi_{\mathsf{in}}, \chi)$ denote the *CKKS-to-BGV oracle* that takes as input a CKKS ciphertext $\mathsf{ct}_{\mathsf{in}} = (c_0, c_1) \in R_Q^2$ encrypting some m, i.e., $c_0 + c_1 s \bmod Q = \Delta m + e$ for $m = m(X) = \mathsf{encode}_{\mathsf{CKKS}}(\mathbf{v}, \Delta)/\Delta$, $\mathbf{v} \in \mathbb{C}^{N/2}$ with error distributed as $e \sim \chi_{\mathsf{in}}$, a CKKS scaling factor Δ, and is parameterized by a a BGV plaintext modulus $p \in \mathbb{Z}^+$, an input ciphertext modulus Q, an output ciphertext modulus Q', an input noise distribution χ_{in}, and an output noise distribution χ, potentially a randomized function of χ_{in}, $\chi = f(\chi_{\mathsf{in}})$. The oracle returns a BGV encryption of m, i.e., $\mathsf{ct}_{\mathsf{out}} = (c_0', c_1') \in R_{Q'}^2$ where $c_0' + c_1' s \bmod Q' = m' + pe_\chi$ with $m' = \mathsf{encode}_{\mathsf{BGV}}(\mathbf{v}')$ where $\mathbf{v}' = (\Re(\mathbf{v}), \Im(\mathbf{v}))$, for a potentially smaller modulus $Q' \leq Q$ and an error e_χ, where $e_\chi = f(e)$, under the same secret key as the input ciphertext.

Next, we show how to take a weak scheme-switcher and transform it into a strong scheme-switcher using homomorphic linear transformations. We will use the following two lemmas on homomorphically encoding and decoding in CKKS and BGV. The following algorithms/lemmas are given in terms of minimal plaintext-ciphertext depth for simplicity. Note, homomorphically applying the BGV and CKKS encoding/decoding algorithms can be done with homomorphic linear transformations where the matrix is stored as plaintexts polynomials. In the case of CKKS, this is an approximate linear transformation due to the CKKS encoding function. We summarize this in Lemmas 3.1 and 3.2 below.

Lemma 3.1 ([HS14a]). *Given a BGV ciphertext* $\mathsf{ct} = (c_0, c_1) \in R_Q^2$, $c_0 + c_1 s = m + pe$ *for* $m(X) = \mathsf{encode}_{\mathsf{BGV}}(\mathbf{v})$, $\mathbf{v} \in \mathbb{Z}_p^N$*, together with its public evaluation key evk, one can homomorphically transform* ct *into a BGV ciphertext* ct'*, under the same secret key, encrypting* $\mathbf{v}$ *in the coefficients of* $m'(X)$*, in a plaintext-ciphertext multiplicative depth one circuit and* $N/2$ *ciphertext rotations.*

Lemma 3.2 (Algorithm 1 in [CHK+18]). *Given a CKKS ciphertext* $\mathsf{ct} = (c_0, c_1) \in R_Q^2$, $c_0 + c_1 s = \Delta m + e$ *for* $m(X) = \mathsf{encode}_{\mathsf{CKKS}}(\mathbf{v}, \Delta)/\Delta$, $\mathbf{v} \in \mathbb{C}^{N/2}$*, together with its public evaluation key evk, one can homomorphically transform* ct *into a CKKS ciphertext* $\mathsf{ct}' \in R_{Q'}^2$*, under the same secret key, encrypting* $\mathbf{v}$ *in the coefficients of* $m'(X)$*, in a plaintext-ciphertext multiplicative depth one circuit and* $N/2$ *ciphertext rotations. The resulting ciphertext modulus* $Q' \leq Q$ *depends on the precision to which this approximate computation is performed, which a more precise computation resulting in* ct' *having a smaller* Q'*.*

See Chen et al. [CCS19] or Han et al. [HHC19] for a faster algorithm for homomorphic encoding and decoding using an FFT-like algorithm with a deeper circuit. Further efficiency optimizations can be found in [BMTPH21]. The algorithms in Lemmas 3.1 and 3.2 encode each linear transformation's matrix by N (resp., $N/2$) diagonals as plaintext polynomials and apply the matrix-vector multiplication homomorphically. See [HS14a, CHK+18] for more details.

If the input ciphertext in Lemma 3.1 (resp., Lemma 3.2) has distribution χ_{in}, then denote the output distribution as $T_B(\chi_{\mathsf{in}})$ (resp., $T_C(\chi_{\mathsf{in}})$). In practice, the noise growth from applying the homomorphic linear transformations is relatively small, but the homomorphic computation involved is time-intensive.

We now show how to take a weak scheme-switching oracle and turn it into a strong scheme-switching oracle at the cost of two levels of plaintext-ciphertext multiplicative depth.

Theorem 3. *Given a weak scheme-switching oracle from BGV to CKKS,* $\mathcal{O}_{B\hookrightarrow C}(\mathsf{ct}_{\mathsf{in}}; p, \Delta, Q, Q', T_B(\chi_{\mathsf{in}}), \chi)$*, there exists a strong scheme-switching oracle from BGV to CKKS,* $\mathcal{O}^{\mathsf{strong}}_{B\hookrightarrow C}(\mathsf{ct}_{\mathsf{in}}; p, \Delta, Q, Q'', \chi_{\mathsf{in}}, T_C(\chi))$*, where* Q'' *is related to* Q' *by the instantiation of Lemma 3.2.*

Proof. Let $\mathsf{ct}'' = (c_0'', c_1'') \in R_Q^2$ be a BGV ciphertext where $c_0'' + c_1''s = m'' + pe''$ and $m'' = m''(X) = \mathsf{encode}_{\mathsf{BGV}}(\mathbf{v})$ for some vector $\mathbf{v} \in \mathbb{Z}_p^N$. Let χ'' denote the input error's distribution, $e'' \sim \chi''$. Compute the following:

1. Run the algorithm in Lemma 3.1 to get a ciphertext $\mathsf{ct}' = (c_0', c_1')$ where $c_0' + c_1's = m' + pe'$ encrypts $\mathbf{v}$ as the coefficients in m': $m'(X) = v_0 + v_1X + \cdots + v_{N-1}X^{N-1}$. Denote the error distribution of ct' by $T_B(\chi'')$.
2. Call the weak scheme-switching oracle on ct',
$$\hat{\mathsf{ct}} \leftarrow \mathcal{O}_{B\hookrightarrow C}(\mathsf{ct}'; p, \Delta, Q, Q', T_B(\chi''), \chi),$$
to get $\hat{\mathsf{ct}} = (\hat{c}_0, \hat{c}_1) \in R_{Q'}^2$ where $\hat{c}_0 + \hat{c}_1 s = \Delta m' + \hat{e}$.
3. Let $\mathbf{v}' \in \mathbb{C}^{N/2}$ denote $\mathbf{v}$'s entries stored as a complex vector. Run the algorithm in Lemma 3.2 to get $\mathsf{ct} = (c_0, c_1) \in R_{Q''}^2$ where $c_0 + c_1 s = \Delta m + e$ and $m = m(X) = \mathsf{encode}_{\mathsf{CKKS}}(\mathbf{v}', \Delta)/\Delta$. The error distribution of e is $T_C(\chi)$.

□ □

Theorem 4. *Given a weak scheme-switching oracle from CKKS to BGV,* $\mathcal{O}_{C\hookrightarrow B}(\mathsf{ct}_{\mathsf{in}}; \Delta, p, Q', Q'', T_C(\chi_{\mathsf{in}}), \chi)$, *there exists a strong scheme-switching oracle from CKKS to BGV,* $\mathcal{O}^{\mathsf{strong}}_{C\hookrightarrow B}(\mathsf{ct}_{\mathsf{in}}; \Delta, p, Q, Q'', \chi_{\mathsf{in}}, T_B(\chi))$, *where* Q' *is related to* Q *by the instantiation of Lemma 3.2.*

Proof. The proof is analogous to that of Theorem 3. □ □

4 Bootstrapping Via A Weak Scheme-Switching Oracle

In this section, we show our main results that one can bootstrap a BGV (resp., CKKS) ciphertext using a single call to a weak scheme-switching oracle *without* computing homomorphic linear transformations. Recall that since BFV ciphertexts can be simply converted to BGV ciphertexts via scalar multiplication [AP13], our result for BGV also applies to BFV.

We show two directions:

1. Using a BGV to CKKS weak scheme-switching oracle, one can bootstrap a CKKS ciphertext using one oracle query and
2. Using a CKKS to BGV weak scheme-switching oracle, one can bootstrap a BGV ciphertext using one oracle query.

To bootstrap a CKKS ciphertext, we only need to perform one CKKS rescaling in addition to the oracle call. To bootstrap a BGV ciphertext, we only need to perform a homomorphic inner product and one BGV modulus-switching in addition to the oracle call. Thus, we provide powerful evidence that scheme-switching is at least as hard as bootstrapping and improvements to scheme-switching should lead to improvements to bootstrapping.

4.1 Bootstrapping in CKKS from a BGV-to-CKKS Oracle

We begin by showing that a weak BGV-to-CKKS scheme-switching oracle allows one to bootstrap a CKKS ciphertext immediately. As mentioned above, this allows the user to bootstrap a CKKS ciphertext *without* the costly coefficients-to-slots homomorphic linear transformation or its inverse. In the reduction, we are calling $\mathcal{O}_{B\hookrightarrow C}$ with a BGV plaintext modulus equal to the input ciphertext modulus q. We emphasize that treating the lowest level ciphertext modulus as a plaintext modulus is common in BGV bootstrapping. See [GHS12a, HS15] for more details.

Theorem 5. *Fix some CKKS parameters:* $R = \mathbb{Z}[X]/(X^N + 1)$, $\Delta, \Delta_{\mathsf{in}}$ *positive integers, and* $q \ll Q'' \leq Q' \leq Q$ *as ciphertext moduli with* $q \mid \Delta Q'' = Q' \mid Q$. *Let the ciphertext* $\mathsf{ct} \in \mathsf{CKKS}^s(m)_{\Delta_{\mathsf{in}}, q}$ *be the input. Then, the existence of a BGV to CKKS weak scheme-switching oracle,* $\mathcal{O}_{B\hookrightarrow C}(\,\cdot\,; q, \Delta, Q, Q', \chi', \chi)$, *implies the existence of a CKKS bootstrapping algorithm where the time to bootstrap is the time complexity of the oracle plus the time complexity to rescale from* Q' *to* Q'/Δ, *and the output ciphertext has noise at most* $\|e\|_\infty + \frac{\|e_\chi\|_\infty}{\Delta} + (N\|s\|_\infty + 1)/2$, *where* e *is the error term of* ct *and* e_χ *is the error term in the output of* $\mathcal{O}_{B\hookrightarrow C}$.

Proof. Let $\mathsf{ct} = (c_0, c_1) \in R_q^2$ be the input ciphertext in $\mathsf{CKKS}^s(m)_{\Delta_{\mathsf{in}},q}$. That is, the ciphertext satisfies $c_0 + c_1 s \bmod q = \Delta_{\mathsf{in}} m + e$. By embedding c_0, c_1 into R_Q, it follows that this new ciphertext with modulus Q satisfies $c_0 + c_1 s \bmod Q = \Delta_{\mathsf{in}} m + e + I(X)q$ where $I(X)$ is an integer polynomial that depends on the Hamming weight and norm of the secret key, $s \in R$. Note that $I(X) \cdot q \ll Q$ for common parameter settings. Then, the reduction simply calls the weak scheme-switching oracle with BGV plaintext $p = q$, where χ' is the implicit error distribution of viewing ct as a BGV ciphertext in the above manner.

Let
$$\mathsf{ct}^{(1)} = \mathcal{O}_{B \hookrightarrow C}(\mathsf{ct}; q, \Delta, Q, Q', \chi', \chi).$$
Notice that $\mathsf{ct}^{(1)} \in R_{Q'}^2$ satisfies
$$c_0^{(1)} + c_1^{(1)} s \mod Q' = \Delta(\Delta_{\mathsf{in}} m + e) + e_\chi$$
where $e_\chi \sim \chi$. Therefore, we simply call the CKKS rescale function and return the result $\mathsf{ct}^{(2)} \leftarrow$CKKS.$\mathsf{Rescale}(\mathsf{ct}^{(1)}, \Delta)$. The output $\mathsf{ct}^{(2)} \in R_{Q''}^2$, $Q'' = Q'/\Delta$, now satisfies $c_0^{(2)} + c_1^{(2)} s \bmod Q'' = \Delta_{\mathsf{in}} m + e''$ with $\|e''\|_\infty \leq \|e\|_\infty + \frac{\|e_\chi\|_\infty}{\Delta} + (N\|s\|_\infty + 1)/2$ by the rescaling lemma, Lemma 2.2. An illustration of the proof is given in Figure 2. □ □

4.2 Bootstrapping in BGV from a CKKS-to-BGV Oracle

In this subsection, we show how to use a weak CKKS-to-BGV scheme-switching oracle to bootstrap a BGV ciphertext. At a high level, bootstrapping a BGV ciphertext consists of linear operations and then digit extraction. The digit extraction step involves taking a BGV encryption of a message z with plaintext modulus p^{r+1} and computing a BGV encryption of the message $z\langle r\rangle$ with plaintext modulus p, where $z\langle r\rangle$ is the rth digit in the base-p decomposition of z. By viewing the BGV encryption of z as a CKKS encryption with $\Delta = p^r$, the CKKS-to-BGV oracle will output a BGV encryption of $z\langle r\rangle$ with plaintext modulus p.

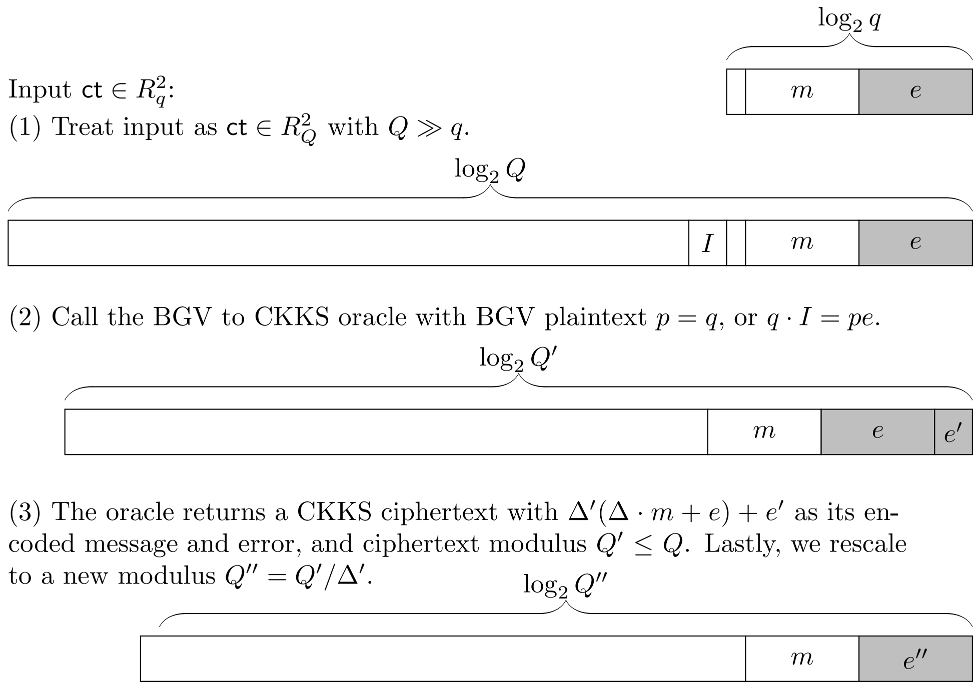
Theorem 6 below uses the same bootstrapping hint as the state-of-the-art BGV bootstrapping algorithm. That is, if we wish to refresh a ciphertext $\mathsf{ct} = (c_0, c_1) \in \mathsf{BGV}^s(m)_{p,q}$ to a larger modulus Q, we first modulus-switch to a smaller modulus of a special form, $\tilde{q} = p^r + 1$, yielding a ciphertext in $\mathsf{BGV}^s(m)_{p,\tilde{q}}$. Then, we use a bootstrapping hint, an encryption of the secret key under the target ciphertext modulus and a larger plaintext modulus $\mathsf{BGV}^s(s)_{p^{r+1},Q}$, to get the encrypted inner product in the larger ciphertext space $\mathsf{BGV}^s(c_0 + c_1 s)_{p^{r+1},Q}$. This is done by treating the ciphertext in $\mathsf{BGV}^s(m)_{p,\tilde{q}}$ as *plaintext* to the bootstrapping hint in $\mathsf{BGV}^s(s)_{p^{r+1},Q}$. See Halevi and Shoup's 2015 paper, [HS15], for the full details.

Theorem 6. *Fix some BGV parameters: $R = \mathbb{Z}[X]/(X^N + 1)$, p a prime, and $\tilde{q} = p^r + 1 \ll Q' \leq Q$ are ciphertext moduli. Let $\mathsf{ct} \in \mathsf{BGV}^s(m)_{p,\tilde{q}}$ be the input ciphertext and let $\mathsf{hint} = (h_0, h_1) \in \mathsf{BGV}^s(s)_{p^{r+1},Q}$ be a bootstrapping hint where $s \in R$ is ct's secret key. Then, the existence of a CKKS to BGV weak scheme-switching oracle, $\mathcal{O}_{C \hookrightarrow B}(\ \cdot\ ; \Delta, p, Q, Q', \chi', \chi)$, implies the existence of a BGV bootstrapping algorithm with time complexity that of the oracle plus the time complexity to modulus-switch from Q to Q' and the time complexity to perform a homomorphic inner product between ct and hint, where ct's polynomials are treated as plaintext to the hint. The output ciphertext has noise as most $\frac{Q'}{Q}\tilde{q}^2\|e_{\mathsf{hint}}\|_\infty + (1 + N\|s\|_\infty)/2 + e_\chi$, where e_{hint} is the noise of hint and e_χ is the noise of the output of $\mathcal{O}_{C \hookrightarrow B}$.*

Proof. Let $\mathsf{ct} = (c_0, c_1) \in \mathsf{BGV}^s(m)_{p,\tilde{q}}$ be the input ciphertext and $\mathsf{hint} = (h_0, h_1) \in \mathsf{BGV}^s(s)_{p^{r+1},Q}$ be the bootstrapping hint for ct's secret key, $s \in R$. The reduction is as follows.

1. First, perform the homomorphic inner product to get an encryption of $z = \langle \mathsf{ct}, (1, s) \rangle$:
$$\mathsf{ct}^{(1)} = c_1 \cdot \mathsf{hint} + (c_0, 0) \in \mathsf{BGV}^s(z)_{p^{r+1},Q}.$$

2. Next, call the oracle on the resulting ciphertext with $\Delta = p^r$, where χ' is the implicit error distribution of $\mathsf{ct}^{(1)}$ when viewed as a CKKS ciphertext:
$$\mathsf{ct}^{(2)} \leftarrow \mathcal{O}_{C \hookrightarrow B}(\mathsf{ct}^{(1)}; p^r, p, Q, Q', \chi', \chi).$$



Input $\mathsf{ct} \in R_q^2$:

(1) Treat input as $\mathsf{ct} \in R_Q^2$ with $Q \gg q$.

(2) Call the BGV to CKKS oracle with BGV plaintext $p = q$, or $q \cdot I = pe$.

(3) The oracle returns a CKKS ciphertext with $\Delta'(\Delta \cdot m + e) + e'$ as its encoded message and error, and ciphertext modulus $Q' \leq Q$. Lastly, we rescale to a new modulus $Q'' = Q'/\Delta'$.

Figure 2: CKKS bootstrapping using a BGV to CKKS weak scheme-switching oracle. Our input is a CKKS ciphertext with small modulus q. We call the weak scheme-switching oracle on the input ciphertext with BGV plaintext modulus equal to the input CKKS ciphertext modulus.

3. View $\mathsf{ct}^{(1)}$ as a ciphertext with plaintext modulus p and mod switch to Q', $\mathsf{ct}^{(3)} \leftarrow \text{BGV}.\mathsf{ModReduce}(\mathsf{ct}^{(1)}, Q, Q')$, and return the difference

$$\mathsf{ct}_{\mathsf{out}} = \mathsf{ct}^{(2)} - \mathsf{ct}^{(3)} \mod Q' \in \mathsf{BGV}^s(m)_{p,Q'}.$$

Correctness is as follows. Recall, the operation $(z \bmod \tilde{q}) \bmod p$ is $z\langle 0\rangle - z\langle r\rangle \mod p$ by Lemma 2.3. The oracle treats the input ct as a CKKS ciphertext where the least significant digit is in the r-th position (the lower digits are considered as part of the error). Then, we treat $\mathsf{ct}^{(1)}$ as implicitly encrypting $z\langle 0\rangle$ modulo p, and the result follows.

To analyze the error associated with $\mathsf{ct}_{\mathsf{out}}$, let e_{hint} denote the error associated with the ciphertext hint. It follows that $\|e^{(1)}\|_\infty \leq \tilde{q}\|e_{\mathsf{hint}}\|_\infty$. $\mathsf{ct}^{(2)}$ has associated error e_χ given by the oracle and plaintext modulus p. By Lemma 2.1, $\|e^{(3)}\|_\infty \leq \frac{Q'}{Q}\tilde{q}^2\|e_{\mathsf{hint}}\|_\infty + (1+N\|s\|_\infty)/2$. Note that the correctness condition for Lemma 2.1 holds since $\tilde{q}^2\|e_{\mathsf{hint}}\|_\infty \ll Q/2$ for reasonable parameters. Thus,

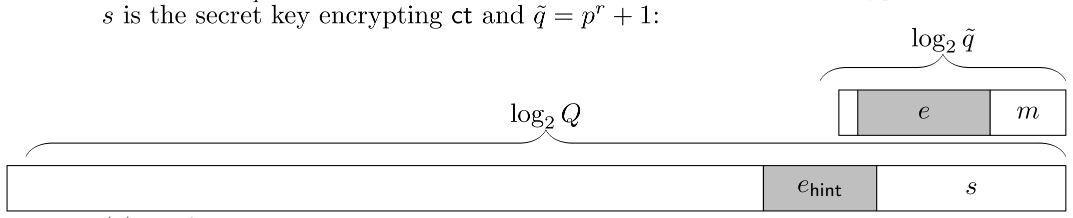
$$\|e_{\mathsf{out}}\|_\infty \leq \frac{Q'}{Q}\tilde{q}^2\|e_{\mathsf{hint}}\|_\infty + (1 + N\|s\|_\infty)/2 + e_\chi.$$

□ □

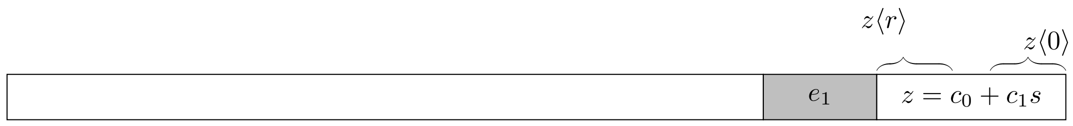
We note that the above proofs adapt easily to the case where the BGV plaintext modulus is a power of p. See Figure 2 and Figure 3 for pictorial illustrations representing the reductions in Theorem 5 and Theorem 6.

4.3 Switching Between Schemes Using Bootstrapping

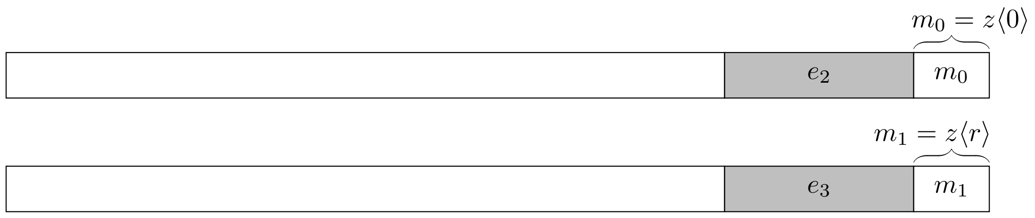
Here, for completeness, we briefly sketch how to switch between schemes using bootstrapping. The main idea is that a coefficient-packed BFV ciphertext can be seen as an "exhausted" coefficient-packed CKKS ciphertext and vice versa. If we have an algorithm that bootstraps a coefficient-packed CKKS ciphertext, Input $\mathsf{ct} \in R_{\tilde{q}}^2$ together with a bootstrapping hint $\mathsf{hint} \in \mathsf{BGV}^s(s)_{Q,p^{r+1}}$ where s is the secret key encrypting ct and $\tilde{q} = p^r + 1$:



(1) Perform the homomorphic inner product where we treat the input ciphetext as two plaintext polynomials to the boostrapping hint. That is, $c_1 \cdot \mathsf{hint} + (c_0, 0) \in R_Q^2$ where $\mathsf{ct} = (c_0, c_1)$.



(2) Label the result as $\mathsf{ct}_{z\langle 0\rangle}$ since it encrypts $z\langle 0\rangle$ modulo p. Then feed the CKKS to BGV oracle a copy of $\mathsf{ct}_{z\langle 0\rangle}$ but with $\Delta := p^{r-1}$. Label the output of the oracle as $\mathsf{ct}_{z\langle r\rangle}$ since it encrypts $z\langle r\rangle$. (Mod-switch $\mathsf{ct}_{z\langle 0\rangle}$ down to the oracle's outputted ciphertext modulus Q' if needed.)



(3) Lastly, subtract $\mathsf{ct}_{z\langle 0\rangle} - \mathsf{ct}_{z\langle r\rangle}$ and mod-switch down to lower the noise. We have that the result is a ciphertext in $\mathsf{BGV}^s(m)_{Q'',p}$ by Lemma 2.3.

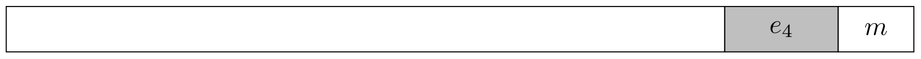


Figure 3: BGV bootstrapping using a CKKS to BGV weak scheme-switching oracle.

which is easily obtained from a SIMD-packed CKKS bootstrapping algorithm via homomorphic linear transformations [HS18], then we can treat a coefficient-packed BFV ciphertext as a CKKS ciphertext and bootstrap using the aforementioned algorithm. The output will be a CKKS ciphertext with the inputted BFV ciphertext's message as its plaintext polynomial. The other direction is analogous. We can treat a coefficient-packed CKKS ciphertext as a coefficient-packed BFV ciphertext and use a BFV bootstrapping algorithm to convert the inputted ciphertext to a BFV ciphertext. Lastly, we can convert between BGV and BFV cheaply with a scalar multiplication [AP13]. This shows how to convert between BGV/BFV and CKKS with bootstrapping.

5 Bootstrapping Via A Comparison Oracle

The SIMD FHE schemes discussed in this work are capable of natively evaluating arithmetic circuits homomorphically in a SIMD fashion over an encrypted vector of plaintext values. However, there are various functions, such as comparison, that are useful to compute for applications, but are not easily expressible as an arithmetic circuit. Several prior works [CKK$^+$19, CKK20, IZ21, LLKN21] have focused on methods of computing comparison (and other related functions such as max/min and ReLU) in these SIMD FHE schemes.

In this section, we explore the relative hardness of homomorphically evaluating the comparison function in these SIMD FHE schemes by showing how to bootstrap in these schemes using several calls to a comparison oracle. In particular, we show how to bootstrap packed CKKS ciphertexts and thinly packed BGV ciphertexts. At a high level, our comparison oracles (one for CKKS and one for BGV) will take as input a ciphertext ct encrypting a vector of plaintext values $(m_1, \ldots, m_t)$ and a value α and output a new ciphertext ct' that encrypts the value 1 in its i-th slot if $m_i \geq \alpha$ and 0 otherwise. We observe that a comparison oracle only outputs the encryption of a single bit in each slot and, thus, it is much weaker than the scheme-switching oracles in Section 3. In particular, we will have to make several calls to the comparison oracle in order to bootstrap (as opposed to only a single call to the weak scheme-switching oracles in Section 4). Moreover, we will also have to apply the homomorphic linear transformations $\mathsf{CoeffsToSlots}$ and $\mathsf{SlotsToCoeffs}$ in order to bootstrap, which was not required previously when using the weak scheme-switching oracles. While we focus on comparison in this section, our approach also extends to related functions such as max/min and ReLU.

To see how a comparison oracle could be used to bootstrap, we will sketch the intuition for the CKKS case as it is slightly simpler than the BGV case. Recall that CKKS bootstrapping begins by taking a ciphertext $\mathsf{ct} \in R_q^2$ and viewing it as a ciphertext with respect to the largest modulus Q. Viewed this way, ct now decrypts to $\Delta m + e + I(X)q$, and thus, we now need to homomorphically compute the mod q function on the coefficients of the encrypted polynomial. If we call the $\mathsf{CoeffsToSlots}$ homomorphic linear transform, we can put the coefficients of this polynomial in the ciphertext slots. Let K denote an upper bound on $\|I\|_\infty$. Then, using $\log K$ calls to a comparison oracle, we can homomorphically compute the mod q function to remove the Iq term. This is done by clearing one bit of I at a time starting from the most significant bit by comparing to an appropriate power of 2 and subtracting the two ciphertexts. Once we have homomorphically computed the mod q function, we apply the $\mathsf{SlotsToCoeffs}$ homomorphic linear transform to finish bootstrapping.

5.1 Comparison Oracles

Here, we define the comparison oracles rigorously in the same manner as in Section 3. The oracles' inputs are a ciphertext $\mathsf{ct} \in R_Q^2$ and a scalar α. For CKKS, the comparison oracle is parameterized by input and output scaling factors Δ, Δ', input and output ciphertext moduli Q, Q', and output error distribution χ. For BGV, the comparison oracle is parameterized by the plaintext modulus p, input and output ciphertext moduli Q, Q', and output error distribution χ. We omit the input error distribution χ_{in} from the parameters of these oracles as it is clear from context. Recall from Section 3 that the output error distribution χ is related to χ_{in} via a randomized function f that is determined by the instantiation of the oracle.

Definition 5.1 (CKKS Comparison Oracle)**.** Let $\mathcal{O}_{\geq}(\mathsf{ct}, \alpha; \Delta, \Delta', Q, Q', \chi)$ denote the CKKS comparison oracle that takes as input a ciphertext $\mathsf{ct} \in \mathsf{CKKS}^s(m)_{\Delta,Q} \subset R_Q^2$ that decrypts to $\Delta m + e = v$ and returns $\mathsf{ct}' \in \mathsf{CKKS}^s(m')_{\Delta',Q'} \subset R_{Q'}^2$. The output ciphertext $\mathsf{ct}' = (c_0', c_1')$ satisfies $c_0' + c_1' s \bmod Q' = \Delta' m' + e_\chi = v'$ where v' is a polynomial with $\mathsf{Slot}_i(v') \approx \Delta'$ if $\mathsf{Slot}_i(v) \geq \Delta \cdot \alpha$ and $\mathsf{Slot}_i(v') \approx 0$ otherwise and $e_\chi \sim \chi$. The error in the approximation of each $\mathsf{Slot}_i(v')$ is determined by the error polynomial e_χ's contribution to the ith slot.

Definition 5.2 (BGV Comparison Oracle)**.** Let $\mathcal{O}_{\geq}(\mathsf{ct}, \alpha; p, Q, Q', \chi)$ denote the BGV comparison oracle that takes as input a ciphertext $\mathsf{ct} \in \mathsf{BGV}^s(m)_{p,Q} \subset R_Q^2$ and returns $\mathsf{ct}' \in \mathsf{BGV}^s(m')_{p,Q'} \subset R_{Q'}^2$. The output ciphertext $\mathsf{ct}' = (c_0', c_1')$ satisfies $c_0' + c_1' s \bmod Q' = m' + p e_\chi$ where m' is a polynomial with $\mathsf{Slot}_i(m') = 1$ if $\mathsf{Slot}_i(m) \geq \alpha$ and $\mathsf{Slot}_i(m') = 0$ otherwise and $e_\chi \sim \chi$.

5.2 Bootstrapping in CKKS from Comparisons

We first show how to bootstrap a CKKS ciphertext using a CKKS comparison oracle. For ease of exposition, we will focus on the case where the ciphertext moduli and scaling factors are powers of two. We show the following theorem.

Theorem 7. *Let R, Q, Δ be CKKS parameters. Let $\mathsf{ct} = (c_0, c_1) \in R_q^2$ be a CKKS ciphertext such that $c_0 + c_1 s = \Delta m + e + I(X)q$ over the ring R with $\|I(X)\|_\infty \leq K$. Then assuming the existence of a list of CKKS comparison oracles $\mathcal{O}_\geq$ from Definition 5.1, with parameters*

$$\mathcal{O}_\geq(\mathsf{ct}'_{i-1}, K/2^i; q, K/2^{i-1} \cdot q, Q_{i-1}, Q_i, \chi)$$

satisfying $Q_i \mid Q_{i-1}$ for $i = 1$ to $\log K$, there is a CKKS bootstrapping algorithm that makes $\log_2 K$ calls to the oracles together with the $\mathsf{CoeffsToSlots}$ *and* $\mathsf{SlotsToCoeffs}$ *homomorphic linear transformations for CKKS bootstrapping.*

Proof. Let $\mathsf{ct} = (c_0, c_1) \in R_q^2$ be the input ciphertext in $\mathsf{CKKS}^s(m)_{\Delta,q}$. That is, the ciphertext satisfies $c_0 + c_1 s \bmod q = \Delta m + e$. By embedding c_0, c_1 into R_Q, it follows that this new ciphertext with modulus Q satisfies $c_0 + c_1 s \bmod Q = \Delta m + e + I(X)q$ where $I(X)$ is an integer polynomial that depends on the Hamming weight and norm of the secret key, $s \in R$. Note that $I(X) \cdot q \ll Q$ for commonly used parameter settings. Let K be an upper bound on the magnitude of a coefficient of $I(X)$. WLOG, assume that K is a power of 2. First, we run the $\mathsf{CoeffsToSlots}$ step of the CKKS bootstrapping procedure (see Sec. 2.3) to obtain a ciphertext $\mathsf{ct}' \in R_{Q'}^2$ that encrypts the coefficients of $\Delta m + e + I(X)q$ in its slots. Note that if ct was a fully packed ciphertext, this would require two ciphertexts to store all the coefficients. For simplicity, we will assume that ct is not fully packed so that the coefficients can all be encrypted in a single ciphertext ct'.

Each coefficient of $I(X)$ can be expressed as $\sum_{i=0}^{\log K - 1} 2^i \cdot I_i$ with $I_i \in \{0, 1\}$. Let $\mathsf{ct}' = \mathsf{ct}'_0$. For $i = 1$ to $\log K$, call

$$\mathcal{O}_\geq(\mathsf{ct}'_{i-1}, K/2^i; q, K/2^{i-1} \cdot q, Q_{i-1}, Q_i, \chi)$$

to obtain ct_i with ciphertext modulus Q_i. Then, mod down ct'_{i-1} to modulus Q_i to obtain ct''_{i-1}. Set $\mathsf{ct}'_i = \mathsf{ct}''_{i-1} - \mathsf{ct}_i$.

Observe that $\mathsf{ct}'_{\log K}$ is a ciphertext with modulus $Q_{\log K}$ that has the coefficients of $\Delta m + e_{\log K}$ in its slots. Call $\mathsf{SlotsToCoeffs}$ to obtain a bootstrapped ciphertext. □ □

5.3 Bootstrapping in BGV from Comparisons

We now show how to bootstrap a BGV ciphertext using a BGV comparison oracle. We show the following theorem.

Theorem 8. *Let R, q, p be BGV parameters and let $\mathsf{ct} \in \mathsf{BGV}^s(m)_{p,q}$ be an input ciphertext. Then, the existence of oracles $\mathcal{O}_\geq(\mathsf{ct}, \alpha; p^{r+1}, Q_{i-1}, Q_i, \chi)$, for $Q_i \gg q$, $p^{r+1} < q$, implies the existence of a BGV bootstrapping algorithm that makes $\lceil \log p \rceil - 1$ calls to the oracles and also computes a homomorphic inner product and the homomorphic linear transformations* $\mathsf{CoeffsToSlots}$ *and* $\mathsf{SlotsToCoeffs}$ *for BGV bootstrapping.*

Proof. First, we calculate the homomorphic inner product with standard techniques: The input is a BGV ciphertext $\mathsf{ct} \in \mathsf{BGV}^s(m)_{p,q}$, and we switch to the modulus $\tilde{q} = p^r + 1$,

$$\mathsf{ct}' \leftarrow \mathsf{ModSwitch}(\mathsf{ct}, \tilde{q}).$$

Using the bootstrapping hint, $\mathsf{hint} \in \mathsf{BGV}^s(s)_{Q,p^{r+1}}$, calculate the homomorphic inner product

$$\mathsf{ct}'' = c'_1 \cdot \mathsf{hint} + (c'_0, 0) \in \mathsf{BGV}^s(z)_{p^{r+1},Q}.$$

Now we have a BGV ciphertext in $\mathsf{BGV}^s(z)_{Q,p^{r+1}}$, and we can call the homomorphic linear transformation $\mathsf{CoeffsToSlots}$ to move these coefficients to the slots modulo p^{r+1}. Call this ciphertext ct_0. Notice that in each slot, we want to next extract the largest p-digit in $z = p^r z\langle r\rangle + p^{r-1}(\mathsf{lowerdigits})$. Let $k = \lceil \log_2 p \rceil$. Now we use the oracle to extract

$$z\langle r\rangle = \sum_{i=0}^{K-1} 2^i z_i^r,$$

for $z_i^r \in \{0,1\}$, bit-by-bit.

Initialize $\mathsf{ct}_{\mathsf{sum}} \leftarrow (0,0)$. For $i = 1, \ldots, k-1$, let

$$\mathsf{ct}'_i \leftarrow \mathcal{O}_{\geq}(\mathsf{ct}_{i-1}, p^r \cdot 2^{k-i}; p^{r+1}, Q_{i-1}, Q_i, \chi)$$

and let $\mathsf{ct}_i = \mathsf{ModSwitch}(\mathsf{ct}_{i-1}, Q_i) - (p^r 2^{k-i}) \cdot \mathsf{ct}'_i$ and $\mathsf{ct}_{\mathsf{sum}} \leftarrow 2^{k-i} \cdot \mathsf{ct}'_i + \mathsf{ModSwitch}(\mathsf{ct}'_i, Q_i)$. Finally, $\mathsf{ct}_{\mathsf{sum}}$ is an encryption of $z\langle r\rangle$ modulo p so we compute the ciphertext

$$\mathsf{ModSwitch}(\mathsf{ct}_{k-1}, Q_{k-1}) - \mathsf{ct}_{\mathsf{sum}}$$

and then call the homomorphic linear transformation $\mathsf{SlotsToCoeffs}$ to move the slots modulo p to the coefficients and return the resulting ciphertext.

Correctness follows from Lemma 2.3. □

6 Conclusion

In this work, we provide strong evidence that homomorphic scheme-switching between the BGV/BFV and CKKS SIMD FHE schemes is as hard as bootstrapping. We achieve this by showing how to bootstrap both BGV/BFV and CKKS ciphertexts with a single call to such an algorithm (in fact, only a weak scheme-switching algorithm that does not convert the differences in packings between the schemes) without having to perform homomorphic linear transformations. In addition, we show how homomorphic comparisons are analogously powerful by bootstrapping with a few calls to a SIMD comparison algorithm. The fact that we can bootstrap with one call to a weak scheme-switching oracle is surprising since weak scheme-switching appears much simpler than bootstrapping.

References

[AdCY$^+$23] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Tugce Yazicigil, Anantha P. Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. FAB: an fpga-based accelerator for bootstrappable fully homomorphic encryption. In *HPCA*, pages 882–895. IEEE, 2023.

[AP13] Jacob Alperin-Sheriff and Chris Peikert. Practical bootstrapping in quasilinear time. In *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013.

[AS08] Frederik Armknecht and Ahmad-Reza Sadeghi. A new approach for algebraically homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 422, 2008.

[BBB$^+$22] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. `https://eprint.iacr.org/2022/915`.

[BCC$^+$22] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. Meta-bts: Bootstrapping precision beyond the limit. Cryptology ePrint Archive, Paper 2022/1167, 2022. To Appear in CCS 2022. `https://eprint.iacr.org/2022/1167`.

[BGGJ20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. CHIMERA: combining ring-lwe-based fully homomorphic encryption schemes. *J. Math. Cryptol.*, 14(1):316–338, 2020.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325. ACM, 2012.

[BMTPH21] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 587–617, Cham, 2021. Springer International Publishing.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In *EUROCRYPT (2)*, volume 11477 of *Lecture Notes in Computer Science*, pages 34–54. Springer, 2019.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, 2016.

[CH18] Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved FHE bootstrapping. In *EUROCRYPT (1)*, volume 10820 of *Lecture Notes in Computer Science*, pages 315–337. Springer, 2018.

[CHK$^+$18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT (1)*, volume 10820 of *Lecture Notes in Computer Science*, pages 360–384. Springer, 2018.

[CKK$^+$19] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun-Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II*, volume 11922 of *Lecture Notes in Computer Science*, pages 415–445. Springer, 2019.

[CKK20] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 221–256. Springer, 2020.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT (1)*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

[CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In *ASIACRYPT (3)*, volume 13092 of *Lecture Notes in Computer Science*, pages 670–699. Springer, 2021.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015.

[DMPS22] Nir Drucker, Guy Moshkowich, Tomer Pelleg, and Hayim Shaul. Bleach: Cleaning errors in discrete computations over ckks. Cryptology ePrint Archive, Paper 2022/1298, 2022. `https://eprint.iacr.org/2022/1298`.

[FK94] Michael Fellows and Neal Koblitz. Combinatorial cryptosystems galore! 1994.

[FSK$^+$21] Axel Feldmann, Nikola Samardzic, Aleksandar Krastev, Srini Devadas, Ronald G. Dreslinski, Karim Eldefrawy, Nicholas Genise, Chris Peikert, and Daniel Sánchez. F1: A fast and programmable accelerator for fully homomorphic encryption (extended version). *CoRR*, abs/2109.05371, 2021.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[GBP$^+$22] Robin Geelen, Michiel Van Beirendonck, Hilder V. L. Pereira, Brian Huffman, Tynan McAuley, Ben Selfridge, Daniel Wagner, Georgios Dimou, Ingrid Verbauwhede, Frederik Vercauteren, and David W. Archer. BASALISC: flexible asynchronous hardware accelerator for fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 657, 2022.

[Gen09a] Craig Gentry. A fully homomorphic encryption scheme. *Diss. Stanford University*, 2009.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.

[GHS12c] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

[GV22] Robin Geelen and Frederik Vercauteren. Bootstrapping for BGV and BFV revisited. *IACR Cryptol. ePrint Arch.*, page 1363, 2022.

[HHC19] K. Han, M. Hhan, and J. H. Cheon. Improved Homomorphic Discrete Fourier Transforms and FHE Bootstrapping. *IEEE Access*, 7:57361–57370, 2019.

[HK20] Kyoohyung Han and Dohyeong Ki. Better Bootstrapping for Approximate Homomorphic Encryption. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pages 364–390, Cham, 2020. Springer International Publishing.

[HPS19] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In *CT-RSA*, volume 11405 of *Lecture Notes in Computer Science*, pages 83–105. Springer, 2019.

[HS14a] Shai Halevi and Victor Shoup. Algorithms in helib. In *CRYPTO (1)*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2014.

[HS14b] Shai Halevi and Victor Shoup. Bootstrapping for helib. *IACR Cryptol. ePrint Arch.*, page 873, 2014.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for helib. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 641–670. Springer, 2015.

[HS18] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in helib. In *CRYPTO (1)*, volume 10991 of *Lecture Notes in Computer Science*, pages 93–120. Springer, 2018.

[HS20] Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. *IACR Cryptol. ePrint Arch.*, page 1481, 2020.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

[IZ21] Ilia Iliashenko and Vincent Zucca. Faster homomorphic comparison operations for BGV and BFV. *Proc. Priv. Enhancing Technol.*, 2021(3):246–264, 2021.

[JM20] Charanjit S. Jutla and Nathan Manohar. Modular Lagrange Interpolation of the Mod Function for Bootstrapping of Approximate HE. Cryptology ePrint Archive, Report 2020/1355, 2020. `https://eprint.iacr.org/2020/1355`.

[JM22] Charanjit S. Jutla and Nathan Manohar. Sine series approximation of the mod function for bootstrapping of approximate HE. In *EUROCRYPT (1)*, volume 13275 of *Lecture Notes in Computer Science*, pages 491–520. Springer, 2022.

[KDE$^+$21] Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for rlwe-based homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 691, 2021.

[KPK$^+$22] Seonghak Kim, Minji Park, Jaehyung Kim, Taekyung Kim, and Chohong Min. Evalround algorithm in CKKS bootstrapping. In *ASIACRYPT*, 2022.

[KPZ21] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting homomorphic encryption schemes for finite fields. In *ASIACRYPT (3)*, volume 13092 of *Lecture Notes in Computer Science*, pages 608–639. Springer, 2021.

[LHH$^+$21] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *IEEE Symposium on Security and Privacy*, pages 1057–1073. IEEE, 2021.

[LLK$^+$22] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 551–580, Cham, 2022. Springer International Publishing.

[LLKN21] Eunsang Lee, Joon-Woo Lee, Young-Sik Kim, and Jong-Seon No. Optimization of homomorphic comparison algorithm on rns-ckks scheme. Cryptology ePrint Archive, Paper 2021/1215, 2021. `https://eprint.iacr.org/2021/1215`.

[LLL$^+$21] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In *EUROCRYPT (1)*, volume 12696 of *Lecture Notes in Computer Science*, pages 618–647. Springer, 2021.

[LM21] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In *EUROCRYPT (1)*, volume 12696 of *Lecture Notes in Computer Science*, pages 648–677. Springer, 2021.

[LMP22] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In *ASIACRYPT*, 2022.

[LMSS22] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In *CRYPTO (1)*, volume 13507 of *Lecture Notes in Computer Science*, pages 560–589. Springer, 2022.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

[MGH10] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2010.

[RAD$^+$78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93. ACM, 2005.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.

[SFK$^+$21] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald G. Dreslinski, Christopher Peikert, and Daniel Sánchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO*, pages 238–252. ACM, 2021.

[SFK$^+$22] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sánchez. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *ISCA*, pages 173–187. ACM, 2022.

[SV14] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.*, 71(1):57–81, 2014.