Simpler and Faster BFV Bootstrapping for Arbitrary Plaintext Modulus from CKKS

Jaehyung Kim[1], Jinyeong Seo[2], and Yongsoo Song[2]

[1] CryptoLab Inc.
jaehyungkim@cryptolab.co.kr
[2] Seoul National University
{jinyeong.seo, y.song}@snu.ac.kr

Abstract. Bootstrapping is currently the only known method for constructing fully homomorphic encryptions. In the BFV scheme specifically, bootstrapping aims to reduce the error of a ciphertext while preserving the encrypted plaintext. The existing BFV bootstrapping methods follow the same pipeline, relying on the evaluation of a digit extraction polynomial to annihilate the error located in the least significant digits. However, due to its strong dependence on performance, bootstrapping could only utilize a limited form of plaintext modulus, such as a power of a small prime number.
In this paper, we present a novel approach to instantiate BFV bootstrapping, distinct from the previous digit extraction-based method. The core idea of our bootstrapping is to utilize CKKS bootstrapping as a subroutine, so the performance of our method mainly depends on the underlying CKKS bootstrapping rather than the plaintext modulus.
We implement our method at a proof-of-concept level to provide concrete benchmark results. When performing the bootstrapping operation for a 51-bits plaintext modulus, our method improves the previous digit extraction-based method by a factor of 37.9 in latency and 29.4 in throughput. Additionally, we achieve viable bootstrapping performance for large plaintext moduli, such as 144-bits and 234-bits, which has never been measured before.

Keywords: Homomorphic Encryption, Bootstrapping, BFV

1 Introduction

Homomorphic encryption is a cryptosystem that enables computation on encrypted data without decryption. Since Gentry's seminal work [Gen09], its performance and functionality have continuously improved, and it now offers viable performance for real-world applications. The most widely used HE schemes to date [BGV14, Bra12, FV12, CKKS17, CGGI20, DM15] are all based on lattice-based assumptions, Learning With Errors (LWE), or its ring-variant, Ring Learning With Errors (RLWE). Among them, RLWE-based schemes such as BFV [Bra12, FV12] and CKKS [CKKS17] are popularly deployed due to their high throughput in homomorphic operations, working in a SIMD-like manner. Let $R = \mathbb{Z}[X]/\Phi_M(X)$ and $R_q = R/qR$, where $\Phi_M(X)$ is the M-th cyclotomic polynomial. Then, both BFV and CKKS ciphertexts are in the form of pairs of polynomials in R_q, but they support different types of homomorphic operations. For BFV, it supports modular arithmetic over integers, while CKKS provides approximate arithmetic over complex numbers. Both the BFV and CKKS schemes inherently share a common limitation: the number of possible homomorphic operations is bounded. Hence, to support the evaluation of arbitrary circuits, one needs a special operation called bootstrapping that refreshes the remaining number of possible operations. However, while the goal of bootstrapping is common, its precise functionality differs for each scheme due to variations in encryption structure. Thus, bootstrapping for BFV and CKKS has been studied individually so far.

For the BFV scheme, its plaintext space is $R_t = R/tR$ for a plaintext modulus t, and a BFV encryption of a plaintext $m \in R_t$ under a secret $s \in R$ is of the form $(b = -as+\Delta m+e, a) \in R_q^2$ for some $a \in R_q$ and a small noise $e \in R$ where $\Delta = \lfloor q/t \rfloor$ is the scaling factor. The decryption is obtained by first computing $b+as = \Delta m+e \pmod q$ and then scaled by Δ. It basically supports homomorphic operations over R_t, which can emulate arithmetic over $\mathbb{Z}_t$ in a SIMD-like manner, and the size of the noise e gradually increases after each homomorphic operation. If the size of the noise e exceeds a certain threshold, it can spoil the plaintext m in the upper bits, leading to decryption failure. Thus, to keep performing homomorphic operations, one needs to decrease the size of the noise while preserving the plaintext in the upper bits. This is the exact functionality of BFV bootstrapping. In the previous literature [HS15, CH18], BFV bootstrapping is achieved through a digit extraction procedure, which corresponds to homomorphically evaluating the rounding function on coefficients of plaintext. However, its performance is notably influenced by the number-theoretic properties of t. Specifically, it provides efficient performance only when the plaintext modulus t is in the form of a power of small primes. Thus, the choice of the plaintext modulus is limited for the efficiency of BFV bootstrapping.

On the other hand, the plaintext space of the CKKS scheme is R, and a CKKS ciphertext is the form of $(b = -as + m + e, a) \in R_q^2$, where $m \in R$ is a plaintext encoding a vector of complex numbers. The decryption is obtained simply by $b+as = m+e \pmod{q}$. It supports approximate arithmetic over complex numbers $\mathbb{C}$ in a SIMD-like manner. The key distinction from BFV is that the size of the ciphertext modulus q keeps decreasing after each homomorphic operation. If the size of q becomes smaller than the plaintext m, it results in decryption failure. Thus, in CKKS, one needs to increase the ciphertext modulus while approximately preserving the plaintext in lower bits for evaluating arbitrary-depth circuits, which is the functionality of CKKS bootstrapping. CKKS bootstrapping is performed by homomorphically evaluating the approximated modular reduction function on coefficients of the plaintext [CHK+18a]. In contrast to BFV bootstrapping, the performance of CKKS bootstrapping is primarily influenced by precision, indicating how many upper bits of the plaintext m are preserved during bootstrapping.

1.1 Our Contributions

In this paper, we propose a novel BFV bootstrapping method that utilizes CKKS bootstrapping as a subroutine, departing from the previous digit extraction-based approach.

Incorporating CKKS Bootstrapping. Our key observation is that if the BFV plaintext modulus t divides the ciphertext modulus q, then the noise part of a BFV ciphertext can be extracted in the form of a CKKS ciphertext by switching the ciphertext modulus from q to Δ. To be precise, for a BFV ciphertext $(b, a) \in R_q^2$ satisfying $b + as = \Delta m + e \pmod{q}$, it holds that $[b]_\Delta + [a]_\Delta \cdot s = e \pmod{\Delta}$. We observe that the ciphertext $([b]_\Delta, [a]_\Delta) \in R_\Delta^2$ is no longer a BFV ciphertext, but can be interpreted as a CKKS ciphertext encrypting the noise e as a plaintext. We then process this ciphertext with CKKS operations such as CKKS bootstrapping. We recall that the functionality of CKKS bootstrapping involves raising the ciphertext modulus while approximately preserving the plaintext. Thus, by applying CKKS bootstrapping to the ciphertext encrypting the extracted noise, we can obtain a CKKS ciphertext that encrypts e' under modulus q, where $e' \approx e$. Finally, subtracting the bootstrapped ciphertext from the original ciphertext results in noise reduction from e to $e - e'$. We note that the performance of our bootstrapping method relies on the efficiency of CKKS bootstrapping, which is used as a subroutine. Thus, improvements in CKKS bootstrapping, such as algorithmic optimization [LLL+21, LLK+22, JM22, BCC+22], or hardware acceleration [JKA+21, KKK+22], directly lead to the enhancement of BFV bootstrapping, bridged by our method.

Flexible Plaintext Modulus. The performance dependency on CKKS bootstrapping in our method also provides flexibility in the choice of the plaintext modulus. We note that the plaintext modulus t corresponds to the modulus gap between the input and output ciphertext modulus in the CKKS bootstrapping subroutine, and only its scale affects the performance rather than the number-theoretic property. Thus, in our method, one can use the plaintext modulus as needed without considering its effect on bootstrapping performance. For example, our bootstrapping provides viable performance with a large prime plaintext modulus, which yields the worst bootstrapping performance with the previous approach. This helps in constructing BFV applications, where using a large prime plaintext modulus is crucial, such as in HE-based private set intersection protocols [CLR17, CHLR18, CMdG+21].

Optimized Circuit Evaluation. Another unique property of our bootstrapping method is its tunable performance, a capability not present in the previous method. In our approach, we can adjust the amount of reduced noise, which directly translates into the precision of the underlying CKKS bootstrapping. This leads to variations in bootstrapping performance, as CKKS bootstrapping is primarily affected by precision. Thus, with our method, we can optimize circuit evaluation by employing appropriate bootstrapping depending on circumstances, varying the amount of noise reduction.

Concrete Efficiency. We implemented our algorithm at a proof-of-concept level, and it outperforms the digit extraction-based bootstrapping method in terms of both latency and throughput. To achieve practical performance, we utilized the recent optimization in CKKS bootstrapping [BCC+22], called META-BTS, which supports efficient arbitrary-precision CKKS bootstrapping. When benchmarking the performance of bootstrapping with 51-bits plaintext moduli, our method outperformed the previous state-of-the-art BFV bootstrapping method [GIKV23] by a factor of 37.9 in latency and 29.4 in throughput (see Table 1). We attribute this result to the high-throughput SIMD operations of the CKKS scheme, whereas the existing solutions suffered from the inefficiency of digit extraction caused by the imbalance between the ring dimension and the number of plaintext slots.

Table 1. Bootstrapping performance comparison. Amortized bootstrapping time denotes the bootstrapping time divided by the number of coefficients.

	Plaintext modulus	Ring dimension	Reduced noise (bits)	Boot time (sec)	Amortized boot time (ms/coeff)
[GIKV23]	2^{51}	42336	131	**1344+**	**31.7+**
Ours	$\approx 2^{51}$	32768	128	**35.5**	**1.08**

1.2 Related Works

BFV Bootstrapping. Since the initial idea of digit extraction was proposed by Halevi and Shoup [HS15], a line of studies has been conducted to optimize its efficiency. Chen and Han [CH18] presented improved digit extraction when the plaintext modulus is a power of small primes, and its performance has been enhanced in subsequent work [GIKV23, OPP23]. The most relevant study that shares the same goal as ours in breaking performance dependency on the plaintext modulus is by Kim et al. [KDE+23]. They presented another way of BFV bootstrapping that leverages the bootstrapping procedure of the TFHE scheme [CGGI20]. However, since TFHE bootstrapping does not support SIMD-style operations, their method suffers from low throughput.

CKKS Bootstrapping. As our BFV bootstrapping method employs CKKS bootstrapping as a subroutine, development on the CKKS bootstrapping method greatly affects the performance of our method. Since the first instantiation of CKKS bootstrapping was accomplished by approximate homomorphic evaluation of the sine function by Cheon et al. [CHK+18a], a series of studies [LLL+21, LLK+22, JM22] have been conducted targeting HE-friendly approximation of the modular reduction function to improve precision metrics. Apart from these approaches, Bae et al. [BCC+22] recently presented a novel method called META-BTS, which achieves arbitrary precision CKKS bootstrapping by iteratively using low-precision CKKS bootstrapping. Since our bootstrapping internally utilizes CKKS bootstrapping, and the required precision is larger than ordinary CKKS use cases, our bootstrapping method benefits from the META-BTS bootstrapping technique.

2 Preliminaries

2.1 Notations

For an integer $M > 0$, we denote by $\Phi_M(X)$ the M-th cyclotomic polynomial, and write $N = \phi(M)$. When M is a power of two, we have $N = M/2$ and $\Phi_M(X) = X^N + 1$. We denote the ring of integers as $R = \mathbb{Z}[X]/(\Phi_M(X))$ and its residue ring modulo an integer q as $R_q = R/qR$. An element $a = \sum_{i=0}^{N-1} a_i X^i$ of R (or R_q) is often identified with the vector of its coefficients $(a_0, \ldots, a_{n-1})$ in $\mathbb{Z}^N$ (or $\mathbb{Z}_q^N$). We use $\mathbb{Z} \cap (-q/2, q/2]$ as a representative of $\mathbb{Z}_q$ for an integer q and denote $[a]_q$ as the reduction of each coefficient of $a \in R$ modulo q. For $a \in R$, we define $\|a\|_p$ as the ℓ^p-norm of its coefficient vector.

For a real number r, $\lfloor r \rceil$ denotes the nearest integer to r, rounding upwards in case of a tie. For a distribution $\mathcal{D}$, we use $x \leftarrow \mathcal{D}$ to denote sampling x according to $\mathcal{D}$. For a finite set S, we denote the uniform distribution over S as $\mathcal{U}(S)$.

2.2 Ring Learning With Errors

Let χ and ψ be distributions over R. The ring learning with errors (RLWE) assumption with respect to the parameter (R, q, χ, ψ) is that given polynomially many samples of either (b, a) or $(-as + e, a)$, where $a, b \leftarrow \mathcal{U}(R_q)$, $s \leftarrow \chi$, $e \leftarrow \psi$, it is computationally hard to distinguish which is the case. The security of lattice-based homomorphic encryption (HE) schemes such as BFV [Bra12, FV12], and CKKS [CKKS17] relies on the hardness of the RLWE assumption.

2.3 Homomorphic Encryption

In this section, we introduce two RLWE-based homomorphic encryption schemes, called BFV [Bra12, FV12] and CKKS [CKKS17]. The plaintext space of BFV is R_t for some plaintext modulus t, while the CKKS scheme [CKKS17] can encrypt elements of R. Both BFV and CKKS schemes support arithmetic operations as well as automorphism evaluation for the rotation of plaintext slots. Below, we will briefly describe setup, encryption, and decryption procedures for BFV and CKKS. We refer the reader to the original papers [Bra12, FV12, CKKS17] for further details on homomorphic operations, such as multiplication or automorphism.

It is noteworthy that the BFV and CKKS schemes can share most parameters and use the same secret, public, and evaluation keys. This property enables a seamless conversion between the two schemes, a feature that will be utilized in our bootstrapping pipeline.

- $\underline{\texttt{Setup}(1^\lambda)}$: Given a security parameter λ, Choose a ring dimension N and a ciphertext modulus q. In the case of BFV, choose a plaintext modulus t. Set a secret key distribution ψ and an error distribution χ over R. Output a parameter set $\mathsf{pp} = (N, q, t, \chi, \psi)$.

In this paper, we make an assumption that $t \mid q$ for simplicity. The scaling factor will be denoted by $\Delta := q/t \in \mathbb{Z}$.

- $\underline{\texttt{KeyGen}(\mathsf{pp})}$: Given a public parameter pp, sample $s \leftarrow \chi$, $a \leftarrow \mathcal{U}(R_q)$ and $e \leftarrow \psi$. Return the secret key $\mathsf{sk} = s$ and the public key $\mathsf{pk} = (b, a) \in R_q^2$ where $b = -as + e \pmod q$.

- $\underline{\texttt{BFV.Enc}_{\mathsf{pk}}(m)}$: Given a public key $\mathsf{pk} \in R_q^2$, and a plaintext $m \in R_t$, sample $z \leftarrow \chi$, $e_0, e_1 \leftarrow \psi$. Return the ciphertext $\mathsf{ct} = z \cdot \mathsf{pk} + (e_0 + \Delta m, e_1) \pmod q$.

- $\underline{\texttt{BFV.Dec}_{\mathsf{sk}}(\mathsf{ct})}$: Given a secret key $\mathsf{sk} = s \in R$ and a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_q^2$, output $m = \left\lfloor \frac{1}{\Delta} \cdot (c_0 + c_1 s) \right\rceil \pmod t$.

A BFV encryption of a plaintext $m \in R_t$ is the form of $\mathsf{ct} = (c_0, c_1) \in R_q^2$ that satisfies $c_0 + c_1 s = \Delta m + e \pmod q$ for some small error $e \in R$.

- $\underline{\texttt{CKKS.Enc}_{\mathsf{pk}}(m)}$: Given a public key $\mathsf{pk} \in R_q^2$, and a plaintext $m \in R$, sample $z \leftarrow \chi$, $e_0, e_1 \leftarrow \psi$. Return a ciphertext $\mathsf{ct} = z \cdot \mathsf{pk} + (e_0 + m, e_1) \pmod q$.

• $\underline{\texttt{CKKS.Dec}_{\mathsf{sk}}(\mathsf{ct}')}$: Given a secret key $\mathsf{sk} = s \in R^2$ and a ciphertext $\mathsf{ct}' = (c_0', c_1') \in R_{q'}^2$, output a plaintext $m' = c_0' + c_1' s \pmod{q'}$.

The CKKS scheme can encrypt an element $m \in R$ that is sufficiently smaller than the ciphertext modulus, i.e., $\|m\|_\infty \ll q$. A CKKS encryption of $m \in R$ takes the form $\mathsf{ct} = (c_0, c_1)$ which satisfies $c_0 + c_1 s = m + e \pmod{q}$, where e is a small noise. Therefore, the decryption of a CKKS encryption does not recover in the exact plaintext value m, but rather an approximate value $m' = m + e \approx m$. This approximate nature, which allows for small numerical errors, is considered acceptable within the context of CKKS. In addition, the ciphertext modulus is not fixed in CKKS, but it employs a rescaling algorithm to manage the growth of plaintexts during homomorphic computations.

Several studies have explored packing multiple messages into a single plaintext to enable SIMD-like operations in HE schemes. In the BFV scheme, an encoding map converts a d-dimensional vector over $\mathbb{Z}_t$ into an element of R_t via a ring homomorphism from $\mathbb{Z}_t^d$ to R_t. Here d denotes the number of slots, and the ratio N/d determines the throughput efficiency of the packing technique. One typical example is the HElib packing method in [HS15, GHS12] where the plaintext modulus is $t = p^r$ for some prime number p. To be precise, let $\mathrm{ord}_M(p)$ be the order of p modulo M and $d = N/\mathrm{ord}_M(p)$. Then, $\Phi_M(X)$ is factored into a product of d irreducible polynomials modulo t and there exists a ring isomorphism between R_t and E^d where $E = \mathrm{GR}(p^r, \mathrm{ord}_M(p))$ denotes the Galois ring of characteristic p^r. Hence, we can instantiate vectorized operations over $\mathbb{Z}_t^d$ using arithmetic over R_t by embedding $\mathbb{Z}_t$ into E. As a special case, when p is a prime satisfying $p = 1 \pmod{M}$, i.e., $\mathrm{ord}_M(p) = 1$, $\Phi_M(X)$ splits in $\mathbb{Z}_t[X]$, and $E \cong \mathbb{Z}_t$. Consequently, R_t is isomorphic to $\mathbb{Z}_t^N$, providing the maximum number of slots $d = N$.

On the other hand, the CKKS scheme can support SIMD-style arithmetic over $\mathbb{C}$. In a nutshell, a complex message vector can be encoded into a plaintext in R by leveraging the property that $\Phi_M(X)$ splits linearly over $\mathbb{C}$. Since every root of $\Phi_M(X)$ is paired with its conjugate, the number of slots is always $N/2$ and arithmetic over R emulates arithmetic over $\mathbb{C}^{N/2}$. For more details on the packing method, we refer to [CKKS17].

3 Review on BFV Bootstrapping

In this section, we present the basic functionality of BFV bootstrapping and review the previous instantiation methods for comparison with our method.

3.1 Basic Functionality

In the decryption procedure for a BFV ciphertext $\mathsf{ct} = (c_0, c_1) \in R_q^2$, we first compute the following formula for a secret key $\mathsf{sk} = s$.

$$c_0 + c_1 s = \Delta m + e \pmod{q}$$

Then, the result can be represented as two terms: the one that contains the plaintext m multiplied by the scaling factor Δ, and the other term e, which we call the noise or error of the ciphertext. In BFV, the size of the noise gradually increases after each homomorphic operation, However, for correct decryption, it is required that the size of the noise should be smaller than a certain bound, i.e., $\|e\|_\infty < \Delta/2$. Thus, to support the evaluation of an arbitrary circuit, one needs an apparatus that reduces the size of noise, which corresponds to the BFV bootstrapping procedure. Below, we describe its functionality together with an illustration in Fig. 1.

• $\underline{\texttt{BFV.Boot}(\mathsf{ct})}$: Given a ciphertext $\mathsf{ct} \in R_q^2$ with plaintext m and the noise e where $\|e\|_\infty < B_{\mathsf{in}}$, it outputs a ciphertext $\mathsf{ct}' \in R_q^2$ with plaintext m and the noise e' where $\|e'\|_\infty < B_{\mathsf{out}} \ll B_{\mathsf{in}}$.

The functionality of the BFV bootstrapping algorithm can be represented by the tuple $(q, t, B_{\mathsf{in}}, B_{\mathsf{out}})$, where q denotes the ciphertext modulus, t denotes the plaintext modulus, and B_{in} and B_{out} denote the upper bounds for the noise of the input and output ciphertexts, respectively. We define the quantity $\log_2(B_{\mathsf{in}}/B_{\mathsf{out}})$ as the denoising factor of BFV bootstrapping since it indicates how many upper bits of noise are removed through bootstrapping. In practice, B_{in} determines the maximum multiplicative depth from initial encryption, and the denoising factor determines the maximum multiplicative depth after bootstrapping.

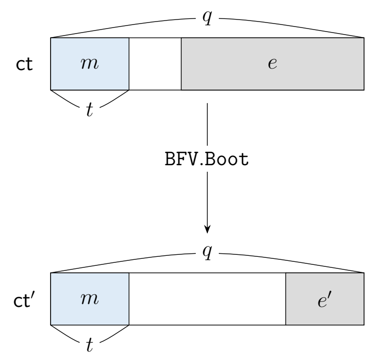


Fig. 1. Functionality of BFV Bootstrapping

3.2 Digit Extraction Framework

In this subsection, we review previous approaches to BFV bootstrapping [HS15, CH18, GIKV23, OPP23]. All these studies basically follow the so-called digit extraction framework by Halevi and Shoup [HS15], which operates on a plaintext modulus $t = p^r$ for some prime p. We illustrate its overall pipeline in Fig. 1.

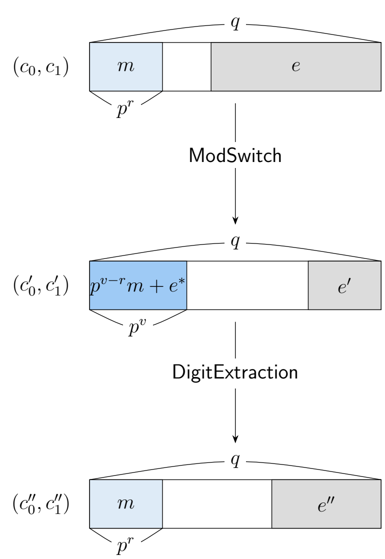


Fig. 2. Previous BFV Bootstrapping Pipeline

Let $(c_0, c_1) \in R_q^2$ be a BFV ciphertext with a plaintext $m \in R_{p^r}$ and noise e bounded by B_{in} so that $c_0 + c_1 s = \left\lfloor \frac{q}{p^r} \right\rceil m + e \pmod{q}$. The bootstrapping procedure starts with a modulus-switching procedure which reduces the ciphertext modulus from q to p^v for some v, and then restores it back to q. To be precise, it first computes $(c_0^*, c_1^*) \leftarrow (\lfloor (p^v/q) \cdot c_0 \rceil, \lfloor (p^v/q) \cdot c_1 \rceil) \in R_{p^v}^2$ that satisfies $c_0^* + c_1^* s = p^{v-r} m + e^* \pmod{p^v}$ for some error $e^* \approx (p^r/q)e$, then generates a new BFV ciphertext $(c_0', c_1') \leftarrow (\lfloor (q/p^v) \cdot c_0^* \rceil, \lfloor (q/p^v) \cdot c_1^* \rceil) \in R_q^2$ such that $c_0' + c_1' s = \left\lfloor \frac{q}{p^v} \right\rceil (p^{v-r} m + e^*) + e' \pmod{q}$ for a small rounding error e'. Here (c_0', c_1') is regarded as a BFV ciphertext encrypting $m' := p^{v-r} m + e^*$ with the plaintext modulus p^v.

Then, digit extraction is performed on this ciphertext, which homomorphically removes the $v - r$ least significant digits e^* of the plaintext of $m' = p^{v-r} m + e^*$ in base-p representation. Hence, the resulting ciphertext $(c_0'', c_1'') \in R_q^2$ is an encryption of $p^{v-r}m \in R_{p^v}$ with some noise e'' bounded by B_{out}, i.e., $c_0'' + c_1''s = \left\lfloor \frac{q}{p^v} \right\rceil p^{v-r}m + e'' \pmod q$. Since $\left\lfloor \frac{q}{p^v} \right\rceil \cdot p^{v-r}m \approx \left\lfloor \frac{q}{p^r} \right\rceil m$, this ciphertext can be reinterpreted as a BFV encryption of $m \in R_{p^r}$ with noise $\approx e''$, as desired. This achieves a BFV bootstrapping with functionality $(q, t, B_{\mathsf{in}}, B_{\mathsf{out}})$.

Digit Extraction. The previous studies primarily aimed to improve the efficiency of digit extraction which is the major performance bottleneck in overall BFV bootstrapping. The digit extraction procedure is accomplished by evaluating a function that maps $x \in \mathbb{Z}_{p^v}$ to $\lfloor x/p^{v-r} \rceil \in \mathbb{Z}_{p^r}$, which can be represented as a composition of various operations supported by the BFV scheme. Halevi and Shoup [HS15] proposed a baseline solution where digit extraction can be achieved by the evaluation of a polynomial of degree p^{v-1}. A subsequent work by Chen and Han [CH18] presented a better polynomial representation of rp^{v-r}, which significantly improved the performance of digit extraction especially when p is a small prime. Currently, the best-known complexity for the evaluation of the digit extraction polynomial is $O(\sqrt{p}\sqrt[4]{r})$ in [GIKV23]. We refer to [GV23] for a more detailed analysis and discussion on digit extraction polynomial and its evaluation.

On the one hand, there is another important factor that affects the total complexity of digit extraction. The evaluation of a digit extraction polynomial utilizes the SIMD operations of BFV which can process at most d elements of $\mathbb{Z}_{p^v}$ at once. As a result, it requires $N/d = \mathrm{ord}_M(p)$ polynomial evaluations to complete the digit extraction procedure for the N coefficient of a plaintext. In particular, this extra factor $\mathrm{ord}_M(p)$ is large when $p \ll M$ is a small prime.

Functionality Analysis. In the perspective of BFV bootstrapping functionality, the digit extraction-based approach produces a constant size of output noise bound B_{out} once t is fixed, as it is determined by the degree of the underlying digit extraction polynomial. Also, its performance is independent of the input noise bound B_{in} since the noise e' after modulus switching is not affected by the input noise bound. Therefore, to achieve the maximum denoising factor, it is usually set to the maximum value that supports the correctness of modulus switching. To sum up, in the previous method, once q and t are determined, B_{in} and B_{out} are automatically decided. Since the choice of q only affects the security level, the choice of t, especially its number-theoretic properties, determines the overall performance of bootstrapping after all.

4 Review on CKKS Bootstrapping

In this section, we present the basic functionality of CKKS bootstrapping and revisit the current state-of-the-art method for instantiating it, especially the META-BTS [BCC+22] method, which is the core building block in our BFV bootstrapping method.

4.1 Basic Functionality.

In the decryption procedure for a CKKS ciphertext $\mathsf{ct} = (c_0, c_1) \in R_q^2$, we first compute the following formula for a secret key $\mathsf{sk} = s$.

$$c_0 + c_1 s = m \pmod q$$

As observed in the above decryption procedure, there is no strict distinction between plaintext and noise for CKKS ciphertexts, unlike BFV ciphertexts. Therefore, for correct decryption, the only requirement is that the size of the plaintext is smaller than the ciphertext modulus, i.e., $\|m\|_\infty < q$. However, in CKKS, the ciphertext modulus decreases after each homomorphic evaluation due to the rescaling procedures. Thus, to support the evaluation of arbitrary circuits, an apparatus is required that increases the ciphertext modulus while keeping the plaintext to a similar value, which is the exact functionality of CKKS bootstrapping. Below, we describe this functionality more precisely together with an illustration in Fig. 3.

• $\underline{\texttt{CKKS.Boot}(\mathsf{ct})}$: Given a ciphertext $\mathsf{ct} \in R_{q_{\mathsf{in}}}^2$ with a plaintext m whose size is bounded by $\|m\|_\infty < B_{\mathsf{pt}}$, it outputs a ciphertext $\mathsf{ct}' \in R_{q_{\mathsf{out}}}^2$ with a plaintext m' such that $\|m' - m\|_\infty < B_{\mathsf{err}}$ and the ciphertext modulus $q_{\mathsf{out}} > q_{\mathsf{in}}$.

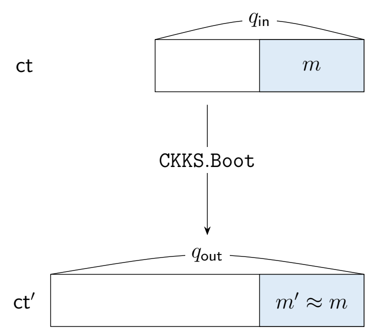


Fig. 3. Functionality of CKKS Bootstrapping

The functionality of CKKS bootstrapping can be parameterized as the tuple $(q_{\mathsf{in}}, q_{\mathsf{out}}, B_{\mathsf{pt}}, B_{\mathsf{err}})$, where q_{in} and q_{out} denote the modulus of the input and output ciphertexts respectively, B_{pt} denotes the upper bound for the plaintext of the input ciphertext, and B_{err} denotes the upper bound for differences between input and output plaintexts. We also refer to the quantity $\log_2(B_{\mathsf{pt}}/B_{\mathsf{err}})$ as the precision of CKKS bootstrapping since it indicates how many upper bits of the input plaintext are preserved during bootstrapping.

4.2 The Base CKKS Bootstrapping

The first instantiation of CKKS bootstrapping is accomplished by Cheon et al. [CHK+18a]. The key idea of their work is homomorphically evaluating the modular reduction function $x \mapsto [x]_{q_{\mathsf{in}}}$ on coefficients of the plaintext in an approximate manner, where q_{in} is the modulus of input ciphertexts.

Given a CKKS ciphertext $(c_0, c_1) \in R_{q_{\mathsf{in}}}^2$ encrypting a plaintext $m \in R$ satisfying $\|m\|_\infty < B_{\mathsf{pt}}$, it first performs a modulus raising operation, which yields a CKKS ciphertext $(c_0', c_1') \in R_q^2$ encrypting a plaintext $m + q_{\mathsf{in}} I$ for some polynomial I. Then, it homomorphically evaluates an approximation of the modular reduction function resulting in a ciphertext $(c_0'', c_1'') \in R_{q_{\mathsf{out}}}^2$ encrypting a plaintext m' satisfying $\|m - m'\|_\infty < B_{\mathsf{err}}$. Since the output ciphertext modulus q_{out} is set to be greater than the input ciphertext modulus q_{in}, it provides CKKS bootstrapping functionality $(q_{\mathsf{in}}, q_{\mathsf{out}}, B_{\mathsf{pt}}, B_{\mathsf{err}})$.

Modular Reduction As the modular reduction function is not a polynomial function, finding a precise polynomial approximation of it has been the main research topic. Since the initial instantiation by Cheon et al. [CHK+18a] is done by evaluating a polynomial approximation of the sine function, a series of studies[LLL+21, JM22, LLK+22] has aimed at enhancing its precision by finding HE-friendly polynomial approximations of the modular reduction function. We refer to this type of bootstrapping instantiation as a base bootstrapping, `CKKS.BaseBoot`. Given a base CKKS bootstrapping algorithm `CKKS.BaseBoot` with functionality $(q_{\mathsf{in}}, q_{\mathsf{out}}, B_{\mathsf{pt}}, B_{\mathsf{err}})$, its time complexity is dominated by the precision factor $\log_2(B_{\mathsf{pt}}/B_{\mathsf{err}})$. To be precise, for achieving x-bits precision, it evaluates a polynomial approximation of the modular reduction function whose degree follows $O(\sqrt{x})$ and its time complexity roughly follows some super-linear function $T(x)$ according to the analysis in [BCC+22]. We also note that it suffices to evaluate the approximated modular reduction function twice since the number of slots in the CKKS scheme is $N/2$, compared to the digit extraction procedure of the BFV bootstrapping.

4.3 META-BTS: Bootstrapping for Arbitrary Precision

In this subsection, we review the META-BTS bootstrapping method, which we utilize as a core building block for our BFV bootstrapping method. Apart from the previous approaches on CKKS bootstrapping, Bae et al. [BCC+22] presented a novel method called META-BTS. Its core idea is to iteratively employ a low-precision base bootstrapping algorithm to attain a high-precision bootstrapping algorithm. Their main result is as follows.[3]

Theorem 1 (Thm 3.2 [BCC+22]). *Given a base CKKS bootstrapping algorithm* `CKKS.BaseBoot` *with functionality* $(q_{\mathsf{in}}, q_{\mathsf{out}}, B_{\mathsf{pt}}, B_{\mathsf{err}})$ *and* $n = \log_2(B_{\mathsf{pt}}/B_{\mathsf{err}})$*-bits precision, one can construct a new bootstrapping algorithm* $\texttt{CKKS.Boot}^{(k)}$ *with functionality* $(q_{\mathsf{in}} \cdot 2^{(k-1)n}, q_{\mathsf{out}}, B_{\mathsf{pt}} \cdot 2^{(k-1)n}, B_{\mathsf{err}})$ *and* kn*-bits precision by repeating* `CKKS.BaseBoot` k *times.*

The advantage of the META-BTS algorithm is twofold: firstly, it provides asymptotically faster time complexity when achieving the same bootstrapping functionality. To be precise, suppose that the target CKKS bootstrapping functionality $(q_{\mathsf{in}}, q_{\mathsf{out}}, B_{\mathsf{pt}}, B_{\mathsf{err}})$ requires kn-bits precision. If we directly instantiate it with the `CKKS.BaseBoot` so that it supports the designated functionality and precision, it takes asymptotically $T(kn)$ time complexity. In contrast, for the META-BTS bootstrapping, it can run k iterations of n-bits precision `CKKS.BaseBoot`, which supports the functionality $(q_{\mathsf{in}}/2^{(k-1)n}, q_{\mathsf{out}}, B_{\mathsf{pt}}/2^{(k-1)n}, B_{\mathsf{err}})$. Then, it yields $k \cdot T(n)$ time complexity, which is reduced from $T(kn)$ since T is a super-linear function. Thus, the META-BTS method reduces asymptotic complexity for attaining the same bootstrapping functionality. Secondly, it provides convenience in adjusting precision. With the base bootstrapping algorithm, one needs to recalculate all the parameters for `CKKS.BaseBoot` if precision changes. In contrast, the META-BTS algorithm $\texttt{CKKS.Boot}^{(k)}$ can adjust precision by simply modifying the iteration number k without altering parameters for the base bootstrapping. For a more detailed analysis, we refer to [BCC+22].

5 New BFV Bootstrapping

In this section, we present a novel BFV bootstrapping method with enhanced simplicity and efficiency. In particular, our method mitigates the inefficiency issues in the existing approach that stems from the heavy dependence of digit extraction on the plaintext modulus. Hence, we offer better flexibility in the selection of plaintext modulus with minimal difference in performance or implementation.

5.1 Our Bootstrapping Pipeline

We redesign the BFV bootstrapping procedure and introduce an entirely new pipeline below. Our method no longer relies on the previous digit extraction framework; instead, it incorporates the CKKS bootstrapping as a fundamental building block. Our bootstrapping pipeline consists of three steps: noise extraction, approximate lifting, and subtraction. The whole pipeline is illustrated in Fig. 4.

Setup: For simplicity, we assume that the plaintext modulus t divides the ciphertext modulus q, and thus $\Delta = q/t$. Suppose we are given as input a noisy BFV encryption $\mathsf{ct} = (c_0, c_1) \in R_q^2$ of $m \in R_t$ under secret $s \in R$. That is, it satisfies

$$c_0 + c_1 s = \Delta m + e \pmod{q} \tag{1}$$

for a large error $e \in R$ with a bound $\|e\|_\infty < B_{\mathsf{in}}$.

Noise Extraction: In the first step, we perform a modular reduction to extract the error term from the input ciphertext. Specifically, we compute $\hat{\mathsf{ct}} = (\hat{c}_0, \hat{c}_1) \in R_\Delta^2$ by $\hat{c}_0 = c_0 \pmod{\Delta}$ and $\hat{c}_1 = c_1 \pmod{\Delta}$, then it holds that

$$\hat{c}_0 + \hat{c}_1 s = e \pmod{\Delta}. \tag{2}$$

from Eq. (1).

It is worth noting that the resulting ciphertext $\hat{\mathsf{ct}} = (\hat{c}_0, \hat{c}_1)$ is no longer a valid BFV encryption. Instead, it can be regarded as an RLWE instance whose error term is $e \in R$.

[3] We modified the original statement, excluding the concept of the scaling factor.

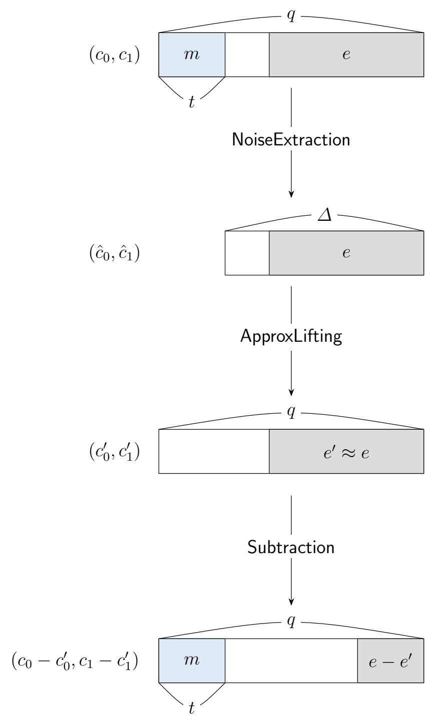


Fig. 4. Our BFV Bootstrapping Pipeline

Approximate Lifting: The next step lies at the heart of our solution where the actual bootstrapping takes place. We aim to raise the ciphertext modulus of $\hat{\mathsf{ct}}$ from Δ to q and generate a new ciphertext $\mathsf{ct}' = (c'_0, c'_1) \in R_q$ such that $c'_0 + c'_1 s \approx e \pmod q$.

To achieve this goal, we first make an observation that the CKKS scheme can be used, instead of BFV, to manipulate the ciphertext from the previous step. In other words, the ciphertext $\hat{\mathsf{ct}} = (\hat{c}_0, \hat{c}_1) \in R_\Delta^2$ can be reinterpreted as a CKKS ciphertext, with $e \in R$ representing the underlying plaintext bounded by B_{in}.

Then, we employ the CKKS bootstrapping, which precisely provides the required functionality for ciphertext modulus lifting. If a CKKS bootstrapping algorithm $\texttt{CKKS.Boot}(\cdot)$ has a functionality parameter $(q_{\mathsf{in}}, q_{\mathsf{out}}, B_{\mathsf{pt}}, B_{\mathsf{err}}) = (\Delta, q, B_{\mathsf{in}}, B_{\mathsf{out}})$, then the resulting ciphertext $\mathsf{ct}' = (c'_0, c'_1) \in R_q^2$ obtained from $\hat{\mathsf{ct}}$ by executing the CKKS bootstrapping algorithm will satisfy $c'_0 + c'_1 s = e' \pmod q$ for some $e' \in R$ such that $\|e - e'\|_\infty < B_{\mathsf{out}}$.

Subtraction: Finally, we revert to the BFV scheme to complete the bootstrapping process. We remark that the resulting ciphertext $\mathsf{ct}' = (c'_0, c'_1) \in R_q^2$ from the previous CKKS bootstrapping step holds $c'_0 + c'_1 \cdot s = e' \pmod q$, allowing it to be interpreted as a BFV encryption of zero with the noise e'.

Subtracting it from the original ciphertext, we get $\mathsf{ct}'' \leftarrow \mathsf{ct} - \mathsf{ct}' \in R_q^2$, which is a valid BFV encryption of m. Notably, its noise $e - e'$ has a reduced upper bound B_{out} compared to the initial noise e bounded by B_{in}. This concludes our new BFV bootstrapping procedure.

We provide an algorithmic description of our bootstrapping method in Alg. 1 and prove its correctness in Thm. 2.

Theorem 2. *Let* $\texttt{CKKS.Boot}(\cdot)$ *denote a CKKS algorithm with functionality parameters* $(\Delta, q, B_{\mathsf{in}}, B_{\mathsf{out}})$. *Then, the* $\texttt{BFV.Boot}(\cdot)$ *algorithm (Alg. 1) realizes the BFV bootstrapping with functionality parameters* $(q, t, B_{\mathsf{in}}, B_{\mathsf{out}})$.

Proof. Let $\mathsf{ct} = (c_0, c_1) \in R_q^2$ be a BFV encryption of $m \in R_t$ under a secret key s and let $e \in R$ denote the noise of ct bounded by B_{in}, i.e., $c_0 + c_1 s = \Delta m + e \pmod q$ with $\|e\|_\infty < B_{\mathsf{in}}$.

Algorithm 1 `BFV.Boot`

Input: A BFV ciphertext $\mathsf{ct} = (c_0, c_1) \in R_q^2$
Output: A BFV ciphertext $\mathsf{ct}'' = (c_0'', c_1'') \in R_q^2$
1: $\hat{\mathsf{ct}} = (\hat{c}_0, \hat{c}_1) \leftarrow \mathsf{ct} \pmod{\Delta}$ ▷ Noise Extraction
2: $\mathsf{ct}' = (c_0', c_1') \leftarrow \texttt{CKKS.Boot}(\hat{\mathsf{ct}})$ ▷ Approximate Lifting
3: $\mathsf{ct}'' = (c_0'', c_1'') \leftarrow \mathsf{ct} - \mathsf{ct}' \pmod{q}$ ▷ Subtraction
4: **return** $\mathsf{ct}'' = (c_0'', c_1'') \in R_q^2$

From the definition, it is obvious that $\hat{c}_0 + \hat{c}_1 s = e \pmod{\Delta}$. Subsequently, the ciphertext $\mathsf{ct}' = (c_0', c_1') \in R_q^2$ derived from approximate lifting satisfies $c_0' + c_1' s = e' \pmod{q}$ for some e' with $\|e - e'\|_\infty < B_{\mathsf{out}}$ from the functionality of $\texttt{CKKS.Boot}(\cdot)$. Finally, the output ciphertext $\mathsf{ct}'' = (c_0'', c_1'') \leftarrow \mathsf{ct} - \mathsf{ct}' \pmod{q}$ is a BFV encryption of m and its noise is bounded by B_{out} since $c_0'' + c_1'' s = \Delta m + e'' \pmod{q}$ for $e'' = e - e'$.

Therefore, our $\texttt{BFV.Boot}(\cdot)$ algorithm instantiates the functionality of BFV bootstrapping with parameters $(q, t, B_{\mathsf{in}}, B_{\mathsf{out}})$.

Lastly, we show that our method is applicable to a general case when t may not divide q. In short, we can address the issue by incorporating simple pre- and post-processings steps into our pipeline. If $t \nmid q$, we switch the ciphertext modulus between q and q' before and after bootstrapping for some integer q' such that $q' \approx q$ and $t \mid q'$. For a BFV ciphertext $\mathsf{ct} = (c_0, c_1) \in R_q^2$ satisfying $c_0 + c_1 s = \Delta m + e \pmod{q}$ for $\Delta = \lfloor q/t \rceil$, the modulus switching from q to q' results in a new ciphertext $(\lfloor (q'/q) \cdot c_0 \rceil, \lfloor (q'/q) \cdot c_1 \rceil) \in R_{q'}^2$ such that $\lfloor (q'/q) \cdot c_0 \rceil + \lfloor (q'/q) \cdot c_1 \rceil \cdot s \approx \Delta' m + \lfloor (q'/q) \cdot e \rceil \pmod{q'}$ for $\Delta' = q'/t$. Hence, if our solution provides the functionality of BFV bootstrapping with parameters $(q', t, B_{\mathsf{in}}', B_{\mathsf{out}}')$, it can be naturally extended to a BFV bootstrapping with new functionality parameters $(q, t, B_{\mathsf{in}}, B_{\mathsf{out}})$ with $B_{\mathsf{in}} \approx (q/q') \cdot B_{\mathsf{in}}'$ and $B_{\mathsf{out}} \approx (q/q') \cdot B_{\mathsf{out}}'$ by combining modulus switchings between q and q'.

5.2 Approximate Lifting from META-BTS

Although our proposed pipeline is simple, nonetheless, implementing the approximate lifting step remains a challenging problem. Recall that we aim to transform a ciphertext $\hat{\mathsf{ct}} = (\hat{c}_0, \hat{c}_1) \in R_\Delta^2$ such that $\hat{c}_0 + \hat{c}_1 \cdot s = e \pmod{\Delta}$ into a new ciphertext $\mathsf{ct}' = (c_0', c_1') \in R_q^2$ satisfying $c_0' + c_1' \cdot s \approx e \pmod{\Delta}$ using a CKKS bootstrapping algorithm. Here e is natively the error term of a noisy BFV ciphertext which is often several hundred bits long, whereas the precision of CKKS bootstrapping algorithms is usually limited to dozens of bits. A straightforward solution might involve improving the precision of CKKS bootstrapping using a more accurate approximation of modulo reduction function in the previous studies [CHK+18a, LLL+21, LLK+22, JM22]. However, this approach is likely to yield impractical results since the performance of CKKS bootstrapping deteriorates rapidly in terms of both complexity and modulus consumption as the precision of CKKS operations increases.

Recently, META-BTS [BCC+22] proposed a new mechanism that achieves a high-precision CKKS bootstrapping based only on low-precision operations. We realize that the same idea can be applied to our approximate lifting procedure without having significant performance overhead by modifying some internal parameters. In addition, the iterative nature of this method introduces a new feature to our bootstrapping procedure, enabling flexible adjustment of the denoising factor. Due to these advantages, we utilize the META-BTS method in the instantiation of approximate lifting. In the rest of this subsection, we provide detailed information on the META-BTS method, parameter selection, and its implications for our bootstrapping pipeline.

Performance Improvements. Assume that we want to achieve a BFV bootstrapping with functionality parameters $(q, t, B_{\mathsf{in}}, B_{\mathsf{out}})$ with the denoising factor $kn = \log_2(B_{\mathsf{in}}/B_{\mathsf{out}})$ bits. For the approximate lifting, we utilize a CKKS bootstrapping with the functionality $(\Delta, q, B_{\mathsf{in}}, B_{\mathsf{out}})$, where the denoising factor directly translates into the precision of the CKKS bootstrapping. Thus, we need to instantiate a CKKS bootstrapping with kn-bits precision. We first recall that the time complexity $T(x)$ in achieving x-bits precision via a base CKKS bootstrapping grows super-linearly. If we directly achieves kn-bits precision by single iteration of base CKKS bootstrapping, it takes $T(kn)$ complexity. On the other hand, if we apply the META-BTS method, which iterates the n-bits base CKKS bootstrapping `CKKS.BaseBoot` k times, the functionality of `CKKS.BaseBoot` is determined as $(\Delta/2^{(k-1)n}, q, B_{\mathsf{in}}/2^{(k-1)n}, B_{\mathsf{out}})$, and the time complexity is $k \cdot T(n)$. Therefore, employing the META-BTS method reduces the time complexity from $T(kn)$ to $k \cdot T(n)$ as T is a super-linear function. Since our method mainly handles large precisions kn, the performance improvement is more significant.

Adjustable Functionality. We recall that another important aspect of META-BTS is its ability to adjust precision. We show how this affects our BFV bootstrapping, resulting in the ability to adjust bootstrapping functionality. Before stating it more clearly, we provide some useful properties of CKKS bootstrapping below.

Lemma 1. *Given a CKKS bootstrapping algorithm* `CKKS.Boot` *with functionality* $(q_{\mathsf{in}}, q_{\mathsf{out}}, B_{\mathsf{in}}, B_{\mathsf{out}})$. *Then for any positive integer* q'*, one can instantiate the CKKS bootstrapping algorithms with the following functionalities:*

1. $(q' \cdot q_{\mathsf{in}}, q_{\mathsf{out}}, B_{\mathsf{in}}, B_{\mathsf{out}})$
2. $(q_{\mathsf{in}}, q_{\mathsf{out}}/q', B_{\mathsf{in}}, B_{\mathsf{out}})$ *if* $q' | q_{\mathsf{out}}$
3. $(q' \cdot q_{\mathsf{in}}, q' \cdot q_{\mathsf{out}}, q' \cdot B_{\mathsf{in}}, q' \cdot (B_{\mathsf{out}} + \frac{\|s\|_1+1}{2}))$ *if* $\|s\|_1 \ll B_{\mathsf{in}}$

by running `CKKS.Boot` *a single time.*

Proof. The first and second functionalities can be easily instantiated by taking modulo q and q/q' operations on input and output ciphertexts of `CKKS.Boot` respectively. For the last functionality, let $(c_0, c_1) \in R^2_{q' q_{\mathsf{in}}}$ be an input ciphertext. Then, we can instantiate the last functionality as follows:

Step 1. Run `CKKS.Boot` on $(\lfloor c_0/q' \rceil, \lfloor c_1/q' \rceil) \in R^2_{q_{\mathsf{in}}}$ and obtain the result $(c_0', c_1') \in R^2_{q_{\mathsf{out}}}$.
Step 2. Output $(q' \cdot [c_0']_{q_{\mathsf{out}}}, q' \cdot [c_1']_{q_{\mathsf{out}}}) \in R^2_{q' q_{\mathsf{out}}}$.

If we set $c_0 + c_1 s = m \pmod{q' q_{\mathsf{in}}}$, then $\lfloor c_0/q' \rceil + \lfloor c_1/q' \rceil s = m/q' + e \pmod{q_{\mathsf{in}}}$ holds where e is a rounding noise satisfying $\|e\|_\infty < \frac{\|s\|_1+1}{2}$. Since we assume $\|s\|_1 \ll B_{\mathsf{in}}$, running `CKKS.Boot` with $(\lfloor c_0/q' \rceil, \lfloor c_1/q' \rceil)$ yields the ciphertext $(c_0', c_1') \in R^2_{q_{\mathsf{out}}}$ such that $c_0' + c_1' s = m/q' + e + e' \pmod{q_{\mathsf{out}}}$ with a bootstrapping noise e' satisfying $\|e'\|_\infty < B_{\mathsf{out}}$. Finally, multiplying q' results in $q'[c_0']_{q_{\mathsf{out}}} + q'[c_1']_{q_{\mathsf{out}}} s = m + q'(e + e') \pmod{q' q_{\mathsf{out}}}$. Since $\|q'(e+e')\|_\infty < q' \cdot (B_{\mathsf{out}} + \frac{\|s\|_1+1}{2})$, we instantiate the last functionality.

Applying the above lemma to the base CKKS bootstrapping of the functionality $(\Delta/2^{(k-1)n}, q, B_{\mathsf{in}}/2^{(k-1)n}, B_{\mathsf{out}})$ results in the following theorem.

Theorem 3. *Let* $(\Delta/2^{(k-1)n}, q, B_{\mathsf{in}}/2^{(k-1)n}, B_{\mathsf{out}})$ *be the functionality of the base CKKS bootstrapping* `CKKS.BaseBoot`*. If* $\|s\|_1 \ll B_{\mathsf{in}}$*, one can instantiate CKKS bootstrapping with functionality* $(\Delta, q, B_{\mathsf{in}}/2^{an}, (B_{\mathsf{out}} + \frac{\|s\|_1+1}{2}) \cdot 2^{bn})$ *by iterating* `CKKS.BaseBoot` *with* $k - a - b$ *times where* a, b *are non-negative integers satisfying* $a + b < k$.

Proof. Firstly, one can instantiate a CKKS bootstrapping with functionality $(\Delta/2^{(a+b)n}, q, B_{\mathsf{in}}/2^{(a+b)n}, B_{\mathsf{out}})$ by iteratively running `CKKS.BaseBoot` $k - a - b$ times by Theorem 1. Next, we apply the third property in Lem.1 with $q' = 2^{bn}$, then it yields a CKKS bootstrapping with functionality $(\Delta/2^{an}, q \cdot 2^{bn}, B_{\mathsf{in}}/2^{an}, (B_{\mathsf{out}} + \frac{\|s\|_1+1}{2}) \cdot 2^{bn})$ assuming $h \ll B_{\mathsf{out}}$. Finally, applying the first and the second property in Lem. 1 results in a CKKS bootstrapping with functionality $(\Delta, q, B_{\mathsf{in}}/2^{an}, (B_{\mathsf{out}} + \frac{\|s\|_1+1}{2}) \cdot 2^{bn})$.

The above theorem directly implies that given the base CKKS boostrapping, we can also instantiate another BFV bootstarpping with the functionality $(q, t, B_{\mathsf{in}}/2^{an}, (B_{\mathsf{out}} + \frac{\|s\|_1+1}{2}) \cdot 2^{bn})$, and the iteration count is determined by the denoising factor $\approx 2^{(a+b)n}$, which reduces overall time complexity by a factor $k/(k-a-b)$. Hence, in our bootstrapping, the META-BTS method provides the ability to decrease the input noise bound or increase the output noise bounds, and in that case, the performance of bootstrapping gets improved according to the reduced denoising factor.

5.3 Analysis

We discuss our bootstrapping method for achieving the target BFV bootstrapping functionality in comparison with the previous digit extraction based approaches. In the prior work, the performance of BFV bootstrapping was heavily dependent on the number-theoretic property of the plaintext modulus t. Assuming $t = p^r$ for some prime number p, it requires the evaluation of a digit extraction polynomial whose degree is determined by p and r. Hence, it achieves both reasonable performance and acceptable noise bound only when p is a small prime number. Additionally, the polynomial evaluation had to be repeated by $\mathrm{ord}_M(p)$ times due to the limited number of plaintext slots.

On the contrary, the denoising factor $\log_2(B_{\mathsf{in}}/B_{\mathsf{out}})$ is the parimary parameter that mainly affect the performance of our bootstrapping method. Recall that to achieve the given BFV bootstrapping functionality of parameter $(q, t, B_{\mathsf{in}}, B_{\mathsf{out}})$, we require a CKKS bootstrapping with the functionality $(\Delta, q, B_{\mathsf{in}}, B_{\mathsf{out}})$, and the performance of CKKS bootstrapping is typically determined by its precision. Since the denoising factor of BFV bootstrapping directly translates into the precision parameter for the CKKS bootstrapping, the performance of our bootstrapping method highly depends on the denoising factor, which is determined by the input and output noise upper bounds. On the other hand, the plaintext modulus is only related to some parameters of CKKS bootstrapping, so has little impact on the overall performance. Moreover, we can always exploit high-throughput SIMD operations of the CKKS scheme, regardless of t. Therefore, our method offers the flexibility to choose an arbitrary plaintext modulus without much consideration for its impact on bootstrapping performance. This flexibility may lead to the adoption of our technique in a wide range of BFV applications such as HE-based private set intersection protocols [CLR17, CHLR18, CMdG$^+$21], where using a large prime plaintext modulus is crucial.

Another distinctive feature of our method is its ability to adjust bootstrapping functionality. Leveraging the property of META-BTS, one can decrease the input noise bound or increase the output noise bound, leading to performance enhancement based on the reduced denoising factor. This offers further optimization in circuit evaluation. For example, when evaluating circuits where the required depth after bootstrapping is small, one can benefit by setting B_{out} large, resulting in enhanced bootstrapping performance proportional to the reduced denoising factor. However, such an optimization strategy is impossible with the previous method since the evaluation of the digit extraction polynomial provides a fixed output noise bound, and there is no performance gain in adjusting the input noise bound. Therefore, our bootstrapping method provides room for further optimization in BFV applications by offering tunable bootstrapping functionality.

6 Experimental Results

We present a proof-of-concept level implementation to demonstrate the performance of our bootstrapping method. Our code is developed using the C++ HEaaN library [Cry23]. The experiments were conducted on an Intel Xeon Gold 6242 at 2.8GHz with 503GiB of RAM running Linux in a single-threaded environment. In our implementation, we set the key distribution χ to be a sparse ternary distribution with a hamming weight h, and the error distribution ψ as a discrete Gaussian distribution with a standard deviation of 3.2. The security level of each bootstrapping parameter is measured by the lattice estimator [APS15] and set to achieve a 128-bit security level.

In the rest of this section, we first describe the optimization techniques employed in our implementation. Subsequently, we present the concrete performance of our method along with benchmark results. The benchmark results cover three aspects: firstly, we compare the bootstrapping performance between our method and the digit extraction-based one to illustrate the performance improvement of our method in achieving similar BFV bootstrapping functionality. Next, we provide timing results for various plaintext modulus sizes to demonstrate the flexibility in choosing the plaintext modulus. Finally, we measure timing results for various input and output noise bounds to highlight another unique property of our bootstrapping method, where we can adjust bootstrapping functionality depending on the scenarios.

6.1 Optimization Techniques

RNS Represetation. When implementing RLWE-based homomorphic encryption schemes such as CKKS and BFV, one needs to instantiate arithmetic over R_q with a large modulus q. Introducing big integer operations yields additional computational overheads; therefore, the Residue Number System (RNS)-based instantiation [CHK+18b, HPS19] has been popularly deployed due to its efficiency. In a nutshell, RNS representation exploits the algebraic isomorphism between R_q and $R_{q_1} \times \cdots \times R_{q_\ell}$ when $q = q_1 \ldots q_\ell$ and q_i's are pairwise coprime. Then, operations over R_q can be instantiated with operations over R_{q_i}'s, and since q_i's are usually set to be word-size, it can be efficiently implemented without introducing big integer arithmetic. The HEaaN library we used is also implemented in a full RNS manner, i.e., no external big integer library is used.

Sparse-secret Encapsulation. When other parameters are fixed, the performance of CKKS bootstrapping is affected by the Hamming weight h of the secret key distribution. Previously, there existed a trade-off relation between the performance of bootstrapping and the security level. Specifically, a low Hamming weight h provides efficient bootstrapping performance but a low security level, and vice versa. Hence, it was usually set to a middle ground that yields both fair performance and an acceptable security level. Recently, Bossuat et al. [BTPH22] introduced an optimization called the sparse-secret encapsulation technique, which resolves this trade-off relation. At a high level, it leverages the nature of the CKKS scheme where the ciphertext modulus keeps decreasing until bootstrapping is applied. Initially, ciphertexts are encrypted with a dense secret key at a large ciphertext modulus, providing a high-security level. However, for input ciphertexts for bootstrapping, their modulus is usually smaller than that of fresh ciphertexts due to homomorphic evaluations, and a sparse secret key provides a good security level at this reduced ciphertext modulus. Thus, the proposed optimization technique first changes the input ciphertext's secret key with a sparse secret key before the modulus raising step and switches the output ciphertext's secret key back to the original dense one, and then it performs the remaining bootstrapping procedure. This results in accelerating bootstrapping performance without compromising security level. Our implementation of CKKS bootstrapping also incorporates this optimization for better performance.

Parameter Selection for META-BTS In theory, one can choose arbitrary precision for the base bootstrapping. However, the sweet spot is often a moderate precision, as both high and low precision cases present different inefficiencies. If n is too small, then the iteration number k should be large and the bootstrapping becomes very slow. If n is too large, then each base bootstrapping consumes too much modulus and the base bootstrapping cost increases by greatly. In this regard, we find the sweet spot based on experiments, i.e. try multiple choices of (n, k) and choose the one with the best performance. Note that the choice may depend on specific parameters such as N. For instance, we choose $n = 16$ for $N = 2^{15}$ (in Sec. 6.2), $n = 21$ for $N = 2^{16}$, and $n = 30$ for $N = 2^{17}$ (in Sec. 6.2).

6.2 Benchmark Results

Before presenting benchmark results, we share the basic parameter-setting strategy for our bootstrapping method. We first fix the target BFV bootstrapping functionality $(q, t, B_{\mathsf{in}}, B_{\mathsf{out}})$. Then, the required CKKS bootstrapping functionality is derived as $(\Delta, q, B_{\mathsf{in}}, B_{\mathsf{out}})$. As we instantiate this CKKS bootstrapping with the META-BTS algorithm $\texttt{CKKS.Boot}^{(k)}$, parameter setting boils down to deciding the precision parameter n of the base CKKS bootstrapping $\texttt{CKKS.BaseBoot}$. After fixing the precision parameter n, the iteration count k is set to $\lceil \log_2(B_{\mathsf{in}}/B_{\mathsf{out}})/n \rceil$, and the functionality of the base CKKS bootstrapping $\texttt{CKKS.BaseBoot}$ is determined as $(\Delta/2^{(k-1)n}, q, B_{\mathsf{in}}/2^{(k-1)n}, B_{\mathsf{out}})$. In the case of the redundant base CKKS bootstrapping functionality, where B_{out} is large, we instantiate it by the CKKS bootstrapping with functionality $(\Delta/(q' \cdot 2^{(k-1)n}), q/q', B_{\mathsf{in}}/(q' \cdot 2^{(k-1)n}), B_{\mathsf{out}}/q')$ for some proper integer q' leveraging the property in Lemma 1. This results in better bootstrapping performance since the overall ciphertext modulus is decreased by a factor of q'. Finally, the bootstrapping key, precomputed data required for performing bootstrapping, is generated according to the base CKKS bootstrapping functionality.

Performance Comparison with Digit Extraction. We first compare the performance of our bootstrapping with the state-of-the-art digit extraction-based bootstrapping [GIKV23]. We set the functionality parameters of our method at a similar level to the benchmark results in [GIKV23] to provide a fair comparison. The detailed BFV bootstrapping functionality parameters are presented in Table 2.

Table 2. BFV bootstrapping functionality used in the benchmark in Table 3

	q	t	B_{in}	B_{out}
[GIKV23]	1200 bits	51 bits	1137 bits	1006 bits
Ours	1200 bits	51 bits	1077 bits	949 bits

To achieve the given BFV bootstrapping functionality, we utilized the 16-bits base CKKS bootstrapping `CKKS.BaseBoot` with the iteration count 8. Also, to provide a fair comparison in latency, we set the ring dimension as $2^{16} = 32768$, which is similar to the ring dimension 42336 in the previous benchmark [GIKV23]. Under these parameter settings, we measured the elapsed time for bootstrapping, and its results are presented in Table 3, along with the benchmark result in [GIKV23].[4]

Table 3. Bootstrapping performance comparison between ours and [GIKV23]

	[GIKV23]	Ours
Cyclotomic index M	42799 $= 127 \cdot 337$	65536 $= 2^{16}$
Ring dimension N	42336	32768
Security level (bits)	80	128
Plaintext modulus t (bits)	51	
Denoising factor (bits)	131	128
Boot time (sec)	**1344+**[5]	**35.5**
Amortized boot time (ms/coefficient)	**31.7+**	**1.08**

While offering similar bootstrapping functionality, our method outperforms the previous approach by a factor of 37.9 in latency. Additionally, our method improves the throughput by a factor of 29.4 when comparing the elapsed time per each coefficient of the plaintext. We attribute this result to the large number of slots in CKKS since our method utilizes CKKS bootstrapping as a key step. To be precise, we recall that the time complexity of digit extraction is not only affected by the size of p, but also the number of slots, which is determined by $\mathrm{ord}_M(p)$. Actually, in the benchmark result in [GIKV23], it can only utilize 2016 slots, so it needs to evaluate the digit extraction polynomial $42336/2016 = 21$ times, whereas the CKKS packing method supports 16384 slots, making it sufficient for the evaluation of the modular reduction function twice. We also note that the plaintext modulus in [GIKV23] is set to 2^{51} for the efficiency of digit extraction-based bootstrapping, while our method uses a similarly scaled prime number to leverage the efficiency of RNS representation. If the plaintext modulus of [GIKV23] is changed to ours, its performance will be greatly degraded since the digit extraction method performs its worst in such a case. Conversely, our method would still yield similar latency even if we use the plaintext modulus 2^{51}, since, in the asymptotic scale, the performance of our algorithm mainly depends on the denoising

[4] The timing in [GIKV23] is measured on an Intel Core i7-6700HQ CPU, which is comparable to our hardware specifications.

[5] We estimated it from Table 7 in [GIKV23] by multiplying $21 = 42336/2016$ since it only measured elapsed time for a single iteration of digit extraction polynomial evaluation.

factor, not the number-theoretic properties of the plaintext modulus, as discussed in the previous section. We also note that the benchmark in [GIKV23] only measured elapsed time for digit extraction due to technical issues, excluding other operations such as homomorphic linear transformations. Furthermore, our bootstrapping parameters provide a higher security level. Hence, the actual performance improvement of our method yields even better results.

Arbitrary Plaintext Modulus. In this benchmark, we highlight the flexibility of our bootstrapping method in choosing the plaintext modulus. The performance of our method is primarily influenced by the denoising factor $\log_2(B_{\mathsf{in}}/B_{\mathsf{out}})$ rather than the number-theoretic properties of the plaintext modulus t. Therefore, in our benchmark, we maintain fixed functionality parameters B_{in} and B_{out} while varying the plaintext modulus t. The results are presented in Table 4.

Table 4. Bootstrapping performance for various plaintext moduli.

N	q	t	B_{in}	B_{out}	Boot time
2^{17}	791 bits	**54** bits	376 bits	16 bits	**392** sec
		144 bits			
		234 bits			
2^{16}	541 bits	**42** bits	223 bits	13 bits	**128** sec
		105 bits			
		168 bits			

In our benchmark, we maintain a fixed denoising factor of 360 bits (resp. 210 bits). To achieve this, we employ the META-BTS method with a 30-bit (resp. 21-bit) precision base for CKKS bootstrapping and 12 (resp. 10) iteration counts. Additionally, we set the ring dimension to 2^{17} (resp. 2^{16}) to support large precision CKKS bootstrapping. A notable observation is that the elapsed time for bootstrapping remains constant, even with changes in the plaintext modulus. This constancy arises because the plaintext modulus only impacts the input ciphertext modulus of CKKS bootstrapping for homomorphic lifting when other parameters are fixed. The input ciphertext modulus, however, does not affect the CKKS bootstrapping performance if it exceeds a certain bound. Consequently, all experiments internally utilize the same CKKS bootstrapping for approximated lifting. This result directly illustrates the unique properties of our bootstrapping method, where the denoising factor plays a crucial role in performance, whereas the performance of the digit extraction-based method varies significantly depending on the plaintext modulus. In addition, our method achieves viable bootstrapping performance for large plaintext moduli, such as 144 and 234 bits, which, to the best of our knowledge, has not been presented before. Therefore, our method significantly overcomes the previous limitations on plaintext modulus in BFV bootstrapping.

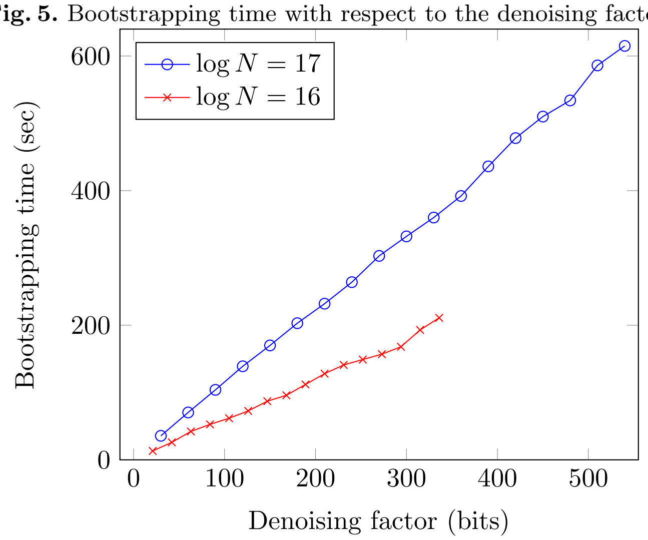
Adjustable Functionality. We discuss another distinctive property of our bootstrapping method: the ability to adjust the bootstrapping functionality depending on the situation. As mentioned earlier, our bootstrapping method allows adjusting the input and output noise bounds by leveraging the properties of CKKS bootstrapping and the META-BTS method. Additionally, the performance of the adjusted bootstrapping is determined by the denoising factor $(k - a - b)n$ bits. To demonstrate this effect more concretely, we measure the bootstrapping performance for various input and output noise bounds in Table 5. We also plot the bootstrapping time with respect to the denoising factor in Fig. 5. We note that the benchmark is performed with the same parameters as the benchmark in Table 4.

As indicated in Table 5, the elapsed time is identical when the denoising factor is the same, as it internally runs the same CKKS bootstrapping. Additionally, the bootstrapping time is proportional to the denoising factor since the iteration number of META-BTS is determined by the denoising factor, as presented in Fig. 5. Consequently, our method facilitates a more adaptive evaluation strategy depending

Table 5. Bootstrapping performance for various input and output noise bounds.

N	q	t	B_{in}	B_{out}	Boot time
2^{17}	791 bits	54 bits	556 bits	16 bits	615 sec
				106 bits	510 sec
				196 bits	392 sec
			466 bits	16 bits	510 sec
				106 bits	392 sec
			376 bits	16 bits	392 sec
2^{16}	541 bits	42 bits	349 bits	13 bits	211 sec
				76 bits	157 sec
				139 bits	128 sec
			286 bits	13 bits	157 sec
				76 bits	128 sec
			223 bits	13 bits	128 sec

Fig. 5. Bootstrapping time with respect to the denoising factor



on the scenarios. For instance, when performing bootstrapping for ciphertexts with small noise, setting a smaller input noise bound yields faster bootstrapping while maintaining the same output noise bounds. Conversely, if the required multiplicative depth is small after bootstrapping, setting a larger output noise bound results in better performance.

7 Conclusion

In this work, we presented a novel BFV bootstrapping method that incorporates CKKS bootstrapping as a subroutine. Since our bootstrapping follows a completely different pipeline, it does not inherit the limitation of previous digit extraction-based bootstrapping, where the plaintext modulus must be a power of a small prime. This not only provides flexibility in choosing the plaintext modulus but also enhances bootstrapping performance since our method utilizes high-throughput SIMD operations in CKKS, whereas digit extraction can utilize a relatively small number of slots due to limitations on the plaintext modulus. Additionally, our method allows for the adjustment of bootstrapping performance by varying bootstrapping functionality, which is intractable with the previous method. This property enables the evaluation of complex circuits in a more optimized way.

We expect that our bootstrapping method will integrate various subdivided research areas in homomorphic encryption. For instance, as our method utilizes CKKS bootstrapping, improvements in CKKS bootstrapping, such as algorithmic optimization [LLL+21, LLK+22, JM22, BCC+22], or hardware acceleration [JKA+21, KKK+22], directly contribute to enhancing BFV bootstrapping. Simultaneously, our method broadens the range of potential applications for BFV. In some BFV applications, such as private machine learning [GBDL+16], the significance lies in the size of the plaintext modulus rather than its number-theoretic properties for achieving sufficient precision. Additionally, these applications often require bootstrapping, as they typically involve evaluating large-depth arithmetic circuits. Conversely, for other applications like private database queries [KLLW16, TLW+20] or private set intersection [CLR17, CHLR18, CMdG+21], having a prime plaintext modulus is crucial for efficient homomorphic comparison or equality tests, leveraging number-theoretic properties. The previous method faced challenges in covering all these BFV use cases, especially when the plaintext modulus is a large prime number. This limitation hindered the convergence of various applications of BFV. Our method resolves this issue by providing viable performance regardless of the number-theoretic properties of the plaintext modulus. This flexibility potentially expands the range of applications for BFV.

Acknowledgement

This work was supported by Creative-Pioneering Researchers Program through Seoul National University.

References

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

BCC+22. Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. Meta-bts: Bootstrapping precision beyond the limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 223–234, 2022.

BGV14. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

BTPH22. Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In *Applied Cryptography and Network Security*, pages 521–541, 2022.

CGGI20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

CH18. Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved fhe bootstrapping. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 315–337. Springer, 2018.

CHK+18a. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.

CHK+18b. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, pages 347–368. Springer, 2018.

CHLR18. Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled psi from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1223–1237, 2018.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

CLR17. Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1243–1255, 2017.

CMdG+21. Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled psi from homomorphic encryption with reduced computation and communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1135–1150, 2021.

Cry23. CryptoLab. Heaan. `https://heaan.it/`, 2023.

DM15. Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

GBDL+16. Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pages 201–210. PMLR, 2016.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM, 2009.

GHS12. Craig Gentry, Shai Halevi, and Nigel P Smart. Better bootstrapping in fully homomorphic encryption. In *International Workshop on Public Key Cryptography*, pages 1–16. Springer, 2012.

GIKV23. Robin Geelen, Ilia Iliashenko, Jiayi Kang, and Frederik Vercauteren. On polynomial functions modulo pe and faster bootstrapping for homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 257–286. Springer, 2023.

GV23. Robin Geelen and Frederik Vercauteren. Bootstrapping for bgv and bfv revisited. *Journal of Cryptology*, 36(2):12, 2023.

HPS19. Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In *Cryptographers' Track at the RSA Conference*, pages 83–105. Springer, 2019.

HS15. Shai Halevi and Victor Shoup. Bootstrapping for helib. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 641–670. Springer, 2015.

JKA+21. Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 114–148, 2021.

JM22. Charanjit S Jutla and Nathan Manohar. Sine series approximation of the mod function for bootstrapping of approximate he. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 491–520. Springer, 2022.

KDE+23. Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for rlwe-based homomorphic encryption. *IEEE Transactions on Computers*, 2023.

KKK+22. Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. Bts: An accelerator for bootstrappable fully homomorphic encryption. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 711–725, 2022.

KLLW16. Myungsun Kim, Hyung Tae Lee, San Ling, and Huaxiong Wang. On the efficiency of fhe-based private queries. *IEEE Transactions on Dependable and Secure Computing*, 15(2):357–363, 2016.

LLK+22. Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 551–580. Springer, 2022.

LLL+21. Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 618–647. Springer, 2021.

OPP23. Hiroki Okada, Rachel Player, and Simon Pohmann. Homomorphic polynomial evaluation using galois structure and applications to bfv bootstrapping. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 69–100. Springer, 2023.

TLW+20. Benjamin Hong Meng Tan, Hyung Tae Lee, Huaxiong Wang, Shuqin Ren, and Khin Mi Mi Aung. Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. *IEEE Transactions on Dependable and Secure Computing*, 18(6):2861–2874, 2020.