Bootstrapping GBFV with CKKS

Jaehyung Kim

Stanford University, Stanford, USA
jaehk@cs.stanford.edu

Abstract. Generalized BFV (GBFV) replaces the integer plaintext modulus of BFV with a polynomial plaintext modulus, yielding a flexible tradeoff between precision and SIMD parallelism. This framework encompasses both BFV and CLPX, with CLPX corresponding to the extreme large-precision endpoint where very large plaintext moduli can be supported without increasing the underlying RLWE dimension. However, bootstrapping remains a bottleneck throughout the large-precision GBFV regime: existing GBFV bootstrapping methods rely on BFV-style linear transformations, whose noise growth becomes prohibitive as the plaintext precision grows. We introduce a GBFV bootstrapping method that treats the noise of a GBFV ciphertext as a CKKS plaintext and bootstraps it through CKKS. The main obstacle is that the BFV-from-CKKS bootstrapping technique of Kim et al. (CCS'24) relies on integer plaintext scaling and does not directly apply to GBFV, whose scaling factor involves a polynomial plaintext modulus. We overcome this by developing a noise-extraction and recovery procedure tailored to polynomial plaintext moduli. Our implementation bootstraps large-precision GBFV instances, including CLPX for the first time, supporting plaintext moduli of more than 500,000 bits in less than 20 seconds. We also introduce GBFV-to-CKKS and CKKS-to-GBFV conversions, interpreting the former as an approximate digit-decomposition toolkit for non-arithmetic operations over large GBFV plaintext spaces.

Keywords: GBFV · Bootstrapping · CLPX · CKKS.

1 Introduction

Fully homomorphic encryption (FHE) is a cryptographic primitive that supports computation over encrypted data. Among modern FHE schemes, BGV [12] and BFV [10, 28] are widely used for exact integer arithmetic, supporting single-instruction multiple-data (SIMD) operations over plaintext rings of the form $\mathcal{R}/p\mathcal{R}$, where $\mathcal{R} = \mathbb{Z}[X]/(\Phi_M(X))$ is a cyclotomic ring and $p > 0$ is an integer plaintext modulus. A fundamental limitation of this representation is its dependence on p: after ciphertext multiplication, the noise grows by a factor proportional to p, and supporting a large plaintext modulus requires correspondingly

© IACR 2026. This article is a minor revision of the version published by Springer available at [DOI to be added].

larger ciphertext parameters. This makes BGV/BFV inefficient for very-large-precision integer arithmetic.

A promising way to avoid this dependence is to replace the integer plaintext modulus by a polynomial plaintext modulus. Chen et al. [17] observed that quotienting by $X - b$ gives

$$\mathbb{Z}[X]/(\Phi_M(X), X - b) \cong \mathbb{Z}_{\Phi_M(b)},$$

which can represent a much larger integer space while the noise growth is governed by b, rather than by the full modulus $\Phi_M(b)$. This gives an asymptotic advantage over BGV/BFV in the large-precision regime. However, the CLPX construction sacrifices SIMD parallelism and, more importantly, no bootstrapping method was known for it. More recently, generalized BFV (GBFV) [30, 14] unified BFV and CLPX by quotienting by more general polynomials, thereby providing a tradeoff between precision, measured by the plaintext modulus size, and parallelism, measured by the number of slots.

This tradeoff is especially attractive for applications that evaluate large-integer cryptographic computations inside FHE. Examples include homomorphic signing for threshold and blind signatures [5, 7], delegated proof generation for zkSNARKs [29], and matrix triple generation for SPDZ [14]. In such applications, the plaintext modulus may need to be very large, while the number of SIMD slots is often less important than the ability to keep the underlying RLWE parameters small. GBFV is well suited to this setting: it can increase the available plaintext precision without increasing the FHE parameters in the same way as BGV/BFV, leading to substantially better latency.

A central remaining obstacle is bootstrapping. The first GBFV bootstrapping method [30] demonstrated that GBFV can outperform BGV/BFV for moderate plaintext moduli such as $p = 2^{16} + 1$; for example, it reports bootstrapping in about 2 seconds at ring dimension 2^{14}. However, the approach still relies on conversions between GBFV and BFV, and in particular on BFV linear transformations whose noise growth depends on the plaintext modulus. As a result, the method does not extend efficiently to the very-large-precision regime, including the CLPX endpoint.

This paper addresses the following question:

> *Can one bootstrap GBFV efficiently across the entire precision-parallelism tradeoff, including the CLPX regime?*

Our answer is based on CKKS bootstrapping [22, 19]. We build on the philosophy of the BFV bootstrapping from CKKS [38]: the BFV noise can be viewed as a CKKS plaintext, bootstrapped through CKKS, and subtracted from the original ciphertext. At a high level, we follow the same noise-refreshing paradigm for GBFV. The main difficulty is that the BFV construction relies on integer plaintext scaling, whereas GBFV uses a polynomial plaintext modulus. A GBFV ciphertext encrypting m decrypts to a value of the form $\lfloor (q/t) \cdot m \rceil + e$, where t is a polynomial. Thus the direct BFV-style noise extraction is not well-defined: the scaling factor q/t is not an integer modulus.

We overcome this obstacle by developing a noise-extraction and recovery procedure tailored to polynomial plaintext moduli. This yields a generic GBFV bootstrapping method whose denoising factor is governed by the precision of the underlying CKKS bootstrapping, rather than by the size of the plaintext modulus. As a result, the method applies to the large-precision GBFV regime, including the CLPX setting. In particular, while prior GBFV bootstrapping remains preferable for small plaintext moduli, its BFV-linear-transformation cost grows with the plaintext precision. As discussed in Section 3.2, this makes our method advantageous for plaintext moduli of a few hundred bits, which is the relevant range for many cryptographic applications.

1.1 Technical Overview

Let $N > 1$ be a power-of-two integer, and let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. Given a polynomial $t \in \mathcal{R}$, the plaintext space of GBFV is

$$\mathcal{R}_t = \mathcal{R}/t\mathcal{R} = \mathbb{Z}[X]/(X^N + 1, t(X)).$$

Given a plaintext $m \in \mathcal{R}_t$, a GBFV ciphertext encrypting m can be written as

$$\mathsf{ct} = (-as + \lfloor (q/t) \cdot m \rceil + e, a) \in \mathcal{R}_q^2,$$

where $\mathsf{sk} = (1, s) \in \mathcal{R}^2$ is the secret key. Equivalently,

$$\mathsf{ct} \cdot \mathsf{sk} = \left\lfloor \frac{q}{t} \cdot m \right\rceil + e \pmod{q}. \tag{1}$$

Thus, m is encoded in the most significant part through the polynomial scaling factor q/t, while e is the noise. The goal of GBFV bootstrapping is to reduce this noise without changing the underlying message.

Definition 1 (Informal). *Given an RLWE ciphertext $\mathsf{ct} \in \mathcal{R}_q^2$ that decrypts to $\lfloor (q/t) \cdot m \rceil + e \in \mathcal{R}$, GBFV bootstrapping is a homomorphic operation that outputs an RLWE ciphertext $\mathsf{ct}' \in \mathcal{R}_q^2$ that decrypts to $\lfloor (q/t) \cdot m \rceil + e' \in \mathcal{R}$, where e' is much smaller than e.*

Our starting point is the BFV bootstrapping from CKKS [38], which extracts the BFV noise, CKKS-bootstraps the extracted noise, and subtracts it from the original ciphertext. We first explain why this strategy does not directly extend to polynomial plaintext moduli, and then describe our replacement: scaled-noise extraction followed by approximate division by the plaintext polynomial.

The BFV-from-CKKS template. When t is an integer, GBFV specializes to BFV. Suppose that $t \mid q$ and let $\Delta = q/t$. Then Equation (1) becomes

$$\mathsf{ct} \cdot \mathsf{sk} = \Delta \cdot m + e \pmod{q}.$$

The BFV-from-CKKS method extracts the noise by reducing the ciphertext modulo Δ:

$$[\mathsf{ct}]_\Delta \cdot \mathsf{sk} = e \pmod{\Delta}.$$

CKKS bootstrapping then approximately preserves this extracted noise while raising the modulus back to q. This gives a ciphertext ct' satisfying

$$\mathsf{ct}' \cdot \mathsf{sk} \approx e \pmod{q}.$$

Finally, subtracting ct' from the original ciphertext removes the old noise:

$$(\mathsf{ct} - \mathsf{ct}') \cdot \mathsf{sk} \approx \Delta \cdot m \pmod{q}.$$

Thus, in the integer-modulus case, their BFV bootstrapping can be understood as noise extraction, noise bootstrapping, and subtraction.

Why the template fails for GBFV. For a general polynomial plaintext modulus t, the first step of the above template is no longer available. The scaling factor

$$\Delta = \frac{q}{t}$$

is not an integer modulus. Hence neither reduction modulo q/t nor reduction modulo $\lfloor q/t \rceil$ defines a compatible modular operation over $\mathcal{R}_q$. Therefore, the BFV-style noise extraction step has no direct analogue for GBFV.

Scaled-noise extraction. Instead of reducing modulo q/t, we multiply the ciphertext by t. From Equation (1), we obtain

$$(t \cdot \mathsf{ct}) \cdot \mathsf{sk} \approx t \cdot e \pmod{q}.$$

Thus, multiplication by t removes the message modulo q. It does not extract the noise e itself, but it extracts a scaled version of the noise, namely $t \cdot e$.

This changes the problem. Instead of extracting e directly, we first extract $t \cdot e$ and then recover e by approximately dividing by the plaintext polynomial t.

Approximate division by a plaintext polynomial. We construct the approximate division procedure using CKKS bootstrapping and CKKS rescaling. Let $\mathsf{ct}_{\mathrm{in}} \in \mathcal{R}_q^2$ be an RLWE ciphertext whose underlying plaintext is u:

$$\mathsf{ct}_{\mathrm{in}} \cdot \mathsf{sk} = u \pmod{q}.$$

CKKS bootstrapping can be viewed as an approximate homomorphic modulus-raising operation. Given an integer $q' \approx q$, it outputs a ciphertext at modulus qq' whose underlying plaintext is still approximately u:

$$\mathsf{CKKS.Bootstrap}(\mathsf{ct}_{\mathrm{in}}) \approx u \pmod{qq'}.$$

The larger modulus gives room to divide by t. We multiply by $\lfloor q'/t \rceil$ and then rescale by q':

$$\mathsf{Divide}_t(\mathsf{ct}_{\mathrm{in}}) = \mathsf{Rescale}_{q'}\left(\left\lfloor \frac{q'}{t} \right\rceil \cdot \mathsf{CKKS.Bootstrap}(\mathsf{ct}_{\mathrm{in}})\right).$$

Here $\lfloor q'/t \rceil$ is an integral approximation to $q't^{-1}$ in the ambient rational algebra. Therefore, multiplying by $\lfloor q'/t \rceil$ and then rescaling by q' approximately divides the underlying plaintext by t:

$$\mathsf{Divide}_t(\mathsf{ct}_{\mathrm{in}}) \cdot \mathsf{sk} \approx \frac{u}{t} \pmod{q}.$$

In particular, when $\mathsf{ct}_{\mathrm{in}} = t \cdot \mathsf{ct}$, the underlying plaintext is approximately $t \cdot e$, so

$$\mathsf{Divide}_t(t \cdot \mathsf{ct}) \cdot \mathsf{sk} \approx e \pmod{q}.$$

This approximate division step is the main new ingredient: it realizes homomorphic division by a plaintext polynomial using multiplication by an integral approximation to $q't^{-1}$ followed by ordinary integer rescaling.

GBFV bootstrapping. We now combine scaled-noise extraction, approximate division, and subtraction. Start with a GBFV ciphertext ct satisfying

$$\mathsf{ct} \cdot \mathsf{sk} = \left\lfloor \frac{q}{t} \cdot m \right\rceil + e \pmod{q}.$$

The bootstrapping procedure consists of the following three steps.

1. **Scaled-noise extraction.** Multiply by t:

$$(t \cdot \mathsf{ct}) \cdot \mathsf{sk} \approx t \cdot e \pmod{q}.$$

2. **Approximate division.** Apply the division procedure:

$$\mathsf{Divide}_t(t \cdot \mathsf{ct}) \cdot \mathsf{sk} \approx e \pmod{q}.$$

3. **Subtraction.** Subtract the recovered-noise ciphertext from the original ciphertext:

$$\mathsf{ct}_{\mathrm{out}} = \mathsf{ct} - \mathsf{Divide}_t(t \cdot \mathsf{ct}).$$

   Then

$$\mathsf{ct}_{\mathrm{out}} \cdot \mathsf{sk} \approx \left\lfloor \frac{q}{t} \cdot m \right\rceil \pmod{q}.$$

The output ciphertext therefore encrypts the same GBFV plaintext as the input, but with the original noise replaced by the smaller error introduced during the approximate division procedure.

GBFV–CKKS conversions. As a further consequence, we introduce conversions between GBFV and CKKS. These conversions are not needed for the bootstrapping algorithm above, but they follow from a similar way of separating the message and noise.

Recall that a GBFV ciphertext $\mathsf{ct} \in \mathcal{R}_q^2$ decrypts to

$$\mathsf{ct} \cdot \mathsf{sk} = \left\lfloor \frac{q}{t} \cdot m \right\rceil + e \pmod{q}$$

for a message $m \in \mathcal{R}_t$ and a small noise e. In the bootstrapping algorithm, multiplying by t removes the message modulo q:

$$t \cdot \left\lfloor \frac{q}{t} \cdot m \right\rceil \approx q \cdot m \equiv 0 \pmod{q}.$$

For conversion, we instead raise the modulus before multiplying by t, so that the term $\lfloor (q/t) \cdot m \rceil$ is not lost modulo q. A modulus-raised ciphertext $\widetilde{\mathsf{ct}}$ decrypts to

$$\widetilde{\mathsf{ct}} \cdot \mathsf{sk} = \left\lfloor \frac{q}{t} \cdot m \right\rceil + e + q \cdot I$$

for some small integral polynomial $I \in \mathcal{R}$. Using the modular-reduction viewpoint of CKKS bootstrapping, bootstrapping $\widetilde{\mathsf{ct}}$ gives a ciphertext ct' decrypting to

$$\left\lfloor \frac{q}{t} \cdot m \right\rceil + e + q \cdot I' + e_{\mathrm{BTS}}$$

for some bounded integral polynomial $I' \in \mathcal{R}$ and bootstrapping error e_{BTS}. Multiplying by t then gives a ciphertext decrypting to

$$q \cdot (m + t \cdot I') + e'$$

for a small error $e' = t \cdot (e + e_{\mathrm{BTS}}) + e_{\mathrm{Round}}$, where e_{Round} is a rounding error.

Depending on how the plaintext space $\mathcal{R}_t$ is represented inside $\mathcal{R}$, such a ciphertext gives additional encrypted information about the underlying plaintext. A common choice is $t(X) = X^k - b$. In this case, the polynomial representation of m corresponds to a base-b digit decomposition of the coefficients in

$$\mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b).$$

Hence $m + t \cdot I'$ gives a possibly redundant digit decomposition with bounded coefficients.

Theorem 1 (Informal). *Let* ct *be a GBFV ciphertext encrypting a message* $m \in \mathcal{R}_t$*, identified with a polynomial*

$$\sum_{i=0}^{k-1} m_i \cdot X^i \in \mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b).$$

The GBFV-to-CKKS conversion gives a CKKS ciphertext encrypting vectors $(\boldsymbol{\mu}_0, \ldots, \boldsymbol{\mu}_{k-1})$ *such that*

$$m_i = \sum_{j=0}^{N/k-1} \mu_{ij} \cdot b^j,$$

where $\boldsymbol{\mu}_i = (\mu_{i0}, \ldots, \mu_{i(N/k-1)})$ *for each* $0 \le i < k$*. Moreover, the digits satisfy* $|\mu_{ij}| < O(b)$*.*

In other words, the GBFV-to-CKKS conversion provides an approximate digit decomposition. This gives a toolkit for logical and other non-arithmetic operations over GBFV plaintexts, as discussed later in the paper.

1.2 Contributions

The main contribution of this paper is a CKKS-based bootstrapping method for large-precision GBFV. Our construction avoids the BFV-linear-transformation bottleneck of prior GBFV bootstrapping methods and, as a result, remains practical in regimes where the plaintext modulus is very large. We also develop GBFV-to-CKKS and CKKS-to-GBFV conversions, which provide a toolkit for non-arithmetic operations over GBFV plaintext spaces.

Large-Precision Bootstrapping. The GBFV bootstrapping method of [30] demonstrated strong performance for moderate plaintext moduli, such as $p = 2^{16} + 1$. However, its construction relies on subprocedures from existing BFV bootstrapping methods, including BFV linear transformations. As the plaintext modulus grows, the modulus consumption of these transformations increases with the plaintext precision and can become too large to be supported within practical RLWE dimensions, such as $\log N \leq 17$.

Our bootstrapping method avoids this bottleneck by extracting the GBFV noise and refreshing it through CKKS bootstrapping. Once the target CKKS bootstrapping precision is fixed, its complexity is almost independent of the underlying plaintext modulus, allowing it to support large-precision GBFV instances beyond the reach of BFV-based approaches. In particular, we bootstrap the CLPX setting [17], corresponding to $t(X) = X - b$, for the first time. Our implementation supports plaintext moduli of about 500,000 bits in less than 20 seconds.

Although this CLPX experiment demonstrates scalability at the extreme large-precision endpoint, the practical relevance of our approach already appears at much smaller plaintext sizes. In the few-hundred-bit plaintext-modulus regime, the BFV-linear-transformation cost in prior GBFV bootstrapping can consume a substantial fraction of the available ciphertext-modulus budget, whereas the cost of our CKKS-based refresh remains largely unchanged. This suggests that our construction is expected to be practically efficient for the few-hundred-bit plaintext moduli that arise in large-integer cryptographic computations, while also extending naturally to the CLPX endpoint.

A GBFV–CKKS Conversion Toolkit. As a second contribution, we develop conversions between GBFV and CKKS. The GBFV-to-CKKS direction can be viewed as an approximate base-b digit decomposition of a GBFV plaintext, while the CKKS-to-GBFV direction corresponds to recombination. These conversions extend the role of CKKS beyond noise refreshing, providing a way to use CKKS for operations that are less natural to perform directly over large GBFV plaintext spaces.

We demonstrate this toolkit through approximate modular reduction. Using the digit-decomposed CKKS representation, we can homomorphically reduce a GBFV plaintext modulo an arbitrary integer modulus, while the native GBFV plaintext space may be defined modulo a possibly smooth integer $b^{N/k} + 1$. This

is analogous in spirit to the modular-reduction technique of [6], but is realized through the GBFV–CKKS conversion interface. Our implementation shows that this conversion-based modular reduction is practical for cryptographic-size moduli. As a proxy benchmark for arbitrary-modulus integer multiplication, we compare our measured reduction latency with the multiplication timings reported for the radix-based approach of [13] at the same precision. This suggests that GBFV–CKKS conversion can serve as a practical tool for supporting arithmetic modulo arbitrary target moduli inside large-precision GBFV plaintext spaces, and may also provide a route to rescaling and fixed-point arithmetic in this setting.

We also discuss arbitrary function evaluation via multivariate interpolation over the digit-decomposed CKKS representation in Appendix B.

1.3 Additional Related Work

Fully homomorphic encryption for high-precision computation has been explored extensively in the literature. In CGGI/DM [23, 27], high precision is often achieved through a radix-style representation [24] or through the Chinese Remainder Theorem [15, 16]. Since CGGI/DM relies on a small-precision integer computer to handle large precision, it can be difficult to support large precision efficiently. An alternative approach is to use discrete variants of CKKS [26, 4, 2, 37]. In [34], the authors take a radix-based approach, efficiently supporting arithmetic over moderate-size integers such as 32- or 64-bit integers. In [6], the authors construct a nested CRT system inside RLWE to simulate CRT-based large-integer arithmetic. One can also use high-precision approximate CKKS bootstrapping, which bootstraps real numbers rather than integers. In this work, we mainly use the high-precision CKKS bootstrapping of META-BTS [3] to analyze the asymptotic efficiency of our method.

The concept of converting between FHE schemes for better efficiency is well studied [9, 39]. In contrast, little is known about conversions to and from GBFV. The only prior conversion we are aware of is the conversion between GBFV and BFV in [30]. Their conversion is approximate, as in our case, and is mainly used to construct their GBFV bootstrapping.

Concurrent and Subsequent Work. In [31], the authors use subring BFV to minimize the computation performed on the BFV side. In particular, the entire Slots-to-Coefficients transform is computed on the GBFV side, significantly reducing the noise growth. This improves the efficiency of the previous bootstrapping method in [30], but it still does not handle very large precision, since some components remain on the BFV side. In [11], the authors discuss using a non-cyclotomic ring that directly supports a power-of-two modulus. However, their bootstrapping is based on the generic RLWE bootstrapping of [33], and has significantly worse performance than ours. Subsequent work [40] introduces decomposed BFV, which uses relaxed digit decompositions to support arbitrary integer plaintext moduli with error rates and noise growth comparable to those

of BFV at much smaller plaintext moduli. This direction is orthogonal to ours, and the paper does not report a bootstrapping implementation.

Two recent works [35, 13] improve the radix-based approach of [34] and achieve large-precision bootstrapping using a logarithmic number of bootstrappings compared to the previous work. A significant advantage of GBFV over this approach is that radix-based methods need bootstrapping after almost every multiplication, whereas GBFV is much more tolerant of noise growth. A recent high-precision CKKS bootstrapping method [25] significantly improves efficiency over META-BTS [3]. Since our method uses high-precision CKKS bootstrapping as a black box, such improvements directly translate to better efficiency for GBFV bootstrapping.

A very recent work [36] further develops the conversion viewpoint by using Discrete CKKS as a backend for logical operations in BFV and GBFV, including exact digit decomposition and faster comparison. This direction is complementary to ours, which focuses on CKKS-based GBFV bootstrapping and the GBFV–CKKS conversion interface.

2 Preliminaries

Let $N > 1$ be a power-of-two integer, and let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. Given $t \in \mathcal{R}$, let $\mathcal{R}_t = \mathcal{R}/t\mathcal{R}$. The infinity norm $\|\cdot\|_\infty$ on $\mathcal{R}$ denotes the maximum absolute value among its coefficients. We use the notation $\equiv_q$ (resp. $\equiv_t$) to denote equivalence modulo q (resp. modulo t). Unless stated otherwise, $[\cdot]_q : \mathbb{Z} \to \mathbb{Z}_q = [-q/2, q/2) \cap \mathbb{Z}$ denotes signed modular reduction modulo q. Given a polynomial $\alpha(X) \in \mathcal{R}$, we write $\alpha(X) = \sum_{i=0}^{N-1} \alpha_i \cdot X^i$ for its canonical representative. This gives a natural embedding $\mathcal{R} \to \mathbb{Z}[X]$, and we denote by $\deg(\alpha)$ the degree of this embedded polynomial.

We also use division by a polynomial plaintext modulus in the rational extension $\tilde{\mathcal{R}} = \mathbb{Q}[X]/(X^N + 1)$. Assume that t is invertible in $\tilde{\mathcal{R}}$. For $m \in \mathcal{R}$, we write $m/t := m \cdot t^{-1} \in \tilde{\mathcal{R}}$, where t^{-1} is taken in $\tilde{\mathcal{R}}$. For $\beta \in \tilde{\mathcal{R}}$, write its canonical representative as $\beta = \sum_{i=0}^{N-1} \beta_i \cdot X^i$ with $\beta_i \in \mathbb{Q}$. We define $\lfloor \beta \rceil := \sum_{i=0}^{N-1} \lfloor \beta_i \rceil \cdot X^i \in \mathcal{R}$ by rounding each coefficient to the nearest integer, and we write $[\beta]_1 := \beta - \lfloor \beta \rceil \in \tilde{\mathcal{R}}$ for the corresponding coefficient-wise signed fractional part.

2.1 GBFV Basics

Let $1 \le k \le N$ be a power-of-two integer and let $b \in \mathbb{Z} \setminus \{0, \pm 1\}$. Let $t(X) = X^k - b \in \mathcal{R}$, and let $\mathcal{R}_t$ be the plaintext space for GBFV.[1] We rephrase the definitions of GBFV from [30].

Given a message $m \in \mathcal{R}_t$ and a secret key $s \in \mathcal{R}$, the GBFV encryption of m under s is defined as

$$\mathsf{GBFV.Encrypt}(s, m) = (-a \cdot s + \lfloor \Delta \cdot m \rceil + e, a) \in \mathcal{R}_q^2,$$

[1] Our idea naturally generalizes to general $t \in \mathcal{R}$. We keep $t(X) = X^k - b$ for simplicity.

where $\Delta = q/t$ is the scaling factor, $a \leftarrow \mathcal{R}_q$, and $e \leftarrow \chi_{\text{noise}}$.[2] Conversely, given a ciphertext $\mathsf{ct} \in \mathcal{R}_q^2$, the GBFV decryption of ct under s is defined as

$$\mathsf{GBFV.Decrypt}(s, \mathsf{ct} = (c_0, c_1)) = \lfloor (c_0 + c_1 \cdot s)/\Delta \rceil \in \mathcal{R}_t.$$

GBFV supports homomorphic addition and multiplication over $\mathcal{R}_t$, defined as follows.

- $\mathsf{GBFV.Add}(\mathsf{ct}, \mathsf{ct}') \to \mathsf{ct}_{\text{Add}}$: on input two GBFV ciphertexts encrypting $m, m' \in \mathcal{R}_t$, respectively, the homomorphic addition outputs a GBFV ciphertext encrypting $m + m' \in \mathcal{R}_t$.
- $\mathsf{GBFV.Mult}(\mathsf{ct}, \mathsf{ct}') \to \mathsf{ct}_{\text{Mult}}$: on input two GBFV ciphertexts encrypting $m, m' \in \mathcal{R}_t$, respectively, the homomorphic multiplication outputs a GBFV ciphertext encrypting $m \cdot m' \in \mathcal{R}_t$.

This naturally gives homomorphic encryption over $\mathcal{R}_t \cong \mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b)$. The plaintext space $\mathcal{R}_t$ often splits and provides SIMD arithmetic over $\mathbb{Z}_{b^{N/k}+1}$.

Let $\mathsf{ct} = (c_0, c_1)$ be a GBFV ciphertext encrypting a message $m \in \mathcal{R}_t$. This means that

$$(c_0, c_1) \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e$$

for some small noise $e \in \mathcal{R}$. The **GBFV bootstrapping** procedure significantly reduces the size of the noise while preserving the underlying message m. That is,

$$\mathsf{GBFV.Bootstrap}(\mathsf{ct}) \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e',$$

where $e' \ll e$. To quantify this, we assume that there exist $B_1, B_2 > 0$ such that, if the input noise e satisfies $\|e\|_\infty \leq B_1$, then the output noise e' satisfies $\|e'\|_\infty \leq B_2 < B_1$.

2.2 CKKS Basics

Given a message $m \in \mathbb{R}[X]/(X^N + 1)$ and a secret key $s \in \mathcal{R}$, the coefficient-encoded CKKS encryption of m under s is defined as

$$\mathsf{CKKS.Encrypt}(s, m) = (-a \cdot s + \lfloor \Delta' \cdot m \rceil + e, a) \in \mathcal{R}_q^2,$$

where $\Delta' \in \mathbb{R}_{>0}$ is the scaling factor, $a \leftarrow \mathcal{R}_q$, and $e \leftarrow \chi'_{\text{err}}$.[3] Conversely, given a ciphertext $\mathsf{ct} \in \mathcal{R}_q^2$, the CKKS decryption of ct under s is defined as

$$\mathsf{CKKS.Decrypt}(s, \mathsf{ct} = (c_0, c_1)) = [c_0 + c_1 \cdot s]_q / \Delta' \in \mathbb{R}[X]/(X^N + 1).$$

The decryption of $\mathsf{ct} \in \mathcal{R}_q^2$ without scaling down,

$$[\mathsf{ct} \cdot (1, s)]_q = \lfloor \Delta' \cdot m \rceil + e \in \mathcal{R},$$

is called the underlying plaintext of ct.

[2] χ_{noise} is a noise distribution for GBFV.
[3] χ'_{err} is an error distribution for CKKS.

Given a CKKS ciphertext $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_Q^2$, the rescaling of ct by $q \mid Q$ is defined as

$$\mathsf{CKKS.RS}_q(\mathsf{ct}) = \left(\frac{c_0 - [c_0]_q}{q}, \frac{c_1 - [c_1]_q}{q} \right) \in \mathcal{R}_{Q/q}^2.$$

Rescaling by q approximately divides the underlying plaintext by q.

Let $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ be a CKKS ciphertext encrypting an erroneous message $m \in \mathbb{R}[X]/(X^N + 1)$. This means that

$$[(c_0, c_1) \cdot (1, s)]_q = \Delta' \cdot m.$$

The **CKKS bootstrapping** procedure [19] significantly increases the modulus while approximately preserving the message. That is, the resulting ciphertext lies in $\mathcal{R}_Q^2$ for $Q \gg q$ and satisfies

$$[\mathsf{CKKS.Bootstrap}(\mathsf{ct}) \cdot (1, s)]_Q = \Delta' \cdot m + e$$

for some small $e \in \mathcal{R}$. We often assume that $\|m\|_\infty \leq 1$ (so that $\|\Delta' \cdot m\|_\infty \leq \Delta' \ll q$) and $\|e/\Delta'\|_\infty \leq 2^{-p}$, and we refer to this as p-bit bootstrapping precision.

An important remark is that the target modulus Q can be chosen almost arbitrarily in size. For a fixed bootstrapping precision, the number of multiplicative levels consumed during bootstrapping is almost fixed, and choosing an appropriate modulus-raising modulus[4] leads to the desired modulus Q after bootstrapping. One caveat is that RLWE security depends on the modulus-raising modulus, but this can be addressed by increasing the RLWE dimension.

3 Our GBFV Bootstrapping

In this section, we present our CKKS-based GBFV bootstrapping procedure. The technical overview described the main difficulty: multiplying a GBFV ciphertext by the plaintext polynomial t removes the message modulo q, but it extracts the scaled noise $t \cdot e$ rather than the noise e itself. Therefore, the central operation needed for bootstrapping is an approximate homomorphic division by t.

We first define this approximate division procedure as a separate primitive. Given a ciphertext whose underlying plaintext is u modulo q, we CKKS-bootstrap it to a larger modulus qq', multiply by an integral approximation to q'/t, and then rescale by q'. Informally, this computes

$$u \longmapsto \frac{u}{t}$$

homomorphically. We then apply this primitive to the ciphertext $t \cdot \mathsf{ct}$, whose underlying plaintext is approximately $t \cdot e$, and subtract the resulting ciphertext from the original GBFV ciphertext.

[4] Modulus raising is an important step of CKKS bootstrapping. It interprets a CKKS ciphertext at a small modulus q' as a CKKS ciphertext at a larger modulus $Q' \gg q'$.

3.1 Main Algorithm

We start by defining the approximate division procedure. Let $\mathsf{ct}_{\text{in}} \in \mathcal{R}_q^2$ be an RLWE ciphertext whose underlying plaintext is $u \in \mathcal{R}$, namely

$$\mathsf{ct}_{\text{in}} \cdot (1, s) \equiv_q u.$$

Let $q' \approx q$, and let $\mathsf{CKKS.Bootstrap} : \mathcal{R}_q^2 \to \mathcal{R}_{qq'}^2$ be a CKKS bootstrapping procedure that approximately preserves the underlying plaintext while raising the modulus from q to qq'. We define division by t as follows:

$$\mathsf{Divide}_t(\mathsf{ct}_{\text{in}}) = \mathsf{CKKS.RS}_{q'}\left(\left\lfloor \frac{q'}{t} \right\rceil \cdot \mathsf{CKKS.Bootstrap}(\mathsf{ct}_{\text{in}})\right) \in \mathcal{R}_q^2.$$

The multiplication by $\lfloor q'/t \rceil$ approximates multiplication by $q't^{-1}$, and the subsequent rescaling by q' removes the integer scale. Hence, when the approximation errors are small, the output ciphertext has underlying plaintext close to u/t modulo q. The procedure is summarized in Algorithm 1.

Algorithm 1: Approximate Division by t

Setting: $q' \approx q$. $\mathsf{CKKS.Bootstrap} : \mathcal{R}_q^2 \to \mathcal{R}_{qq'}^2$ approximately preserves the underlying plaintext.
Input : $\mathsf{ct}_{\text{in}} \in \mathcal{R}_q^2$ such that $\mathsf{ct}_{\text{in}} \cdot (1, s) \equiv_q u$.
Output: $\mathsf{ct}_{\text{out}} \in \mathcal{R}_q^2$ such that $\mathsf{ct}_{\text{out}} \cdot (1, s) \approx u/t \pmod q$.

$\mathsf{ct}_{\text{BTS}} \leftarrow \mathsf{CKKS.Bootstrap}(\mathsf{ct}_{\text{in}}) \in \mathcal{R}_{qq'}^2$;
$\mathsf{ct}_{\text{out}} \leftarrow \mathsf{CKKS.RS}_{q'}(\lfloor q'/t \rceil \cdot \mathsf{ct}_{\text{BTS}}) \in \mathcal{R}_q^2$;
return ct_{out}

We now apply this division procedure to bootstrap GBFV. Let $\mathsf{ct} \in \mathcal{R}_q^2$ be a GBFV ciphertext encrypting a message $m \in \mathcal{R}_t$. In other words,

$$\mathsf{ct} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e,$$

where $\Delta = q/t$ and $e \in \mathcal{R}$ is the underlying noise. Multiplying by t removes the message modulo q and gives

$$[(t \cdot \mathsf{ct}) \cdot (1, s)]_q = t \cdot e + e_{\text{Round}}$$

for some small rounding error e_{Round}. We then apply Algorithm 1 to $t \cdot \mathsf{ct}$. Since the underlying plaintext of $t \cdot \mathsf{ct}$ is approximately $t \cdot e$, the output of the division procedure is a ciphertext whose underlying plaintext is approximately e. Subtracting this recovered-noise ciphertext from the original ciphertext reduces the noise while preserving the message. The full bootstrapping procedure is described in Algorithm 2.

Algorithm 2: GBFV Bootstrapping from CKKS

Setting: $\Delta = q/t$, $q' \approx q$. Divide_t is defined as in Algorithm 1.
Input : $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ such that $\mathsf{ct} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e$.
Output: $\mathsf{ct}_{\text{out}} = (c_0', c_1') \in \mathcal{R}_q^2$ such that $\mathsf{ct}_{\text{out}} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e'$, s.t. $e' \ll e$.

$\mathsf{ct}_{\text{ext}} \leftarrow t \cdot \mathsf{ct} \in \mathcal{R}_q^2$;
$\mathsf{ct}_{\text{noise}} \leftarrow \mathsf{Divide}_t(\mathsf{ct}_{\text{ext}}) \in \mathcal{R}_q^2$;
$\mathsf{ct}_{\text{out}} \leftarrow \mathsf{ct} - \mathsf{ct}_{\text{noise}} \in \mathcal{R}_q^2$;
return ct_{out}

Theorem 2 (Bootstrapping Correctness). *Let* $\mathsf{CKKS.Bootstrap} : \mathcal{R}_q^2 \to \mathcal{R}_{qq'}^2$ *be a CKKS bootstrapping procedure that takes as input a CKKS ciphertext encrypting a plaintext in* $[-B_1, B_1]^N \subseteq \mathcal{R}$ *and introduces an error in* $[-B_2, B_2]^N \subseteq \mathcal{R}$.[5] *Then Algorithm 2 gives a GBFV bootstrapping procedure whose input noise lies in* $[-B_1/(|b|+1)+1/2, B_1/(|b|+1)-1/2]^N$ *and whose output noise* e' *satisfies*

$$\|e'\|_\infty \le B_2 + \frac{|b| + h + 3}{2} + \frac{N}{2q'} \cdot (B_1 + B_2).$$

Assuming $q' \gg N$*, this reduces the noise by* $\log_2(B_1/B_2) - O_q(1)$ *bits.*[6] *In other words, a p-bit precision CKKS bootstrapping gives a GBFV bootstrapping procedure whose denoising factor is* $p - O(1)$ *bits.*

Proof. Let $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ be a GBFV ciphertext such that

$$\mathsf{ct} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e$$

for a message $m \in \mathcal{R}_t$ and a noise $e \in \mathcal{R}$ satisfying $\|e\|_\infty \le B_1/(|b|+1) - 1/2$. Since $t(X) = X^k - b$, we have

$$\begin{aligned} \|t \cdot e\|_\infty &= \|(X^k - b) \cdot e(X)\|_\infty \\ &\le \|X^k \cdot e(X)\|_\infty + \|b \cdot e(X)\|_\infty \\ &= (|b| + 1) \cdot \|e\|_\infty \le B_1 - \frac{|b| + 1}{2}. \end{aligned}$$

The rounding error e_{Round} is defined as

$$e_{\text{Round}} = t \cdot (\lfloor \Delta \cdot m \rceil - \Delta \cdot m)$$

and satisfies

$$\|e_{\text{Round}}\|_\infty \le (|b| + 1) \cdot \|\lfloor \Delta \cdot m \rceil - \Delta \cdot m\|_\infty \le \frac{|b| + 1}{2}.$$

[5] Here we consider the natural embedding $[-B_i, B_i]^N \subseteq \mathbb{R}^N \xrightarrow{\cong} \mathcal{R}$: an N-dimensional vector is identified with the coefficients of a degree-$(N-1)$ polynomial.
[6] Here h is the secret key Hamming weight.

Thus, by the triangle inequality, we have

$$\|t \cdot e + e_{\mathrm{Round}}\|_\infty \leq B_1,$$

so this is a valid input to $\mathsf{CKKS.Bootstrap}$. Therefore, the bootstrapping error e_{BTS} satisfies $\|e_{\mathrm{BTS}}\|_\infty \leq B_2$. Let $\mathsf{ct}_{\mathrm{ext}} = t \cdot \mathsf{ct}$, $\mathsf{ct}_{\mathrm{BTS}} = \mathsf{CKKS.Bootstrap}(\mathsf{ct}_{\mathrm{ext}})$, and $\mathsf{ct}_{\mathrm{noise}} = \mathsf{CKKS.RS}_{q'}(\lfloor q'/t \rceil \cdot \mathsf{ct}_{\mathrm{BTS}}) = \mathsf{Divide}_t(\mathsf{ct}_{\mathrm{ext}})$. In addition, we have

$$\begin{aligned}
[\mathsf{ct}_{\mathrm{noise}} \cdot (1, s)]_q &= \frac{(t \cdot e + e_{\mathrm{Round}} + e_{\mathrm{BTS}}) \cdot \lfloor q'/t \rceil}{q'} + e_{\mathrm{RS}} \\
&= \frac{(t \cdot e + e_{\mathrm{Round}} + e_{\mathrm{BTS}}) \cdot (q'/t - (q'/t - \lfloor q'/t \rceil))}{q'} + e_{\mathrm{RS}} \\
&= e + (e_{\mathrm{Round}} + e_{\mathrm{BTS}})/t \\
&\quad - \frac{(t \cdot e + e_{\mathrm{Round}} + e_{\mathrm{BTS}}) \cdot (q'/t - \lfloor q'/t \rceil)}{q'} + e_{\mathrm{RS}} \\
&= e - e'.
\end{aligned}$$

Here e_{RS} is the rescaling error, which satisfies $\|e_{\mathrm{RS}}\|_\infty \leq \frac{h+2}{2}$, where h is the secret key Hamming weight.[7] The final noise e' satisfies

$$\begin{aligned}
\|e'\|_\infty &\leq \|(e_{\mathrm{Round}} + e_{\mathrm{BTS}})/t\|_\infty + \frac{N}{2q'} \cdot \|t \cdot e + e_{\mathrm{Round}} + e_{\mathrm{BTS}}\|_\infty + \|e_{\mathrm{RS}}\|_\infty \\
&\leq \|e_{\mathrm{Round}}\|_\infty + \|e_{\mathrm{BTS}}\|_\infty \\
&\quad + \frac{N}{2q'} \cdot (\|t \cdot e + e_{\mathrm{Round}}\|_\infty + \|e_{\mathrm{BTS}}\|_\infty) + \|e_{\mathrm{RS}}\|_\infty \\
&\leq \frac{|b| + h + 3}{2} + B_2 + \frac{N}{2q'} \cdot (B_1 + B_2).
\end{aligned}$$

In the second inequality, we use the fact that $|b| \geq 2$ so that

$$\begin{aligned}
\|t \cdot \alpha(X)\|_\infty &= \|(X^k - b) \cdot \alpha(X)\|_\infty \\
&\geq |b| \cdot \|\alpha(X)\|_\infty - \|X^k \cdot \alpha(X)\|_\infty \\
&= (|b| - 1) \cdot \|\alpha(X)\|_\infty \\
&\geq \|\alpha(X)\|_\infty
\end{aligned}$$

for all $\alpha(X) \in \mathbb{Q}[X]/(X^N + 1)$. This finishes the proof. □

The notation $q' \approx q$ in Algorithm 1 means that q' and q have comparable bit lengths. Since $B_1 < q$, the term $\frac{N}{2q'} \cdot (B_1 + B_2)$ in the upper bound of $\|e'\|_\infty$ in Theorem 2 is small as long as q/q' is not too large. The concrete parameter choice depends on the instantiated values of B_1 and B_2.

3.2 Efficiency Discussions

We analyze the efficiency of the proposed bootstrapping. We first compare our method with the most relevant prior approaches [38, 30], and then discuss how different parameters affect its performance.

[7] See [18, Theorem 3.2] for proof.

Comparison to [38]. Our method extends the BFV-from-CKKS noise-refreshing approach of [38] from integer to polynomial plaintext moduli. Since their method supports only integer plaintext moduli, we restrict the comparison to this setting. Here, q and p denote the BFV ciphertext and plaintext moduli, respectively.

When $p \mid q$, the method of [38] appears to be strictly better than ours, since their bootstrapping raises the modulus by a factor of p, whereas ours raises the modulus by a factor of $q' \approx q$. In general, q is much larger than p in order to allow multiple multiplications within a single bootstrapping cycle. We now consider the case $p \nmid q$. As briefly discussed in [38, Section 5.1], one can handle this case by modulus switching to a modulus $q' \approx q$ such that $p \mid q'$, running the bootstrapping with respect to q', and then modulus switching back to q. Importantly, every modulus used during the bootstrapping is divisible by p. In contrast, the modulus q in our bootstrapping is completely independent of p, at the cost of raising the modulus by a factor of $q' \approx q$ instead of p.

For efficiency, most state-of-the-art FHE implementations use the Residue Number System (RNS; see [20]) to construct ciphertext moduli. In this representation, a ciphertext modulus is a product of word-sized NTT-friendly primes, enabling a uniform NTT-based arithmetic pipeline. The plaintext modulus p, however, is generally not NTT-friendly. Requiring the ciphertext modulus to be divisible by p therefore moves the implementation away from this fully uniform setting. Non-NTT-friendly moduli can be handled in principle, for example by special-case arithmetic, partial splitting, or embedding into a larger modulus, but this introduces heterogeneity in the RNS representation and may incur additional overhead. In contrast, our construction keeps the ciphertext modulus independent of p and can remain entirely within the standard NTT-friendly RNS pipeline.

Comparison to [30]. The most relevant prior GBFV bootstrapping method is [30]. Their construction is highly efficient for small and moderate plaintext moduli; for example, it gives excellent performance for common choices such as $p = 2^{16} + 1$. Our method is not intended to improve this small-modulus regime. Instead, the advantage of our approach appears as the plaintext precision grows.

The bootstrapping of [30] relies on conversions between GBFV and BFV, and in particular on BFV linear transformations. The modulus consumption of these transformations grows with the plaintext precision. In contrast, our construction extracts the GBFV noise and refreshes it through high-precision CKKS bootstrapping. Once the target CKKS precision is fixed, the modulus consumption of our bootstrapping is essentially independent of the resulting plaintext modulus size.

This difference becomes significant already in the few-hundred-bit regime. Under typical parameter choices, such as $N \leq 2^{16}$ and a ciphertext-modulus budget of about 1000 bits, the BFV-linear-transformation component of [30] can consume on the order of $6 \log(p)$ bits of modulus. Thus, for a few-hundred-bit plaintext modulus, this component alone can approach the available modulus budget, leaving little room for the remaining bootstrapping computation.

Therefore, the two approaches are complementary. The method of [30] is preferable for small plaintext moduli, where its BFV linear transformation cost is manageable and its denoising factor can be large. Our method targets the large-precision regime, where avoiding BFV linear transformations is more important than optimizing the small-modulus bootstrapping latency. This is the regime needed for high-precision integer computations inside FHE.

META-BTS [3]. The most important metric for GBFV bootstrapping performance is the denoising factor, defined in [38] for BFV. It measures the ratio between the input and output GBFV noises.

Definition 2 (Denoising Factor). *Let $\lfloor (q/t) \cdot m \rceil + e$ and $\lfloor (q/t) \cdot m \rceil + e'$ be the input and output plaintexts of GBFV bootstrapping, respectively. Suppose that the output noise satisfies $\|e'\|_\infty < B'$ whenever the input noise satisfies $\|e\|_\infty < B$. The denoising factor is defined as $\log(B/B')$.*

Since noise grows after each multiplication, the denoising factor corresponds to the multiplicative depth available between two consecutive bootstrappings. Theorem 2 shows that the denoising factor is proportional to the precision of the underlying CKKS bootstrapping. Thus, higher-precision CKKS bootstrapping yields larger GBFV multiplicative depth. One state-of-the-art method for high-precision CKKS bootstrapping is META-BTS [3], which iteratively bootstraps the errors to achieve higher precision. In particular, the bootstrapping precision is proportional to the number of iterations.

Theorem 3 (Theorem 3.2 of [3], Simplified). *Given a p-bit precision CKKS bootstrapping $\mathsf{BTS} : \mathcal{R}_q^2 \to \mathcal{R}_Q^2$, we may iterate BTS k times and achieve a kp-bit precision CKKS bootstrapping $\mathsf{BTS}^{(k)} : \mathcal{R}_{2^{(k-1)p} \cdot q}^2 \to \mathcal{R}_Q^2$.*

Combining this with Theorem 2 gives the following corollary.

Corollary 1 (META-BTS for GBFV). *Given a CKKS bootstrapping with p bits of bootstrapping precision, we may iterate the bootstrapping $k \geq 1$ times to construct a GBFV bootstrapping whose denoising factor is $kp - O(1)$ and whose computational complexity is $O(k)$.*

See [38, Section 5.2] for more details, as most of the discussion there also applies to our GBFV bootstrapping.

Our GBFV bootstrapping is compatible with any CKKS bootstrapping procedure. For instance, one may replace META-BTS with the recent work [25] to obtain better efficiency. In that case, the asymptotic analysis becomes slightly more involved, since parameters such as the ring dimension and ciphertext modulus grow slowly with the bootstrapping precision.

The Choice of b. In practice, we may choose $t(X) = X^k - b$ so that the plaintext modulus is $b^{N/k} + 1$ and the number of slots is k. The parameter k gives a tradeoff between plaintext modulus size and the number of slots, while b does not affect the number of slots. When targeting high precision, one may increase $|b|$ instead of decreasing k. Theorem 2 shows that both the input and output noise bounds of our bootstrapping depend on b.

Corollary 2 (Theorem 2, Reinterpreted). *Given a fixed CKKS bootstrapping algorithm, the input noise bound is $\Omega(1/|b|)$ and the output noise bound is $O(|b|)$, giving an $O(1) - O(\log |b|)$ denoising factor.*

Since the plaintext modulus $b^{N/k} + 1$ has bit-size $O(\log |b|)$, increasing the plaintext modulus bit-size by increasing $|b|$ decreases the denoising factor linearly in bits. Concretely, since N/k can already be in the thousands, choosing a very large b provides limited benefit while increasing the bootstrapping parameters. The rough behavior of noise growth can be observed in Table 2 in Section 5.1.

Extension to non-power-of-two cyclotomics. At the level of underlying plaintexts, our GBFV bootstrapping consists of the following three steps, with rounding and bootstrapping errors omitted for clarity:

- Scaled error extraction: $\Delta \cdot m + e \pmod q \mapsto t \cdot e \pmod q$.
- CKKS bootstrapping: $t \cdot e \pmod q \mapsto t \cdot e \pmod{qq'}$.
- Scaling and rounding: $t \cdot e \pmod{qq'} \mapsto e \pmod q$.

Subtracting the recovered noise from the original ciphertext completes the refresh.

The same construction extends to a general cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(\Phi_M(X))$, provided that t is invertible in $\mathbb{Q}[X]/(\Phi_M(X))$ and a suitable CKKS bootstrapping procedure is available over $\mathcal{R}$.

CKKS [22] is defined over general cyclotomic rings, but its homomorphic encoding and decoding transforms must be adapted to the ring structure for efficient bootstrapping. When the conductor M factors into small prime powers, mixed-radix butterfly decompositions can make these transforms efficient. For example, MRFHE [21] implements CKKS over cyclotomic rings whose conductors are products of powers of two and three, and exploits radix-2 and radix-3 decompositions to accelerate batch bootstrapping.

4 Conversions between GBFV and CKKS

In this section, we develop conversions between GBFV and CKKS and discuss how they can support operations over GBFV plaintext spaces. The starting point is to represent a GBFV plaintext, which is defined in the quotient ring $\mathcal{R}_t = \mathcal{R}/t\mathcal{R}$, by a bounded polynomial in the larger ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. For $t(X) = X^k - b$, the coefficients of this representative reveal the base-b digit structure of the plaintext. Our GBFV-to-CKKS conversion homomorphically

extracts this digit-wise representation into CKKS, while the reverse conversion recombines it back into a GBFV ciphertext. In this section, we assume $b > 0$ for simplicity.

4.1 Conversions

We first recall the representation map used in [30]. Unlike the GBFV-to-BFV conversion of [30], our conversion identifies a GBFV ciphertext with a corresponding digit-wise representation in CKKS.

Definition 3 (Flatten [30]). *We identify an element in $\mathcal{R}_t$ as an element in $\mathcal{R}$ via the embedding $\mathsf{Flatten} : \mathcal{R}_t \to \mathcal{R}$ defined as*

$$\mathsf{Flatten}(m) = t \cdot \left[\frac{m}{t}\right]_1$$

where $[\cdot]_1$ denotes the (signed) modulo 1 function.

By definition, we have

$$\mathsf{Flatten}(m) = t \cdot \left[\frac{m}{t}\right]_1 \equiv_t \left[t \cdot \frac{m}{t}\right]_t = m,$$

which means that $\mathsf{Flatten}(m)$ is indeed the representation of m. In fact, if m was identified with $\mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b)$, the coefficients of $\mathsf{Flatten}(m)$ can be understood as digit decompositions of m. This was formally discussed for $t(X) = X - b$ case in [17], and implicitly used for general $t(X) = X^k - b$ in [30].

Lemma 1 (Flatten as Decomposition). *Let $m = \sum_{i=0}^{k-1} m_i \cdot X^i \in \mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b) \cong \mathcal{R}_t$. Let $\mathsf{Flatten}(m) = \sum_{i=0}^{N-1} \mu_i \cdot X^i \in \mathcal{R}$. Then we have*

$$m_i \equiv_{b^{N/k}+1} \sum_{j=0}^{N/k-1} \mu_{kj+i} \cdot b^j$$

and $|\mu_i| \leq \frac{b+1}{2}$ for all $0 \leq i < N$.

Proof. To prove the first (modular) equality, we see that

$$\sum_{i=0}^{k-1} m_i \cdot X^i = m \equiv_t \sum_{i=0}^{N-1} \mu_i \cdot X^i \equiv_t \sum_{i=0}^{k-1} \left(\sum_{j=0}^{N/k-1} \mu_{kj+i} \cdot b^j \right) \cdot X^i.$$

The last equality identified X^k with b. Since the leftmost and rightmost polynomials are of degree $\leq k-1$, each coefficient should be the same modulo $b^{N/k}+1$, as desired. It remains to prove that each $|\mu_i|$ are small. This is clear by definition:

$$\|\mathsf{Flatten}(m)\|_\infty = \|t \cdot [m/t]_1\|_\infty = \|(X^k - b) \cdot [m/t]_1\|_\infty \leq (b+1) \cdot \|[m/t]_1\|_\infty \leq \frac{b+1}{2}.$$

□

We observe that a GBFV ciphertext encrypting m can be regarded as a ciphertext encrypting $\mathsf{Flatten}(m)$ up to some small errors.

Theorem 4. *Let* $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ *be a GBFV ciphertext encrypting a message* $m \in \mathcal{R}_t$*, satisfying* $\mathsf{ct} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e$ *and* $\|e\|_\infty < q/2$*. Then we have*

$$[\mathsf{ct} \cdot (1, s)]_q = \lfloor \Delta \cdot \mathsf{Flatten}(m) \rceil + q \cdot I + e$$

where $I \in \mathcal{R}$ *is a small polynomial satisfying* $\|I\|_\infty \leq 1$*.*

If the size of the message is small, we get a stronger result. That is, the smallness of a message implies smaller degree in the corresponding polynomial. This turns out to be useful in the later applications.

Lemma 2. *Let* $m \in \mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b) \cong \mathcal{R}_t$ *be a polynomial. If* $\|m\|_\infty \leq b^\ell$ *for some* $\ell < N/k$*, then* $\deg(\mathsf{Flatten}(m)) \leq k(\ell + 1) - 1$*.*

Theorem 5. *Let* $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ *be a GBFV ciphertext encrypting a message* $m \in \mathcal{R}_t$*, satisfying* $\mathsf{ct} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e$*,* $\|m\|_\infty \leq b^\ell$ *for some* $\ell < N/k$ *as an element of* $\mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b) \cong \mathcal{R}_t$ *and* $\|e\|_\infty < q/4$*. Then we have*

$$[\mathsf{ct} \cdot (1, s)]_q = \lfloor \Delta \cdot \mathsf{Flatten}(m) \rceil + q \cdot I + e$$

where $I \in \mathcal{R}$ *is a small polynomial satisfying* $\|I\|_\infty \leq 1$ *and* $\deg(I) \leq k(\ell+1)-1$*.*

The proofs of Theorem 4, Lemma 2, and Theorem 5 are deferred to Appendix C.

Next, we consider a variant of CKKS bootstrapping that allows messages to be put in the most significant bits. Such bootstrapping can be instantiated as in [37] via the iterative discrete bootstrapping. Theorem 4 and 5, when compiled with this CKKS bootstrapping, give a conversion from GBFV to CKKS:

Algorithm 3: GBFV to CKKS Conversion

Setting: $\Delta = q/t$. $\mathsf{CKKS.Bootstrap} : \mathcal{R}_q^2 \to \mathcal{R}_Q^2$ has input space $[-q/2, q/2)^N$.
Input : $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ such that $\mathsf{ct} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e$.
Output: $\mathsf{ct}_{\mathrm{out}} = (c_0', c_1') \in \mathcal{R}_Q^2$ such that
$[\mathsf{ct}_{\mathrm{out}} \cdot (1, s)]_Q = \lfloor q \cdot \mathsf{Flatten}(m) \rceil + q \cdot t \cdot I + e'$, $\|I\|_\infty \leq 1$.

```
ct_out ← t · CKKS.Bootstrap(ct);
return ct_out
```

Theorem 6 (GBFV to CKKS). *Let* $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ *be a GBFV ciphertext encrypting a message* $m \in \mathcal{R}_t$*. That is,*

$$\mathsf{ct} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e$$

for $\Delta = q/t$ and some error $e \in \mathcal{R}$ satisfying $\|e\|_\infty \ll q/2$. Let $\mathsf{CKKS.Bootstrap} : \mathcal{R}_q^2 \to \mathcal{R}_Q^2$ *be a CKKS bootstrapping procedure that accepts any underlying plaintext in $[-q/2, q/2)^N$ and approximately preserves it while raising the modulus to Q. Then*

$$t \cdot \mathsf{CKKS.Bootstrap}(\mathsf{ct}) \in \mathcal{R}_Q^2$$

can be regarded as a CKKS ciphertext, with scaling factor q, encrypting

$$\mathsf{Flatten}(m) + t \cdot I$$

for some $I \in \mathcal{R}$ satisfying $\|I\|_\infty \leq 1$. Furthermore, if $\|m\|_\infty \leq b^\ell$ as an element of $\mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b)$ for some $\ell < N/k$, then

$$\deg(\mathsf{Flatten}(m)) \leq k(\ell+1) - 1 \quad \textit{and} \quad \deg(I) \leq k(\ell+1) - 1.$$

Proof. By Theorem 4, we have

$$[\mathsf{ct} \cdot (1, s)]_q = \lfloor \Delta \cdot \mathsf{Flatten}(m) \rceil + q \cdot I + e$$

for some $I \in \mathcal{R}$ satisfying $\|I\|_\infty \leq 1$. Thus, when viewed as a CKKS ciphertext, ct has underlying plaintext

$$\lfloor \Delta \cdot \mathsf{Flatten}(m) \rceil + q \cdot I + e.$$

If $\|m\|_\infty \leq b^\ell$, then Theorem 5 further gives

$$\deg(I) \leq k(\ell+1) - 1,$$

and Lemma 2 gives

$$\deg(\mathsf{Flatten}(m)) \leq k(\ell+1) - 1.$$

Let $\mathsf{ct}' = \mathsf{CKKS.Bootstrap}(\mathsf{ct})$. By the correctness of CKKS bootstrapping, ct' lies in $\mathcal{R}_Q^2$ and satisfies

$$[\mathsf{ct}' \cdot (1, s)]_Q = \lfloor \Delta \cdot \mathsf{Flatten}(m) \rceil + q \cdot I + e + e_{\mathrm{BTS}},$$

where e_{BTS} is the CKKS bootstrapping error. Multiplying by t gives

$$[t \cdot \mathsf{ct}' \cdot (1, s)]_Q = q \cdot (\mathsf{Flatten}(m) + t \cdot I) + t \cdot (e + e_{\mathrm{BTS}}) + e_{\mathrm{Round}},$$

where e_{Round} is the rounding error from replacing $t \cdot \lfloor \Delta \cdot \mathsf{Flatten}(m) \rceil$ by $q \cdot \mathsf{Flatten}(m)$. Since $t \cdot (e + e_{\mathrm{BTS}}) + e_{\mathrm{Round}}$ is small, the ciphertext $t \cdot \mathsf{ct}'$ is a CKKS ciphertext with scaling factor q encrypting

$$\mathsf{Flatten}(m) + t \cdot I.$$

This proves the claim. □

Theorem 6 shows that the GBFV-to-CKKS conversion exposes a digit-wise representative of the GBFV plaintext. For the applications below, it is important that this representative remains small: its coefficient size grows only with b, not with the full native plaintext modulus $b^{N/k} + 1$. The following corollary makes this boundedness explicit; its proof is deferred to Appendix C.

Corollary 3. *Let* $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ *be a GBFV ciphertext encrypting a message* $m \in \mathcal{R}_t$*, and* ct' *be the output of the GBFV-to-CKKS conversion on* ct*. Let* $m' = \mathsf{Flatten}(m) + t \cdot I \in \mathcal{R}$ *be the underlying message as in Theorem 6, then we have* $\|m'\|_\infty \leq 3(b+1)/2$*.*

Remark 1. Instantiating a CKKS bootstrapping procedure that covers the full range $[-q/2, q/2)$ may introduce an additional approximation error before modulus raising, in addition to the usual bootstrapping error. In that case, the bound $\|I\|_\infty \leq 1$ in the proof of Theorem 6 does not follow immediately from Theorem 4. However, the additional pre-bootstrapping error can be absorbed into the GBFV noise term. Equivalently, we may regard the perturbed ciphertext as a valid GBFV ciphertext with slightly larger noise and then apply the same argument as in Theorem 6. Thus the conversion proof remains unchanged, up to adjusting the admissible noise bound.

The backward conversion, CKKS to GBFV, can be defined more easily through a simple modulus switching.

Theorem 7 (CKKS to GBFV). *Let* $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_{qq'}^2$ *be a CKKS ciphertext encrypting a message* $\mathsf{Flatten}(m) + t \cdot I$ *where* $m \in \mathcal{R}_t$*,* $I \in \mathcal{R}$ *satisfying* $\|I\|_\infty \leq B$*,* $q' \approx q$*, and* $B = O_q(1)$*. That is,* $[\mathsf{ct} \cdot (1, s)]_{qq'} = q \cdot (\mathsf{Flatten}(m) + t \cdot I) + e$ *for some small error* $e \in \mathcal{R}$*. Then* $\mathsf{CKKS.RS}_{q'}(\lfloor q'/t \rceil \cdot \mathsf{ct}) \in \mathcal{R}_q^2$ *can be regarded as a GBFV ciphertext encrypting* $m \in \mathcal{R}_t$*.*

The proof is deferred to Appendix C. The following correctness statement follows by combining Theorem 6 and Theorem 7.

Corollary 4 (Conversion Correctness). *Let* $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ *be a GBFV ciphertext encrypting a message* $m \in \mathcal{R}_t$*. If we apply the GBFV-to-CKKS conversion* $: \mathcal{R}_q^2 \to \mathcal{R}_{qq'}^2$ *and the CKKS-to-GBFV conversion* $: \mathcal{R}_{qq'}^2 \to \mathcal{R}_q^2$*, the resulting GBFV ciphertext encrypts the same message* m *as before.*

4.2 Modular Reduction

We now present modular reduction as the main application of the GBFV–CKKS conversion interface developed above. The goal is to start from a GBFV ciphertext whose plaintext is represented in the large native plaintext space $\mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b)$ and homomorphically obtain a ciphertext encrypting a smaller representative modulo a target integer u.

This operation is useful for large-integer arithmetic. GBFV can support arithmetic over a large plaintext modulus $b^{N/k} + 1$ by choosing suitable parameters, but many applications require arithmetic modulo an external integer $u < b^{N/k} + 1$, as in recent work on large-integer computation with discrete CKKS [6]. Our construction uses the GBFV-to-CKKS conversion to expose a bounded digit representation, applies modular reduction on this CKKS-side representation, and then converts the result back to GBFV.

We first record the elementary digit-level operation underlying the construction.

Lemma 3. *Let $u < b^\ell$ be a positive integer for some $\ell < N/k$. Let $\Omega = \sum_{i=0}^{\ell-1} \omega_i \cdot b^i$ be a possibly redundant b-digit representation of $\Omega \in \mathbb{Z}$, where $|\omega_i| \leq B$ for each i. Then $\Omega' = \sum_{i=0}^{\ell-1} \omega_i \cdot [b^i]_u$ satisfies $\Omega' \equiv_u \Omega$ and $|\Omega'| \leq Bu\ell/2$.*

Proof. By definition of Ω',

$$\Omega' = \sum_{i=0}^{\ell-1} \omega_i \cdot [b^i]_u \equiv_u \sum_{i=0}^{\ell-1} \omega_i \cdot b^i = \Omega.$$

Moreover,

$$|\Omega'| \leq \sum_{i=0}^{\ell-1} B \cdot (u/2) = Bu\ell/2.$$

This finishes the proof. □

We apply Lemma 3 to the digit representation obtained from Algorithm 3. Let $u < b^\ell$ for some $\ell < N/k$. Let $\Phi(X) = \sum_{i=0}^{N-1} \varphi_i \cdot X^i \in \mathcal{R}$ be such that $|\varphi_i| \leq B$ for some $B > 0$ and $\deg(\Phi) \leq k(\ell+2) - 1$. The polynomial Φ corresponds to

$$\check{\Phi}(X) = \sum_{i=0}^{k-1} \left(\sum_{j=0}^{\ell+1} \varphi_{kj+i} \cdot b^j \right) \cdot X^i = \sum_{i=0}^{k-1} \Omega_i \cdot X^i \in \mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b).$$

We precompute $[b^i]_u$ for $0 \leq i \leq \ell+1$ and write each of them in balanced base-b form as

$$[b^i]_u = \sum_{j=0}^{\ell-1} \beta_{ij} \cdot b^j, \qquad |\beta_{ij}| \leq b/2.$$

For $\ell \leq j < N/k$, set $\beta_{ij} = 0$. We define the digit-wise modular reduction of Φ by

$$\Phi'(X) = \sum_{i=0}^{k-1} \left[\sum_{j=0}^{\ell+1} \left(\varphi_{kj+i} \cdot \left(\sum_{v=0}^{\ell-1} \beta_{jv} \cdot X^{kv} \right) \right) \cdot X^i \right] \tag{2}$$

$$= \sum_{i=0}^{k-1} \left[\sum_{v=0}^{\ell-1} \left(\sum_{j=0}^{\ell+1} \varphi_{kj+i} \cdot \beta_{jv} \right) \cdot X^{kv+i} \right] \in \mathcal{R}. \tag{3}$$

Since $X^k \equiv_t b$, we have

$$\Phi'(X) \equiv_t \sum_{i=0}^{k-1} \left(\sum_{j=0}^{\ell+1} \varphi_{kj+i} \cdot [b^j]_u \right) \cdot X^i.$$

Thus Φ' represents the result of applying Lemma 3 to $\check{\Phi}$.

It remains to realize this operation homomorphically. In CKKS, the above plaintext operation is a linear transformation on the coefficient vector. Specifically, it maps $(\varphi_0, \varphi_1, \ldots, \varphi_{N-1})$ to

$$\left(\sum_{j=0}^{\ell+1} \varphi_{kj+i} \cdot \beta_{jv}\right)_{0 \le kv+i < N},$$

whose $(kv+i)$-th coordinate is $\sum_{j=0}^{\ell+1} \varphi_{kj+i} \cdot \beta_{jv}$. This is a matrix-vector multiplication and can be instantiated with state-of-the-art homomorphic linear-transformation methods such as [32]. Let $\mathsf{LinTrans}$ denote the corresponding homomorphic linear transformation. The correctness of applying Lemma 3 homomorphically relies on the following lemma.

Lemma 4. *Let $m \in \mathcal{R}_t$ satisfies $\|m\|_\infty \le b^\ell$ as an element of $\mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b)$ for some $\ell < N/k - 5$. Let $\tilde{m} \in \mathbb{Z}[X]/(X^k - b)$ be a natural embedding of m via the signed embedding $\mathbb{Z}_{b^{N/k}+1} = [-(b^{N/k}+1)/2, (b^{N/k}+1)/2) \hookrightarrow \mathbb{Z}$. Let $m' = \mathsf{Flatten}(m) + t \cdot I \in \mathcal{R}$ satisfies $I \in \mathcal{R}$, $\|I\|_\infty \le 1$, and $\deg(I) \le k(\ell+1)-1$. Let $\tilde{m'} \in \mathbb{Z}[X]$ be a natural embedding of m' via $\mathcal{R} = \mathbb{Z}[X]/(X^N+1) \hookrightarrow \mathbb{Z}[X]$. Then we have $[\tilde{m'}]_t = \tilde{m}$ as an element of $\mathbb{Z}[X]/(X^k - b)$.*

The proof is deferred to Appendix C.

Now we are ready to describe our approximate modular reduction leveraging CKKS, which we describe in Algorithm 4.

Algorithm 4: Approximate Modular Reduction through CKKS

Setting: $\Delta = q/t$. $q' \approx q$ and $Q \gg q$. $\mathsf{GBFV\text{-}to\text{-}CKKS} : \mathcal{R}_q^2 \to \mathcal{R}_Q^2$ and $\mathsf{CKKS\text{-}to\text{-}GBFV} : \mathcal{R}_{qq'}^2 \to \mathcal{R}_q^2$ are conversions from GBFV to CKKS and vice versa, as in the previous subsection. Let $u > 0$ be the target modulus for modular reduction. Let $\mathsf{LinTrans}$ be the homomorphic linear transformation as defined previously.

Input : $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$ such that $\mathsf{ct} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m \rceil + e$, where $\|m\|_\infty < b^\ell$ as an element of $\mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b)$ for some $\ell < N/k - 5$.

Output: $\mathsf{ct}_{\mathrm{out}} = (c_0', c_1') \in \mathcal{R}_q^2$ such that $\mathsf{ct}_{\mathrm{out}} \cdot (1, s) \equiv_q \lfloor \Delta \cdot m' \rceil + e'$ for some $m' \in \mathcal{R}_t$ such that $m' \equiv m \pmod{u}$ as an element of $\mathbb{Z}[X]/(X^k - b)$ and $\|m'\|_\infty \ll \|m\|_\infty$.

1 $\mathsf{ct}_{\mathrm{out}} \leftarrow \mathsf{CKKS\text{-}to\text{-}GBFV} \circ \mathsf{LinTrans} \circ \mathsf{GBFV\text{-}to\text{-}CKKS}(\mathsf{ct})$;

2 **return** $\mathsf{ct}_{\mathrm{out}}$

Theorem 8 (Modular Reduction). *Let $\mathsf{ct} = (c_0, c_1)$ be a GBFV ciphertext encrypting $m \in \mathcal{R}_t$ which satisfies $\|m\|_\infty < b^\ell$ as an element of $\mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b)$ for some $\ell < N/k - 5$. Assume that the ciphertext parameters and CKKS*

precision are sufficient to ensure correct decryption after the composed procedure. Then the output of Algorithm 4 is a GBFV ciphertext encrypting $m' \in \mathbb{Z}_{b^{N/k}+1}[X]/(X^k - b)$ *such that* $m' \equiv m \pmod{u}$ *as an element of* $\mathbb{Z}[X]/(X^k - b)$ *and* $\|m'\|_\infty \leq 3(b+1)u(\ell+2)/2$.

Proof. After applying GBFV-to-CKKS, we obtain a CKKS ciphertext encrypting

$$\hat{m} = \mathsf{Flatten}(m) + t \cdot I$$

with scaling factor q, where $I \in \mathcal{R}$ satisfies $\|I\|_\infty \leq 1$. Moreover, by Theorem 5, we have $\deg(I) \leq k(\ell+1)-1$, and by Lemma 2, $\deg(\mathsf{Flatten}(m)) \leq k(\ell+1)-1$. Hence

$$\deg(\hat{m}) \leq \max\{\deg(\mathsf{Flatten}(m)), \deg(t) + \deg(I)\} \leq k(\ell+2) - 1.$$

In addition, Corollary 3 gives

$$\|\hat{m}\|_\infty \leq 3(b+1)/2.$$

Next, let Φ' denote the CKKS plaintext obtained by applying LinTrans to $\hat{m} = \sum_{i=0}^{N-1} \varphi_i X^i$. By Lemma 4, the signed coefficients m_i of m satisfy

$$m_i = \sum_{j=0}^{\ell+1} \varphi_{kj+i} b^j$$

as integers. Define

$$M_i = \sum_{j=0}^{\ell+1} \varphi_{kj+i} [b^j]_u \in \mathbb{Z}.$$

Then $\Phi' \equiv_t \sum_{i=0}^{k-1} M_i X^i$ and $M_i \equiv_u m_i$. To ensure that this congruence is preserved in the native plaintext space, let $P = b^{N/k} + 1$. Since $|[b^j]_u| \leq b^j$, we have

$$\begin{aligned} |M_i| &\leq \frac{3(b+1)}{2} \sum_{j=0}^{\ell+1} |[b^j]_u| \\ &\leq \frac{3(b+1)(b^{\ell+2}-1)}{2(b-1)} \leq b^{\ell+5} < P/2, \end{aligned}$$

where the last inequality uses $\ell < N/k - 5$ and $b \geq 2$. Thus $m' = \sum_{i=0}^{k-1} M_i X^i \in \mathbb{Z}_P[X]/(X^k - b)$ has signed coefficients M_i, so $m' \equiv m \pmod{u}$ as an element of $\mathbb{Z}[X]/(X^k - b)$. Moreover, Lemma 3 gives

$$\|m'\|_\infty \leq 3(b+1)u(\ell+2)/4,$$

which implies the stated bound.

It remains to convert this CKKS ciphertext back to GBFV. To apply Theorem 7, we need to write

$$\Phi' = \mathsf{Flatten}(m') + t \cdot I'$$

with $\|I'\|_\infty = O_q(1)$. We first bound $\|\Phi'\|_\infty$. Revisiting Line (3), where Φ' is computed for modular reduction, we have

$$\begin{aligned}\|\Phi'\|_\infty &= \max_{i,v} \left| \sum_{j=0}^{\ell+1} \varphi_{kj+i} \cdot \beta_{jv} \right| \\ &\le (\ell+2) \cdot \max_i |\varphi_i| \cdot \max_{j,v} |\beta_{jv}| \\ &\le (\ell+2) \cdot (3(b+1)/2) \cdot (b/2) \\ &= 3b(b+1)(\ell+2)/4 = O_q(1).\end{aligned}$$

Since $\Phi' \equiv_t m' \equiv_t \mathsf{Flatten}(m')$, we have

$$I' = \frac{\Phi' - \mathsf{Flatten}(m')}{t} \in \mathcal{R}.$$

The inequality $\|t\alpha\|_\infty \ge (b-1)\|\alpha\|_\infty$ from the proof of Theorem 2 therefore gives

$$\|I'\|_\infty \le \frac{\|\Phi'\|_\infty + (b+1)/2}{b-1} = O_q(1).$$

Thus the input to the CKKS-to-GBFV conversion satisfies the boundedness condition required by Theorem 7. Under the assumed ciphertext parameters and CKKS precision, Theorem 7 applies, and the final output is a GBFV ciphertext encrypting the claimed plaintext m'. This finishes the proof. □

We focus on modular reduction as the main application of the conversion interface. The same digit-decomposed CKKS representation can also be used for interpolation-based non-arithmetic operations over large GBFV plaintext spaces; we briefly discuss this direction in Appendix B.

5 Experimental Results

Our experiments aim to validate two claims: first, that CKKS-based GBFV bootstrapping remains practical in very large plaintext-modulus regimes, including CLPX; and second, that the GBFV–CKKS conversion can support useful non-native operations such as modular reduction. We provide proof-of-concept implementation results for our GBFV bootstrapping in Section 3 and our GBFV-to-CKKS conversion in Section 4. Our code is built upon the golang lattigo library [1], and all experiments are run single-threaded on an Apple M4 Max processor with 128GB of RAM running macOS Sequoia ≥15.4.1. The code is available as public repositories[8]. All of our parameters satisfy 128 bits of security according to [8]. All the experiments were measured after at least 20 iterations.

[8] https://github.com/jaehyungkim0/GBFV-Boot-CKKS-1 and https://github.com/jaehyungkim0/GBFV-Boot-CKKS-2

5.1 GBFV Bootstrapping

We first describe the FHE parameter set used for the GBFV bootstrapping experiments. As in Section 3, we write q for the GBFV ciphertext modulus, which is also the input modulus of the CKKS bootstrapping procedure $\mathsf{CKKS.Bootstrap} : \mathcal{R}_q^2 \to \mathcal{R}_{qq'}^2$. We write qq' for the output modulus of $\mathsf{CKKS.Bootstrap}$. In our experiments, q is the product of one 60-bit NTT prime and one 45-bit NTT prime, while q' is the product of two 45-bit NTT primes. Using META-BTS [3], this parameter set supports CKKS bootstrapping with approximately 80 bits of precision in about 18 seconds. The detailed parameters are given in Table 1.

Table 1. FHE parameter set used in the GBFV bootstrapping experiments. Here N denotes the ring dimension, QP denotes the maximum switching-key modulus, $(h, \tilde{h})$ denote the dense and sparse Hamming weights for sparse-secret encapsulation [8], respectively, q denotes the GBFV ciphertext modulus, which is also the CKKS bootstrapping input modulus, and qq' denotes the CKKS bootstrapping output modulus.

$\log N$	$\log QP$	$(h, \tilde{h})$	$\log q$	$\log q'$
16	1288	$(192, 32)$	$60 + 45$	45×2

We evaluate the GBFV bootstrapping algorithm of Algorithm 2, using CKKS bootstrapping as a black box. We vary the parameters b and k in the plaintext polynomial $t(X) = X^k - b$, which gives different plaintext moduli and slot counts. First, we fix $b = 2$ and vary k. This changes the plaintext modulus $p = b^{N/k} + 1$ and the maximum number of slots k, illustrating the precision–parallelism tradeoff of GBFV. Second, we fix $k = 1$, which corresponds to the CLPX setting [17], and vary b. This yields extremely large plaintext moduli; for example, when $b = 256$ and $N = 2^{16}$, the plaintext modulus has bit-size

$$\log_2(2^{2^{19}} + 1) \approx 2^{19} = 524,288,$$

which matches the $500,000$ bit claim in the abstract. Across these parameter choices, the bootstrapping time and denoising factor remain nearly unchanged, because the dominant cost is the same CKKS bootstrapping procedure. The detailed results are shown in Table 2.

5.2 Comparison with Prior Works

We first compare our results with the existing GBFV bootstrapping method of [30].[9] Since [30] reports experiments for the plaintext modulus $p = 2^{16} + 1$, we compare with our result for the same plaintext modulus. The work of [30] considers two optimization targets, namely latency and throughput, and we compare

[9] A more recent work [31] reports improved figures for similar parameters, corresponding roughly to a 1–2× speedup. This does not substantially affect the qualitative comparison.

Table 2. GBFV bootstrapping experimental results. Here $t(X) = X^k - b$, T_{BTS} denotes the GBFV bootstrapping time, $p = b^{N/k} + 1$ denotes the plaintext modulus, the denoising factor denotes $\log(B_1/B_2)$ where B_1 and B_2 are the input and output noise upper bounds, respectively, and bits per multiplication denotes the estimated noise growth per multiplication, computed from the formula in [30].

b	k	$\log(p-1)$	T_{BTS} (sec)	Denoising Factor (bits)	Bits per multiplication
2	1	2^{16}	18.4	77.1	13
	4	2^{14}	18.5	77.3	
	16	2^{12}	18.4	77.3	
	64	2^{10}	18.3	77.3	
	256	2^{8}	18.5	77.2	
	1024	64	18.4	77.2	
	4096	16	18.4	77.1	
4	1	2^{17}	18.4	77.4	14
8		$3 \cdot 2^{16}$	18.4	74.7	15
16		2^{18}	18.4	74.1	16
32		$5 \cdot 2^{16}$	17.9	74.1	17
256		2^{19}	18.4	70.8	18

against both. For the $N = 2^{14}$ implementations of [30], our method attains a comparable denoising factor only relative to the high-throughput setting, and even then at substantially higher latency. Relative to their $N = 2^{15}$ implementation, our method supports twice as many slots, but achieves a much smaller denoising factor. One could in principle match this denoising factor by increasing the number of META-BTS iterations by a factor of $374/70.1 \approx 5.3$. Under this normalization, the $N = 2^{15}$ implementation of [30] would have approximately $7.4\times$ lower latency and $3.7\times$ lower amortized latency. Thus, for small plaintext moduli such as $p = 2^{16} + 1$, the bootstrapping method of [30] remains preferable.

Table 3. Comparison with [30] for plaintext modulus $p = 2^{16} + 1$. The figures for [30] are taken from [30, Tables 1 and 3], including both single and batched bootstrapping experiments.

	N	# slots	T_{BTS} (sec)	Denoising Factor (bits)
[30]	2^{14}	1024	1.94	124
		2048	1.98	115
		4096	1.98	90
		8192	2.01	38
	2^{15}	32768	13.24	374
Ours	2^{16}	65536	18.4	70.1

The comparison changes in the large-precision regime. The bootstrapping method of [30] relies on conversions between GBFV and BFV, and in particular on BFV linear transformations. During this step, the GBFV plaintext space

is handled through an integer-modulus BFV plaintext space whose modulus is roughly $p = b^{N/k} + 1$. The modulus consumption of this component grows with the bit-size of p. For example, under typical parameters such as $N \leq 2^{16}$ and a ciphertext-modulus budget of about 1000 bits, a component consuming on the order of $6\log(p)$ bits of modulus already becomes restrictive once p has a few hundred bits. As N/k grows, this bottleneck becomes even more severe, especially near the CLPX setting [17], where $k = 1$.

We also compare our method with CKKS-style integer-computer approaches [6, 35, 13], which target large-precision arithmetic within RLWE ciphertexts. Since these works typically bootstrap after each homomorphic multiplication, Table 4 compares their multiplication latency with our GBFV bootstrapping latency. At 256-bit precision, our latency is comparable to [6] and lower than [35, 13], while our amortized latency is lower than all three. At 2048-bit precision, our latency remains 18.5 seconds, about 2.0× faster than [35] and 5.9× faster than [13], with even larger gains in amortized latency. Moreover, GBFV is leveled and supports about four multiplications per bootstrapping, further reducing the effective cost in workloads with multiple multiplications between bootstrappings.

	Base	$\lfloor \log p \rceil$	# slots	Latency (sec)	Amortized Latency (ms)	Mult per Boot
[6]	N/A	256	128	18.3	143	1
[13]	16	256	256	74.8	292	
		2048	32	109	3410	
[35]	16	256	512	31.9	62.3	
		2048	64	37.7	589	
Ours	16	256	1024	18.5	18.1	≈ 4
		2048	128	18.5	145	

Table 4. Comparison with CKKS-style integer-computer approaches [6, 13, 35]. For each prior work and precision, we use the fastest reported parameter set, regardless of the exact plaintext-modulus shape. Base denotes the radix base when applicable, and equals b for our method. Latency is the homomorphic-multiplication latency for prior works and the GBFV bootstrapping latency for ours; amortized latency is latency divided by the number of slots. Mult per boot denotes the estimated number of multiplications supported between two consecutive bootstrappings.

This comparison should not be interpreted as saying that our framework uniformly outperforms these approaches. Our native plaintext moduli are restricted to cyclotomic values, for example of the form $b^{N/k} + 1$. In contrast, [35, 13] support power-of-two moduli, while [6] supports smooth (i.e., product of small primes) moduli. Thus, the methods are best viewed as targeting different parts of the large-precision design space.

5.3 GBFV–CKKS Conversion

We provide experimental results for the GBFV-to-CKKS and CKKS-to-GBFV conversions described in Section 4. Table 5 gives the FHE parameter set used

for the conversion experiments. For the GBFV-to-CKKS direction, we use the iterative modular-reduction framework of [37] to convert GBFV ciphertexts into discrete CKKS ciphertexts.

Table 5. RLWE parameter sets used for the GBFV–CKKS conversion experiments. Here N denotes the ring dimension, QP denotes the maximum switching-key modulus, $(h, \tilde{h})$ denotes the dense and sparse Hamming weights for sparse-secret encapsulation [8], q denotes the simulated GBFV ciphertext modulus, and Q denotes the maximum CKKS ciphertext modulus. The first row is used for the standalone GBFV-to-CKKS conversion and the round-trip conversion without modular reduction. The modular-reduction rows reserve one extra CKKS level for the linear transformation, and therefore use 59-bit primes to keep $\log QP < 1550$.

Experiment	$\log N$	$\log QP$	$(h, \tilde{h})$	$\log q$	$\log Q$	b	k	$(P_{\mathrm{MSB}}, P_{\mathrm{Base}})$
Basic conversion / round trip	16	1500	(192, 32)	60	60×22	16	256	(24, 6)
Modular reduction, $2^{256} - 38$		1534		59	59×23			
Modular reduction, $2^{2048} + 981$							32	

We first report the basic conversion cost. The GBFV-to-CKKS conversion takes 82.3 seconds, which corresponds to four iterations of functional bootstrapping at dimension $N = 2^{16}$. A GBFV ciphertext of ring dimension N is converted into a discrete CKKS ciphertext of the same dimension. Over 20 repetitions, each coefficient of the resulting discrete CKKS plaintext lay in $\{-1, 0, 1\}$, which is consistent with Theorem 6. The theorem gives a worst-case bound, whereas the random GBFV plaintexts used in our experiments produce much smaller representatives: empirically, they are very close to the corresponding flattened GBFV plaintexts. The reverse CKKS-to-GBFV conversion is essentially a modulus-switching operation and takes only about 3.2 ms.

As a primary application of the conversion, we implement homomorphic modular reduction using Theorem 8. For each target modulus u, we choose the GBFV plaintext modulus so that $b^{N/k} + 1 \gg u^2$, accommodating integers of the size arising from products of residues modulo u. We use the reduction cost as a proxy benchmark for arbitrary-modulus integer multiplication, which requires reducing such products modulo u, and compare with [6, 13]. Our implementation converts the ciphertext from GBFV to CKKS, applies a homomorphic linear transformation for digit-wise reduction modulo u, and converts the result back to GBFV. The reported timings cover this reduction pipeline; a complete multiplication implementation additionally requires compatible plaintext packing and homomorphic product evaluation. Table 6 gives a step-by-step timing breakdown.

Table 7 compares our reduction timings with the multiplication timings reported in [6, 13].

In this proxy comparison, at 255-bit precision, our reduction latency is approximately 1.9× lower than the multiplication latency reported in [13], and the amortized latency is 3.9× lower. At 2048-bit precision, the corresponding ratios

	$u = 2^{255} - 19$ (sec)	$u = 2^{2048} + 981$ (sec)
GBFV-to-CKKS conversion	88.0	88.2
LinTrans	0.414	0.555
CKKS-to-GBFV conversion	0.537	0.542
Total latency	88.9	89.3

Table 6. Step-by-step timing breakdown for conversion-based modular reduction. The reported timings include the GBFV-to-CKKS conversion, the CKKS linear transformation implementing digit-wise reduction modulo u, and the CKKS-to-GBFV conversion, including the Slots-to-Coeffs transform. The GBFV-to-CKKS conversion time is slightly higher than the standalone conversion time above because the modular-reduction experiment uses one additional level for the subsequent linear transformation.

	Base	$\lfloor \log u \rceil$	# slots	Latency (sec)	Amortized Latency (sec)
[6]	N/A	255	32	150	4.69
		2048	4	190	47.5
[13]	16	255	128	172	1.34
		2048	16	440	27.5
Ours	16	255	256	88.9	0.347
		2048	32	89.3	2.79

Table 7. Modular reduction as a proxy benchmark for arbitrary-modulus multiplication. Our rows report reduction timings and the number of plaintext coefficients reduced per ciphertext; prior rows report multiplication timings and SIMD slot counts from [6, 13]. Amortized latency is latency divided by the number of slots. We use $u = 2^{255} - 19$ and $u = 2^{2048} + 981$ for the 255- and 2048-bit cases, respectively.

are approximately 4.9× and 9.9×. The amortized latency gap is larger when compared with [6].

Acknowledgements. The author thanks Robin Geelen for answering a question about GBFV encoding. The author was supported by the Stanford Graduate Fellowship.

The author used ChatGPT for grammar, clarity, and basic editing of author-written text. The author also used OpenAI Codex for code refactoring, implementation support, and drafting parts of the AI-use disclosure for the accompanying proof-of-concept implementation. All AI-assisted material was reviewed and approved by the author, who takes full responsibility for the submission. A detailed disclosure of AI usage is provided in Appendix A.

References

1. Lattigo v6. Online: https://github.com/tuneinsight/lattigo (Aug 2024), ePFL-LDS, Tune Insight SA
2. Alexandru, A., Kim, A., Polyakov, Y.: General functional bootstrapping using CKKS. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part III. LNCS, vol.

16002, pp. 304–337. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01881-6_10
3. Bae, Y., Cheon, J.H., Cho, W., Kim, J., Kim, T.: META-BTS: Bootstrapping precision beyond the limit. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 223–234. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560696
4. Bae, Y., Kim, J., Stehlé, D., Suvanto, E.: Bootstrapping small integers with CKKS. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 330–360. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0875-1_11
5. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M.R., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 565–596. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_19
6. Boneh, D., Kim, J.: Homomorphic encryption for large integers from nested residue number systems. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part III. LNCS, vol. 16002, pp. 338–370. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01881-6_11
7. Boneh, D., Kim, J.: A universal blinder: One-round blind signatures from FHE. Cryptology ePrint Archive, Paper 2026/574 (2026), https://eprint.iacr.org/2026/574
8. Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In: Ateniese, G., Venturi, D. (eds.) ACNS 2022. LNCS, vol. 13269, pp. 521–541. Springer, Cham (Jun 2022). https://doi.org/10.1007/978-3-031-09234-3_26
9. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. Journal of Mathematical Cryptology **14**(1), 316–338 (2020). https://doi.org/doi:10.1515/jmc-2019-0026, https://doi.org/10.1515/jmc-2019-0026
10. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_50
11. Brakerski, Z., Friedman, O., Golan, D., Gurny, A., Mutzari, D., Sheinfeld, O.: REFHE: Fully homomorphic ALU. In: Daemen, J., Thomé, E. (eds.) EUROCRYPT 2026, Part IV. LNCS, vol. 16544, pp. 183–213. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25327-9_7
12. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012. pp. 309–325. ACM (Jan 2012). https://doi.org/10.1145/2090236.2090262
13. Cha, G., Park, D., Lee, J.W.: Improved radix-based approximate homomorphic encryption for large integers via lightweight bootstrapped digit carry. In: Daemen, J., Thomé, E. (eds.) EUROCRYPT 2026, Part IV. LNCS, vol. 16544, pp. 243–273. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25327-9_9
14. Cha, H., Hwang, I., Min, S., Seo, J., Song, Y.: MatriGear: Accelerating authenticated matrix triple generation with scalable prime fields via optimized HE packing. In: Blanton, M., Enck, W., Nita-Rotaru, C. (eds.) 2025 IEEE Symposium on Security and Privacy. pp. 2453–2471. IEEE Computer Society Press (May 2025). https://doi.org/10.1109/SP61157.2025.00174

15. Chartier, P., Koskas, M., Lemou, M., Méhats, F.: Fully homomorphic encryption on large integers. Cryptology ePrint Archive, Report 2024/155 (2024), https://eprint.iacr.org/2024/155
16. Chartier, P., Koskas, M., Lemou, M., Méhats, F.: Homomorphic sign evaluation with a RNS representation of integers. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 271–296. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0875-1_9
17. Chen, H., Laine, K., Player, R., Xia, Y.: High-precision arithmetic in homomorphic encryption. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 116–136. Springer, Cham (Apr 2018). https://doi.org/10.1007/978-3-319-76953-0_7
18. Cheon, J.H., Cho, W., Kim, J., Stehlé, D.: Homomorphic multiple precision multiplication for CKKS and reduced modulus consumption. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 696–710. ACM Press (Nov 2023). https://doi.org/10.1145/3576915.3623086
19. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 360–384. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_14
20. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: Cid, C., Jacobson, Jr., M.J. (eds.) SAC 2018. LNCS, vol. 11349, pp. 347–368. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-10970-7_16
21. Cheon, J.H., Hong, S., Kang, M., Kim, J., Kim, T., Lee, C., Lee, J.: MRFHE: Mixed-radix fully homomorphic encryption with better batch bootstrapping. Cryptology ePrint Archive, Paper 2026/853 (2026), https://eprint.iacr.org/2026/853
22. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 409–437. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_15
23. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 3–33. Springer, Berlin, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_1
24. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 670–699. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92078-4_23
25. Choe, H., Kim, J., Stehlé, D., Suvanto, E.: Leveraging discrete CKKS to bootstrap in high precision. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025. pp. 1083–1097. ACM Press (Oct 2025). https://doi.org/10.1145/3719027.3765080
26. Drucker, N., Moshkowich, G., Pelleg, T., Shaul, H.: BLEACH: Cleaning errors in discrete computations over CKKS. Journal of Cryptology **37**(1), 3 (Jan 2024). https://doi.org/10.1007/s00145-023-09483-1
27. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 617–640. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_24
28. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://eprint.iacr.org/2012/144

29. Gama, M., Beni, E.H., Kang, J., Spiessens, J., Vercauteren, F.: Blind zkSNARKs: for private proof delegation and verifiable computation over encrypted data. CiC **2**(3), 10 (2025). https://doi.org/10.62056/akgyl8n4e
30. Geelen, R., Vercauteren, F.: Fully homomorphic encryption for cyclotomic prime moduli. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part III. LNCS, vol. 15603, pp. 366–397. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91131-6_13
31. Geelen, R., Vercauteren, F.: Better GBFV bootstrapping and faster encrypted edit distance computation. CiC **2**(4), 35 (2026). https://doi.org/10.62056/a3n56chdj
32. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in HElib. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 93–120. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_4
33. Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., Yoo, D.: General bootstrapping approach for rlwe-based homomorphic encryption. IEEE Transactions on Computers **73**(1), 86–96 (2024). https://doi.org/10.1109/TC.2023.3318405
34. Kim, J.: Efficient homomorphic integer computer from CKKS. IACR TCHES **2025**(4), 873–898 (2025). https://doi.org/10.46586/tches.v2025.i4.873-898
35. Kim, J.: Faster homomorphic integer computer. IACR TCHES **2026**(3), 1128–1151 (Jul 2026). https://doi.org/10.46586/tches.v2026.i3.1128-1151, https://tches.iacr.org/index.php/TCHES/article/view/13141
36. Kim, J.: Faster logical operations from discrete CKKS. Cryptology ePrint Archive, Paper 2026/732 (2026), https://eprint.iacr.org/2026/732
37. Kim, J., Noh, T.: Modular reduction in CKKS. CiC **2**(2), 17 (2025). https://doi.org/10.62056/aevur-iuc
38. Kim, J., Seo, J., Song, Y.: Simpler and faster BFV bootstrapping for arbitrary plaintext modulus from CKKS. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024. pp. 2535–2546. ACM Press (Oct 2024). https://doi.org/10.1145/3658644.3670302
39. jie Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 2021 IEEE Symposium on Security and Privacy. pp. 1057–1073. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00043
40. Peikert, C., Zarchy, D., Zyskind, G.: High-precision exact FHE made simple, general, and fast. In: Heninger, N., Rosulek, M. (eds.) CRYPTO 2026, Part II. LNCS, vol. 16801, pp. 525–557. Springer, Cham (Aug 2026). https://doi.org/10.1007/978-3-032-35374-0_17

A Disclosure of AI Tool Usage

The author used OpenAI Codex, a GPT-based programming assistant, during the preparation of this manuscript and the accompanying proof-of-concept implementation. The tool was used under author direction and review. It was not used as a source of independent scientific claims.

Codex also assisted with improving the clarity of author-written text and proofs, checking notation and citations, and correcting typographical and mathematical errors.

The use of OpenAI Codex began from author-written prototype implementations. The tool assisted with refactoring and modularizing the code, adding command-line options for one-iteration experiments, and adapting the implementation to special-case experiments. In particular, Codex assisted with reorganizing the GBFV-to-CKKS conversion and MSB bootstrapping examples, implementing round-trip conversion experiments, and adding proof-of-concept modular-reduction experiments for power-of-two, Mersenne, and pseudo-Mersenne moduli.

Codex also assisted with source-level consistency review between the implementation and the paper draft, including comparing the implemented flow with the intended conversions

$$\text{GBFV-to-CKKS}, \qquad \text{LinTrans}, \qquad \text{CKKS-to-GBFV},$$

and comparing the small-message condition used in the modular-reduction experiments with the condition stated in the paper. These comparisons were used as implementation review aids only; the author independently reviewed the relevant mathematical statements, parameters, and experimental outputs.

Codex was also used to draft code comments, README provenance statements, and this disclosure text. All AI-assisted code, comments, and text were reviewed, edited where necessary, and approved by the author. The author takes full responsibility for the correctness, appropriateness, and originality of all material in the submission and accompanying implementation.

B Arbitrary Function Evaluation via Interpolation

Another potential application of the GBFV–CKKS conversion is interpolation over large GBFV plaintext spaces. The GBFV-to-CKKS conversion can be viewed as an approximate digit decomposition: by Lemma 1, a GBFV plaintext can be represented through bounded base-b digits after applying Flatten.

When the plaintext modulus $p = b^{N/k} + 1$ is prime, a direct GBFV approach to arbitrary function evaluation would require evaluating a univariate interpolation polynomial over a field of size p, whose degree can be as large as $p - 1$. Instead, one may first convert the GBFV ciphertext to CKKS and work with the digit-decomposed representation. In this representation, the target function can be expressed by multivariate interpolation over the digit variables. Since the CKKS-side representative has coefficients bounded by Corollary 3, each variable ranges over a set of size $O(b)$ rather than $b^{N/k}$.

This changes the interpolation problem from a high-degree univariate one to a multivariate one with small individual degree. At a high level, the multiplicative depth can decrease from about

$$\log_2(b^{N/k} + 1) \approx (N/k) \log_2 b$$

to roughly

$$\log_2(N/k) + \log_2 O(b).$$

The conversion costs are negligible compared with the interpolation costs when the target precision is sufficiently large. This suggests that the GBFV–CKKS conversion may be useful for non-arithmetic operations over large GBFV plaintext spaces. We leave a full algorithmic and experimental treatment of this direction to future work.

C Deferred Proofs

Proof of Theorem 4.

Proof. By definition,

$$\begin{aligned}[\mathsf{ct}\cdot(1,s)]_q &= [\lfloor \Delta\cdot m\rceil + e]_q = [\lfloor (q/t)\cdot \mathsf{Flatten}(m)\rceil + e]_q \\ &= \left[\left\lfloor q\cdot\left[\frac{m}{t}\right]_1\right\rceil + e\right]_q = \left\lfloor q\cdot\left[\frac{m}{t}\right]_1\right\rceil + q\cdot I + e\end{aligned}$$

for some $I\in\mathcal{R}$. Note that

$$I = \frac{1}{q}\cdot\left(\left[\left\lfloor q\cdot\left[\frac{m}{t}\right]_1\right\rceil + e\right]_q - \left\lfloor q\cdot\left[\frac{m}{t}\right]_1\right\rceil - e\right) = -\left\lfloor\frac{\left\lfloor q\cdot\left[\frac{m}{t}\right]_1\right\rceil + e}{q}\right\rceil.$$

Since $\|[m/t]_1\|_\infty \le 1/2$, we have that

$$\left\|\left\lfloor q\cdot\left[\frac{m}{t}\right]_1\right\rceil + e\right\|_\infty \le \frac{q}{2} + \|e\|_\infty < q.$$

Therefore, $\|I\|_\infty \le 1$ as desired. □

Proof of Lemma 2.

Proof. Let $m(X) = \sum_{i=0}^{k-1} m_i X^i$, where $|m_i| \le b^\ell$ for each $0\le i\le k-1$. By the definition of $\mathsf{Flatten}$, we have

$$\begin{aligned}\mathsf{Flatten}(m) &= t\cdot\left[\frac{m}{t}\right]_1 = t\cdot\left(\frac{m}{t} - \left\lfloor\frac{m}{t}\right\rceil\right) = m - t\cdot\left\lfloor\frac{m}{t}\right\rceil \\ &= m(X) - (b-X^k)\cdot\left\lfloor\frac{b^{N/k-1} + b^{N/k-2}\cdot X^k + \cdots + X^{N-k}}{b^{N/k}+1}\cdot\left(\sum_{i=0}^{k-1} m_i X^i\right)\right\rceil \\ &= m(X) - (b-X^k)\cdot\left\lfloor\sum_{j=0}^{N/k-1}\sum_{i=0}^{k-1}\frac{b^{N/k-j-1}\cdot m_i}{b^{N/k}+1}\cdot X^{kj+i}\right\rceil = m(X) - (b-X^k)\cdot\alpha(X)\end{aligned}$$

where $\alpha(X)\in\mathbb{Q}[X]/(X^N+1)$. Since $|m_i|\le b^\ell$ and $b\ge 2$,

$$\left|\frac{b^{N/k-j-1}\cdot m_i}{b^{N/k}+1}\right| < \left|\frac{b^{N/k-j-1}\cdot b^\ell}{b^{N/k}}\right| = b^{\ell-j-1} \le 1/2$$

for all $j\ge\ell$. This means that $\deg(\alpha(X)) \le k\ell - 1$, leading to $\deg(\mathsf{Flatten}(m)) \le k(\ell+1)-1$ as desired. □

Proof of Theorem 5.

Proof. As in the proof of Theorem 4, we have

$$[\mathsf{ct} \cdot (1, s)]_q = \left\lfloor q \cdot \left\lceil \frac{m}{t} \right\rfloor_1 \right\rceil + q \cdot I + e$$

for some $I \in \mathcal{R}$. Note that

$$I = -\left\lfloor \frac{\lfloor q \cdot \lceil \frac{m}{t} \rfloor_1 \rceil + e}{q} \right\rceil.$$

For $\ell + 1 \leq j < N/k$ and $0 \leq i < k$, the coefficient of X^{kj+i} in m/t is

$$a_{j,i} = -\frac{b^{N/k-j-1} m_i}{b^{N/k} + 1}.$$

As in the proof of Lemma 2, we have

$$|a_{j,i}| < b^{\ell-j-1} \leq 1/4,$$

so this is also the coefficient of X^{kj+i} in $[m/t]_1$. Hence

$$I_{kj+i} = -\left\lfloor \frac{\lfloor q a_{j,i} \rceil + e_{kj+i}}{q} \right\rceil = 0.$$

Indeed, $|qa_{j,i}| < q/4$ and $|e_{kj+i}| < q/4$, together with the integrality of q and e_{kj+i}, imply $|\lfloor qa_{j,i} \rceil + e_{kj+i}| < q/2$. Thus $\deg(I) \leq k(\ell + 1) - 1$. The bound $\|I\|_\infty \leq 1$ follows from Theorem 4. □

Proof of Corollary 3.

Proof. By Theorem 6 and Lemma 1, we have

$$\begin{aligned} \|m'\|_\infty &= \|\mathsf{Flatten}(m) + t \cdot I\|_\infty = \|\mathsf{Flatten}(m) + (X^k - b) \cdot I\|_\infty \\ &\leq \|\mathsf{Flatten}(m)\|_\infty + (b + 1) \cdot \|I\|_\infty \leq 3(b + 1)/2. \end{aligned}$$

This finishes the proof. □

Proof of Theorem 7.

Proof. Write $F = \mathsf{Flatten}(m)$, $\mu = F + tI$, $A = \lfloor q'/t \rceil$, and $\delta = [q'/t]_1 = q'/t - A$. Let e_{RS} denote the rescaling error relative to unrounded division by q'. The output $\mathsf{ct}' = \mathsf{CKKS.RS}_{q'}(A\mathsf{ct})$ satisfies

$$\begin{aligned} \mathsf{ct}' \cdot (1, s) &\equiv_q \frac{A(q\mu + e)}{q'} + e_{\mathrm{RS}} \\ &= \frac{q}{t} F + qI + \frac{e}{t} - \frac{\delta(q\mu + e)}{q'} + e_{\mathrm{RS}} \\ &\equiv_q \left\lfloor \frac{q}{t} F \right\rceil + E, \end{aligned}$$

where

$$E = \left[\frac{q}{t}F\right]_1 + \frac{e}{t} - \frac{\delta(q\mu + e)}{q'} + e_{\mathrm{RS}}.$$

Using $\|\delta\|_\infty \le 1/2$, $\|\mu\|_\infty \le (b+1)(B+1/2)$, and the norm inequality from the proof of Theorem 2, we obtain

$$\begin{aligned}\|E\|_\infty \le \frac{1}{2} + \frac{\|e\|_\infty}{b-1} + \|e_{\mathrm{RS}}\|_\infty \\ + \frac{N}{2q'}\left(q(b+1)\left(B+\frac{1}{2}\right) + \|e\|_\infty\right).\end{aligned}$$

Finally, the definition of F gives

$$\lfloor (q/t)F \rceil \equiv_q \lfloor (q/t)m \rceil .$$

Thus ct' has the required GBFV form with noise E, bounded as above. In particular, decryption is correct whenever $(b+1)(\|E\|_\infty + 1/2) < q/2$. □

Proof of Lemma 4.

Proof. We write $m' \in \mathcal{R}$ as

$$m'(X) = \sum_{i=0}^{N-1} \mu_i \cdot X^i = \sum_{i=0}^{k-1}\left(\sum_{j=0}^{N/k-1} \mu_{kj+i} \cdot X^{kj}\right) X^i$$

which gives $\tilde{m'}(X) = \sum_{i=0}^{N-1} \mu_i \cdot X^i$. Then we have

$$[\tilde{m'}]_t = [\tilde{m'}]_{X^k - b} = \sum_{i=0}^{k-1}\left(\sum_{j=0}^{N/k-1} \mu_{kj+i} \cdot b^j\right) \cdot X^i \in \mathbb{Z}[X]/(X^k - b).$$

Since

$$\deg(m') \le \max(\deg(\mathsf{Flatten}(m)), \deg(t) + \deg(I)) \le k(\ell+2) - 1$$

and

$$\|m'\|_\infty \le \|\mathsf{Flatten}(m)\|_\infty + (b+1) \cdot \|I\|_\infty \le 3(b+1)/2$$

(as in the proof of Corollary 3), we have that

$$\begin{aligned}\|[\tilde{m'}]_t\|_\infty &\le \max_i\left(\sum_{j=0}^{N/k-1} |\mu_{kj+i}| \cdot b^j\right) = \max_i\left(\sum_{j=0}^{\ell+1} |\mu_{kj+i}| \cdot b^j\right) \le \sum_{j=0}^{\ell+1} 3(b+1) \cdot b^j/2 \\ &= \frac{3(b+1)\cdot(b^{\ell+2}-1)}{2(b-1)} \le b^{\ell+5} < (b^{N/k}+1)/2 \quad (\because b \ge 2, \ell < N/k - 5).\end{aligned}$$

In addition, we check that

$$[[\tilde{m'}]_t]_{b^{N/k}+1} = [[\tilde{m'}]_t]_{X^N+1} = [[\tilde{m'}]_{X^N+1}]_t = [\mathsf{Flatten}(m) + t \cdot I]_t \equiv_t m.$$

Therefore, $[\tilde{m'}]_t \in \mathbb{Z}[X]/(X^k - b)$ is a representation of m with normalized coefficients, meaning that it should be equal to $\tilde{m}$. □