**TOTA: Fully Homomorphic Encryption with Smaller Parameters and Stronger Security**

Zhaomin Yang\textsuperscript{1,2}, Xiang Xie\textsuperscript{1,2}, Huajie Shen\textsuperscript{3,}\textsuperscript{*}, Shiying Chen\textsuperscript{3,}\textsuperscript{*}, and Jun Zhou\textsuperscript{3}

\textsuperscript{1} Shanghai Key Laboratory of Privacy-Preserving Computation
\textsuperscript{2} MatrixElements Technologies
\textsuperscript{3} East China Normal University

**Abstract.** We present fully homomorphic encryption schemes for fixed-point arithmetic with fixed precision. Our scheme achieves IND-CPA\textsuperscript{D} security and uses RLWE ring with dimension \(2^{13}\) or less. Our techniques could also be extended to construct fully homomorphic encryption schemes for approximate numbers with IND-CPA security. The bootstrapping process of our IND-CPA scheme preserves about 39-bit precision with ring dimension \(2^{13}\), which is the first construction that preserves high precision while keeping the parameters small.

The core technique in this paper is a new and efficient functional bootstrapping algorithm that avoids the negacyclicity constraint of the evaluated functions, which enables us to extract bits blocks homomorphically. This new functional bootstrapping algorithm could be applied to BFV and TFHE schemes as well, and is of independent interest.

**Keywords:** Fully Homomorphic Encryption · Functional Bootstrapping · Approximate Encryption.

1 **Introduction**

Fully homomorphic encryption (FHE) \[38\] allows to perform arbitrary computation on encrypted data without decrypting the ciphertexts. Since the breakthrough work by Gentry \[23\], this field has been made a lot of progress both in theory and in practice. The core of Gentry’s blueprint is a bootstrapping technique allows to refresh a ciphertext into a new one that encrypts the same message while containing smaller errors. Bootstrapping is the only known technique to construct FHE, and all existing constructions, e.g., \[8,7,6,25,2,20,22,16,\] are based on this framework.

Different types of FHE schemes are proposed in the literature. BFV \[6,21\] and BGV \[7\] focus on homomorphic operations on finite field, FHEW \[20\] and TFHE \[14,15,16\] provide fast bootstrapping for binary operations, and CKKS \[13\] is dedicated for approximate numbers. In this paper, we focus on the fast bootstrapping procedure in FHEW, which is further improved in the TFHE scheme.

Functional bootstrapping, which is implicitly used in the TFHE scheme and formally introduced in \[4\], is a generalization of bootstrapping. It allows to refresh the error while performing some pre-determined function \(f\) on the encrypted message simultaneously. Functional bootstrapping turns out to be very useful, especially for oblivious inference in privacy-preserving machine learning. It is an effective method to deal with non-linear layers in machine learning models. Actually, \[5,28\] compute the sign function in the functional bootstrapping procedure to handle neural networks. However, the functions evaluated are subject to some constraints, it is an interesting problem to perform functions without the constraints in functional bootstrapping.

Let us first recall the main idea in FHEW. Based on the learning with errors problem \[37\], a ciphertext on \(m\) is of the form \((a, b = \langle a, s \rangle + m' + e \mod q)\), where \(a \leftarrow \mathbb{Z}_q^N, s \leftarrow \{0,1\}^N\) are uniformly random, \(e\) is

\* Work done as an internship at Shanghai Key Laboratory of Privacy-Preserving Computation and MatrixElements Technologies.
small and \( m' = \text{encode}(m) \) for some encoding algorithm. One could compute \( \text{decode}(b - \langle a, s \rangle) \) to recover \( m \), where \( \text{decode} \) is the corresponding decoding algorithm. For simplicity, we assume \( \text{encode} \) and \( \text{decode} \) both be the identity function. The FHEW bootstrapping leverages the ring structure of \( \mathbb{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1) \) for a larger modulus \( Q \). A key observation is that the elements \( \{1, X, X^2, \ldots, X^{N-1}, X^N, \ldots, X^{2N-1}\}^4 \) in \( \mathbb{R}_Q \) forms a multiplicative group of order \( 2N \). This allows to encrypt the elements in \( a \) and \( b \) in the exponent when \( q = 2N \). Given a bootstrapping key that encrypts each element of \( s \), we could homomorphically compute an encryption of \( v(X) \cdot X^{b - \langle a, s \rangle} \mod 2N = v(X) \cdot X^{m' + \epsilon} \), where \( v(X) \in \mathbb{R}_Q \) is some test vector. For a function \( f: \mathbb{Z}_{2N} \to \mathbb{Z}_Q \), we set the test vector as \( v(X) = f(0) - f(N - 1)X - \cdots - f(1)X^{N-1} \). Extract the constant term of \( v(X) \cdot X^{m' + \epsilon} \) will result an LWE ciphertext that encrypts \( f(m' + \epsilon) \). However, note that we could only embed \( N \) values in the coefficients and \( X^N = -1 \), the function \( f \) should satisfy that \( f(x + N) = -f(x) \mod Q \) for \( 0 \leq x < N \) to get the correct ciphertext. This property is called negacyclicity. This constraint heavily limits the usability of functional bootstrapping, and we will show later that functional bootstrapping for arbitrary functions will benefit us to solve some interesting problems and even improve the original bootstrapping.

In this paper, we focus on the CKKS-like scheme [13]. It provides an approximate encryption for fixed-point numbers. More specifically, the encoding function is set to \( \text{encode}(x) = [\Delta x] \) for some large \( \Delta \) and \( x \in \mathbb{R} \). The decoding \( \text{decode}(x) = x/\Delta \) removes the scale and returns a real number. Although the decoding function is not exactly correct, the resulting message is very close to the input of the encoding function. The error occurs in the encryption and homomorphic operation is viewed as part of the approximation error of the fixed-point number. This is acceptable in applications where fixed-point or floating-point numbers are applied. Due to the high efficiency of the CKKS scheme, it is widely used in applications related to privacy-preserving machine learning.

For practicality reasons, we often consider the \( \text{IND-CPA} \) security of homomorphic encryption, which means the public key and ciphertext do not leak any information of the plaintext. A recent work by Li and Micciancio [35] pointed out that the \( \text{IND-CPA} \) security may not adequately capture security against passive adversaries when applied to approximate encryption. In fact, they present concrete attacks to the CKKS scheme assuming the existence of passive decryption oracle. The main reason behind these attacks is that the approximation error leaks information of the secret key.

Li and Micciancio [35] formally define a stronger notion called \( \text{IND-CPA}^\text{D} \) security. In addition to the original \( \text{IND-CPA} \) model, the adversary could passively access a decryption oracle. As demonstrated in [35], \( \text{IND-CPA}^\text{D} \) security is equivalent to \( \text{IND-CPA} \) security for homomorphic encryption schemes with exact decryption, while \( \text{IND-CPA}^\text{D} \) security is strictly stronger than \( \text{IND-CPA} \) for approximate homomorphic encryption schemes. A natural method to achieve \( \text{IND-CPA}^\text{D} \) security for CKKS is to use noise flooding. It adds a sufficiently large error to mask the original error (may leak information of secret key) in the decryption procedure. Although this method fills the security gap, it involves in using multi-precision modulus and large ring dimensions, especially when turning CKKS into an FHE scheme. To construct a CKKS-like scheme under \( \text{IND-CPA}^\text{D} \) security without noise flooding is left as an open problem in [35]. It is also open to construct CKKS-like FHE schemes under \( \text{IND-CPA}^\text{D} \) security without noise flooding.

Research on bootstrapping of the CKKS scheme is very active recently. The first bootstrapping for CKKS is presented in [12], all the follow-up works [10,27,33,34,32] apply the same framework to design improved bootstrapping for CKKS. In a nutshell, it requires to approximate the modular reduction function with large-depth homomorphic operations. The errors amplified by these operations will distort the plaintext significantly. The bootstrapping procedures can only achieve 20-bit precision until using the very recent techniques in [32]. [32] devises better approximate approaches to achieve about 40-bit precision with ring dimension \( 2^{16} \). It is still unknown how to design bootstrapping for CKKS-like scheme with small parameters, say with dimension \( 2^{13} \) or less.

---

\(^4\) Note that \( X^N = -1 \) in \( \mathbb{R}_Q \).
1.1 Our Contributions

In this paper, we propose an efficient functional bootstrapping algorithm, which we call fully functional bootstrapping, that allows to perform arbitrary functions with in domain \( \mathbb{Z}_N \) on the plaintext. We note that in our setting the ciphertext modulus is \( N \), it is different from the case that restricting the plaintext in the interval \([0, N)\) with ciphertext modulus \( 2N \). Although the plaintext size is 1-bit smaller, our fully functional bootstrapping algorithm is simple and efficient, and is of independent interest.

Based on our fully functional bootstrapping algorithm, we construct a fully homomorphic encryption scheme TOTA with IND-CPA\(^D\) security for fixed-point arithmetic with fixed precision. By fixed precision, we mean that the precision always preserves after different operations, e.g., multiplication on fixed-point numbers. More specifically, given two fixed-point numbers \( x, y \) with precision \( p \), the fixed-precision multiplication drops the last \( p \) bits of \( xy \) to ensure it still preserves \( p \) bits. In fact, we construct an efficient homomorphic truncation algorithm based on our fully functional bootstrapping algorithm with ring dimension \( 2^{13} \). From another perspective, instead of encrypting approximate numbers, we perform operations on approximate circuits. Given a fixed-point or floating-point circuit, we first approximate it with a fixed-precision circuit and perform our FHE scheme with exact decryption on the new circuit.

Our techniques could be applied to approximate encryption schemes as well, which results in a totally different bootstrapping procedure for CKKS-like schemes. Our new bootstrapping technique does not need to approximate the modular reduction function and only introduces very small errors in the new ciphertext, which enables us to preserve high precision with small parameters. In fact, our FHE for approximate numbers preserves about 39-bit precision with ring dimension \( 2^{11} \).

We implement TOTA from scratch with C++ and conduct a series of numerical experiments under different parameter sets which provide at least 127-bit security. For IND-CPA\(^D\) security, the truncation algorithm of TOTA runs in about 77s (50s) for 14-bit (12-bit) fixed precision. For IND-CPA security, the algorithm runs in about 68s (8s) for about 39-bit (11-bit) precision.

1.2 Overview of Our Techniques

Let us briefly describe our fully functional bootstrapping algorithm. Let \((a, b) = (a - \langle a, s \rangle + m + e \mod N)\) be a ciphertext with module \( N \), where \( N \) is the ring dimension of the bootstrapping key. As discussed before, the FHEW bootstrapping technique encrypts \( a, b \) in the exponent and performs the decryption circuit homomorphically. However, we can not directly apply this method in this case, since the multiplicative group is of order \( 2N \), while the decryption circuit is performed in \( \mathbb{Z}_N \).

To overcome this problem, we view \((a, b)\) as elements in \( \mathbb{Z}_{2N} \). A key observation is that we can write \( b + (a, s) = m + e + kN \mod 2N \) for some \( k \in \{0, 1\} \). In fact, \( k \) is the most significant bit of \( m + e + kN \). Apply the original (functional) bootstrapping method in FHEW, we get a ciphertext of \(-kN \) with modulus \( 2N \). Thus, we obtain a ciphertext of \( m + e \) with modulus \( 2N \) by adding the two ciphertexts. Given any function \( f \) defined in \( \mathbb{Z}_N \), we extend it to \( F(x) \) defined in \( \mathbb{Z}_{2N} \) by setting \( F(x) = f(x) \) for \( 0 \leq x < N \) and \( F(x) = -f(x - N) \) for \( N \leq x < 2N \). Note that \( m + e \in [0, N) \), we could perform the functional bootstrapping algorithm again on \( F \) and get the desired result. Our fully functional bootstrapping only involves two original functional bootstrapping procedures and one homomorphic addition. It could be applied to BFV-like and TFHE-like ciphertexts as well.

In order to construct FHE schemes with IND-CPA\(^D\) security, we first define fixed-precision arithmetic. Denote \( \mathbb{F}_p \) as a set of fixed-point numbers with \( p \) precision. Multiplication on \( \mathbb{F}_p \) is defined as \( \text{Trunc}_p(xy) \) for \( x, y \in \mathbb{F}_p \) by dropping the last \( p \) bits of \( xy \). Note that the decryption will recover the exact \( \mathbb{F}_p \) value as long as the scale factor is sufficiently large and the errors do not distort the plaintext. In applications related to fixed-point or floating-point arithmetic, we first approximate the circuit with a fixed-precision circuit and then perform operations on \( \mathbb{F}_p \) with exact decryption.

It remains to design an efficient homomorphic truncation procedure. Actually, we design an extraction algorithm to extract bits blocks using our fully functional bootstrapping algorithm. Given a ciphertext of \( m \) under modulus \( q \), assume \( q \) is a power of \( 2 \) and \( N/q \). For an integer \( \delta \), one could drop the last \( \delta \) bits of
\(m\) and then leftshift it by \(\ell_e\) bits by switching the modulus from \(q\) to \((2^\ell q)/2^\delta\). The \(\ell_e\) bits (all 0’s) are reserved to eliminate the errors. Let \(d = \log N - \ell_e\), we get \(d\) bits of \(m\) starting from position \(\delta\) by taking the ciphertext modulo \(N\). Then, invoking the fully functional bootstrapping with \(f = 2^\delta \cdot \lfloor x/2^{\ell_e} \rfloor\) on the new ciphertext refreshes the errors and puts the \(d\) bits into position \(\delta'\). With careful noise analysis, we could set \(d \approx \ell_e \approx (\log N)/2\).

With the bits extraction algorithm, we could truncate the message as follows. Given two ciphertexts of \(\Delta x\) and \(\Delta y\), where \(\Delta\) is a scale factor such that \(\log \Delta > 2p\), the multiplication of the two ciphertexts results in a new ciphertext of \(\Delta^2 xy\), where \(xy \in \mathbb{F}_{2^p}\). Note that \(\Delta\) is sufficiently large such that the error in the multiplication will not distort \(xy\). We first extract the last \(p\) bits of \(xy\) by using the bits extraction algorithm starting from position \(2 \log \Delta - 2p\), then subtract it to get a ciphertext of \(\text{Trunc}_p(xy)\) with scale \(\Delta^2\) and large errors. Finally, we set the scale to \(\Delta\) and refresh the errors by invoking the bits extraction algorithm again. The basic workflow of our truncation algorithm is given in Fig 1.

![Fig. 1. Overview of Homomorphic Truncation](image)

However, this technique incurs another problem. As depicted in the last step of Fig 1, it moves the bits blocks to the lower position. If the encrypted number is negative, which is represented in the two’s complement form, this method will not preserve all the higher bits (e.g., all 1’s) in the original plaintext which results in a wrong message in the final ciphertext. Note that there is no such problem when the plaintext is positive, because the higher bits are all 0’s. To overcome this problem, we apply our fully functional bootstrapping algorithm to the last \(d\)-bit block with a carefully designed function. If the most significant bit of the last \(d\)-bit block is 0, the function outputs the \(d\)-bit block. Otherwise, it brings the higher bits to the output. Since the higher bits are publicly known (i.e., all 1’s), the function could be defined in advance.

The bits extraction algorithm could be easily extended to bootstrap approximate encryption. In this case, \(p \approx \log \Delta\) since approximation errors are allowed. We do not need to drop the last \(p\) bits, and could apply the bits extraction algorithm at a position close to \(\log \Delta\) directly. The main difference is when invoking the bits extraction algorithm in the first \(d\)-bit block, there is not enough bits reserved to eliminate the errors. The extracted block may be different from the original one. Fortunately, we show that the whole plaintext extracted in this case is also very close to the original plaintext. The approximation errors in this case are dominated by the errors in our fully functional bootstrapping algorithm, which is
very small. Thus, the approximate FHE is able to preserve high precision while keeping the ring dimension small.

1.3 Related and Concurrent Works

Li and Micciancio [35] introduce the notion of IND-CPAD, which captures the passive security of approximate homomorphic encryption. Functional bootstrapping is formally introduced in [4], applications and further improvements are proposed in [5,28,9,29,26,17]. The method of bootstrapping on CKKS scheme is developed by a series of works [12,10,27,33,34,32,3], which approximates the modular reduction function with polynomial evaluations.

Concurrent Works. Very recently, Chillotti et al. [18] and Kluczniak and Schild [31] present functional bootstrapping algorithms without the negacyclicity restriction independently. We note that our fully functional bootstrapping algorithm is different from theirs, and is simpler and more efficient.

Chillotti et al. [18] propose two methods to perform functional bootstrapping avoiding negacyclicity, which allow to evaluate any function on domain $[0, 2N)$. Both methods have to invoke at least two bootstrapping procedures and one multiplication by extending the BFV-like multiplication to TFHE. The BFV-like multiplication causes fast noisy growth in their scheme, it is also the main reason that they could not use small parameters as in TFHE and do not report concrete performance.

Kluczniak and Schild [31] present full domain functional bootstrapping to perform any function on $[0, 2N)$ without the restriction of negacyclicity. They split a function $F(x)$ defined on $[0, 2N)$ into two negacyclic functions $F_0$ and $F_1$, and then invoke multiple functional bootstrapping procedures to compute a GSW ciphertext of a bit $b$ indicating which interval the message lies in. Finally, they apply another functional bootstrapping to compute $\text{GSW}(1-b) \cdot \text{LWE}(F_0(x)) + \text{GSW}(b) \cdot \text{LWE}(F_1(x))$, which is a ciphertext of $F(m) = F_b(m)$. However, in order to get a GSW ciphertext of $b$, they have to run about $\log Q$ functional bootstrapping procedures, which is the main bottleneck of their scheme.

Recall that our fully functional bootstrapping focuses on functions on $[0, N)$. It only involves two bootstrapping procedures and one addition. Our fully functional bootstrapping could be easily applied to [18] and [31] to reduce parameters and improve efficiency.

A concurrent work by Kim et al. [30] proposes a new bootstrapping for CKKS without approximating the modular reduction function. They also apply the FHEW bootstrapping but with very different techniques from ours. Although they claim it is possible to set the ring dimension to $2^{14}$ or less, they do not report implementations of the algorithms at the time of this writing.

2 Preliminaries

2.1 Notations

All logarithms are base 2. We denote by $\langle \cdot, \cdot \rangle$ the inner product of two vectors. For a real number $r$, $\lfloor r \rfloor, \lfloor r \rfloor, \lceil r \rceil$ denote the nearest round, upper round and floor round of $r$ to integers, respectively. We use randomized rounding in our analysis and deterministic rounding in implementation as in [20]. We use $x \leftarrow D$ to denote the sampling of $x$ according to distribution $D$. It denotes the sampling from the uniform distribution from $D$ when $D$ is a finite set.

For $q \in \mathbb{Z}$, we identify $\mathbb{Z}_q$ with the set $[-q/2, q/2) \cap \mathbb{Z}$ by default. We will point it out when $\mathbb{Z}_q$ lies in $[0, q)$. For $x \in \mathbb{Z}$, let $[x]_q \in \mathbb{Z}_q$ be such that $[x]_q \equiv x \pmod{q}$. Define the rings $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$, where $N$ is a power of 2. We use bold $\mathbf{a} \in \mathbb{Z}_q^N$ for vectors, and lower case $a \in \mathcal{R}$ or $a \in \mathcal{R}_q$ for ring elements. Given a ring element $a \in \mathcal{R}$ or $a \in \mathcal{R}_q$, define the corresponding coefficient vector as $\mathbf{a} = (a_0, \ldots, a_{N-1})$, where $a_i \in \mathbb{Z}$ or $a_i \in \mathbb{Z}_q$ for $0 \leq i \leq N-1$ and $a = a_0 + a_1 X + \cdots + a_{N-1} X^{N-1}$. Denote $\mathbf{a} = \text{ToCoef}(a)$ and $a = \text{ToRing}(a)$.
2.2 Fixed-Precision Numbers

In this paper, we use fixed-precision numbers to approximate fixed-point and floating-point numbers. In a nutshell, fixed-precision numbers always preserve some fixed precision even after different operations, say multiplication. Formally, we define the fixed-precision real numbers as follows.

Definition 1 (Fixed-Precision Numbers). A set of fixed-precision numbers with \( p \) bits of precision is defined as

\[
\mathbb{FP}_p = \left\{ x \mid x = \pm \left( z + \sum_{i=1}^{p} b_i \cdot 2^{-i} \right) ; z \in \mathbb{Z}_{\geq 0}, b_i \in \{0, 1\} \right\}.
\]

We also define operations on \( \mathbb{FP}_p \). It is easy to verify that \( x + y \in \mathbb{FP}_p \) for \( x, y \in \mathbb{FP}_p \). It remains to define the multiplication operation. Before that, we first define the truncation function as \( \text{Trunc}_p(x) = \lfloor 2^p x \rfloor / 2^p \) for \( x \in \mathbb{R} \). It preserves \( p \)-bit precision of the fractional part of \( x \) and drops the others. We define multiplication over \( \mathbb{FP}_p \) as follows.

Definition 2. For any \( x, y \in \mathbb{FP}_p \), define the multiplication operation as \( x \odot y = \text{Trunc}_p(xy) \in \mathbb{FP}_p \).

Note that \( xy \in \mathbb{FP}_{2p} \), the truncation function drops the last \( p \) bits of the fractional part and moves the result back to \( \mathbb{FP}_p \).

In our fully homomorphic encryption scheme, when dealing with fixed-point or floating-point arithmetic, we approximate it with fixed-precision arithmetic. Our FHE scheme supports homomorphic operations over \( \mathbb{FP}_p \), the exact decryption procedure ensures the IND-CPA security.

2.3 Gaussian Distributions and (R)LWE Encryption

Let \( \sigma > 0 \), define the Gaussian function \( \rho_\sigma(x) = \exp(-\pi x^2 / \sigma^2) \) for \( x \in \mathbb{R} \). Denoted by \( \chi_\sigma \) the discrete Gaussian distribution on \( \mathbb{Z} \) with variance \( \sigma^2 \) by sampling from continuous Gaussian probability distribution with density \( \rho_\sigma(x) / \sigma \) and then rounding it to the nearest integer. Denoted by \( \chi^N_\sigma \) the discrete Gaussian distribution on \( \mathbb{Z}^N \) by sampling each element from \( \chi_\sigma \) independently. We follow the heuristic approaches of error analysis in [19,24]. Hence, we will use \( 6\sigma \) as a high-probability bound on the size of elements sampled from \( \chi_\sigma \).

The Learning with Errors (LWE) problem is introduced by Regev [37]. It is extended to the ring version, which is called RLWE, by Lyubashevsky, Peikert and Regev [36]. Let \( q > 0 \) be an integer, \( a \leftarrow \mathbb{Z}_q^N \) and \( s \leftarrow \mathcal{D}^N \). For simplicity, we focus on \( \mathcal{D}^N = \{0, 1 \}^N \). For a message \( m \in \mathbb{Z} \), the (symmetric) LWE encryption of \( m \) under \( s \) is of the form \((a, \lfloor \langle a, s \rangle + m \cdot e \rfloor_q) \in \mathbb{Z}^N + 1 \), where \( e \leftarrow \chi^N_\sigma \) and \( m' = \text{encode}(m) \). There always exists a \( \text{decode}() \) function to recover \( m \) from \( \text{encode}(m) + e \). We assume the \( \text{encode}/\text{decode} \) function be the identity function for simplicity. Let \( \text{LWE}_{s,q}^N(m) \) be the set of LWE ciphertexts of \( m \). We say \( \text{ct} = (a, b) \in \text{LWE}_{s,q}^N(m) \) has error variance \( \sigma^2 \), if \( |b + \langle a, s \rangle|_q - m \) has variance \( \sigma^2 \). Sometimes, we mix the use of \( \text{LWE}_{s,q}^N(m) \) and \( \text{LWE}_{s,q}^N(m + e) \) where \( e \) is the error term. If \( \text{ct} \in \text{LWE}_{s,q}^N(m) \) with error term \( e \), then \( \text{ct} \in \text{LWE}_{s,q}^N(m + e) \) with error term \( 0 \).

Let \( a \leftarrow \mathcal{R}_q^N \), \( s \leftarrow \mathcal{D}^N \) and \( s = \text{ToRing}(s) \). For a message \( m \in \mathcal{R} \), the RLWE encryption of \( m \) under \( s \) is of the form \((a, -as + m' + e) \in \mathcal{R}_q \times \mathcal{R}_q \), where \( e = \text{ToRing}(e) \) and \( e \leftarrow \chi^N_\sigma \) and \( m' = \text{encode}(m) \). Similarly, we assume \( \text{encode} \) be the identity function. Let \( \text{RLWE}_{s,q}^N(m) \) be the set of RLWE ciphertexts of \( m \). We say \( \text{ct} = (a, b) \in \text{RLWE}_{s,q}^N(m) \) has error variance \( \sigma^2 \), if each coefficient of \( (b + a \cdot s) - m \in \mathcal{R}_q \) has variance \( \sigma^2 \). In this paper, we only consider the “unpacking” version of RLWE encryption, which means \( m = m_0 + 0 \cdot X + \cdots + 0 \cdot X^{N-1} \) for some \( m_0 \in \mathbb{Z} \). Sometimes, we mix the use of \( \text{RLWE}_{s,q}^N(m) \) and \( \text{RLWE}_{s,q}^N(m + e) \) where \( e \) is the error term. If \( \text{ct} \in \text{RLWE}_{s,q}^N(m) \) with error term \( e \), then \( \text{ct} \in \text{RLWE}_{s,q}^N(m + e) \) with error term \( 0 \).

Additionally, RGSW [25] ciphertexts are needed in our constructions. Let \( G \in \mathcal{R}_q^{2 \times 2} \) be a gadget matrix for some integer \( l \), the concrete definition is presented in Definition 3 in Appendix A. For a
message $m \in \mathcal{R}$, sampling $ct_0, \ldots, ct_{2\ell-1} \in \text{RLWE}_x(0)$, the RGSW ciphertext of $m$ is of the form:

$$
\begin{pmatrix}
ct_0 \\
\vdots \\
ct_{2\ell-1}
\end{pmatrix} + m \cdot G.
$$

Similar to LWE and RLWE encryptions, we let $\text{RGSW}_x(m)$ be the set of RGSW ciphertexts. An RGSW ciphertext has error variance $\sigma^2$ if $\{ct_i\}_{i=0}^{2\ell-1}$ have error variance $\sigma^2$.

### 2.4 Useful Algorithms

In this subsection, we list some useful algorithms which will be used in our constructions.

**Extracting LWE from RLWE.** Given a RLWE ciphertext, it is well-known to extract LWE ciphertexts. Since our RLWE ciphertexts only encrypt constant messages, the extract procedure is much simpler as described in Algorithm 1. We remark that the extract algorithm does not change the error variance of the ciphertext.

**Algorithm 1** Extract ($ct$)

| Input: $ct = (a, b) = \text{RLWE}_x(m)$, |
| Output: $ct = \text{LWE}_x(m_0)$, where $s = \text{ToCoeff}(s)$. |
| 1: return $(a_0, -a_{N-1}, -a_{N-2}, \ldots, -a_1, b_0) \in \mathbb{Z}_{q+1}^{N+1}$ |

**External Product.** It allows to perform multiplication on a RGSW ciphertext and a RLWE ciphertext. The resulting ciphertext is a RLWE ciphertext encodes the product of the messages. Specifically, adapted from [16] and [20], the external product between a RGSW ciphertext $C \in \text{RGSW}_x(m_1)$ and a RLWE ciphertext $ct \in \text{RLWE}_x(m_2)$ produces a RLWE ciphertext $ct' \in \text{RLWE}_x(m_1 \cdot m_2)$.

$$
C \odot ct \rightarrow ct' \in \text{RLWE}_x(m_1 \cdot m_2)
$$

The details and error analysis of external product are given in Lemma 1 in Appendix A.

**Modulus Switching and Modular Reduction.** Two approaches are given to change modulus of ciphertexts. Modulus switching is used to reduce the magnitude of ciphertext modulus and that of the noise. Modular reduction allows to compute the residue of the plaintext.

**Algorithm 2** ModS$_{q \rightarrow q'}(ct)$: Switching modulus between LWE ciphertexts

| Input: $ct = (a, b) \in \text{LWE}_x(m)$; |
| $q'$ satisfies $q' < q$. |
| Output: $ct' = (a', b') \in \text{LWE}_x(m')$, where $m' = \lfloor (q'/q)m \rfloor$. |
| 1: Let $a' = \lfloor (q'/q) \cdot a \rfloor$, $b' = \lfloor (q'/q) \cdot b \rfloor$. |
| 2: return $ct' = (a', b')$. |

The correctness and error analysis are given in Lemma 2 in Appendix B. We note that the modulus switching procedure is crucial in our constructions to manage the errors. It is natural to extend to the ring case by performing the rounding operation on each coefficient of the ring elements. The error analysis is similar.

Modular reduction simply computes the residue of the ciphertext under a new and smaller modulus. The new ciphertext carries the residue of the original message.
Algorithm 3 Mod\(_{q 	o q'}(\text{ct})\): Modular reduction between LWE ciphertexts

**Input:** \(\text{ct} = (a, b) = \text{LWE}_{s,q}^N(m);\)
\(q'\) satisfies that \(q'|q\).

**Output:** \(\text{ct}' = (a', b') = \text{LWE}_{s,q'}^N(m'),\) where \(m' = m \mod q'\).

1. Let \(a' = a \mod q',\) \(b' = b \mod q'.\)
2. return \(\text{ct}' = (a', b').\)

The correctness of modular reduction is easy to verify since \(q'|q\). This procedure will not change the error variance as long as \(|m + e| < q'/2\), where \(e\) is the error term in \(\text{ct}\).

**Key Switching and Dimension Switching.** These two algorithms switch secret keys between two ciphertexts. To clarify the differences in this paper, key switching only focuses on the RLWE case and the dimension of the secret keys remains the same. Dimension switching is widely used in our fully functional bootstrapping procedure, it focuses on the LWE case and will change the dimension of the secret keys. Key switching is given in Algorithm 4, the correctness and error analysis are given in Lemma 3 in Appendix B.

Algorithm 4 Key\(_{s 	o s'}(\text{ct}, \text{KSK})\): Switching keys between RLWE ciphertexts

**Input:** \(\text{ct} = (a, b) \in \text{RLWE}_{s,q}^N(m);\)
A key-switching key \(\text{KSK} \in \text{RGSW}_{s,q}^N(s).\)

**Output:** \(\text{ct}' = (a', b') = \text{RLWE}_{s,q'}^N(m).\)

1. Let \((a', b') = (0, b) + \text{KSK} \oplus (0, a) \in \mathcal{R}_q \times \mathcal{R}_q\)
2. return \(\text{ct}' = (a', b').\)

For efficiency reasons, we apply the method proposed in [11] to switch dimensions for LWE ciphertexts while keeping the message unchanged. Given \(n|N\) and \((a, b) \in \text{LWE}_{s,q}^N(m),\) define polynomials with degree \(n\) as

\[
\tilde{a}_i = a_{m+n+1} - a_{m+n-2}X^2 - \cdots - a_{m+1}X^{n-1}, \quad \text{and}
\]
\[
\tilde{s}_i = s_{m+n+1} + s_{m+n-2}X^2 + \cdots + s_{m+1}X^{n-2} + s_{m}X^{n-1},
\]

where \(0 \leq i \leq N/n - 1.\) View \(\tilde{a}_i\) and \(\tilde{s}_i\) as elements in \(\mathbb{Z}_q[X]/(X^n + 1).\) Note that the constant term of \(\sum_{i=0}^{N/n-1} \tilde{a}_i \cdot \tilde{s}_i\) is exactly \((a, s) \in \mathbb{Z}_q.\) The algorithm is given in Algorithm 5. The correctness and error analysis are given in Lemma 4 in Appendix B.

Algorithm 5 Dim\(_{s 	o s'}(\text{ct}, \text{KSK}'_{s'}):\) Switching dimensions between LWE ciphertexts

**Input:** \(\text{ct} = (a, b) = \text{LWE}_{s,q}^N(m);\)
A key-switching key \(\text{KSK}'_{s' \to s'} = \{\text{KSK}'_i \in \text{RGSW}_{s',q}^N(s_i)\}_{i=0}^{N/n-1};\)

**Output:** \(\text{ct}' = \text{LWE}_{\text{ToCoS}(s')}_{q}(m)\) under key \(s' \in \mathbb{Z}_q[X]/(X^n + 1)\)

1. Construct polynomials \(\tilde{a}_i\) from \(a\) as above, for \(i \in \{0, \ldots, N/n - 1\}\)
2. Let \(\tilde{ct} = (0, b) + \sum \text{KSK}'_i \oplus (0, \tilde{a}_i),\)
3. return Extract(\(\tilde{ct}\))

**Evaluating Trace.** Given \(f(X) \in \mathcal{R}_q,\) let \(\varphi_i(x) : f(X) \mapsto f(X^i)\) for \(i = 1, 3, 5, ..., 2N - 1\) be the Galois automorphisms. It is well-known that automorphisms could be homomorphically evaluated via key switching, which is listed as in Algorithm 6. The error analysis is exactly the same as the key switching procedure.
We will apply the method in [11] to transfer an LWE ciphertext into a RLWE ciphertext via evaluating the trace function homomorphically. For \( a \in \mathcal{R}_q \), define the trace of \( a \) as the sum of all Galois transformations on \( a \), i.e., \( \text{Trace}(a) = \sum_i \varphi_i(a) \), where \( i \in \{1, 3, \ldots, 2N - 1\} \). A key observation is that \( \text{Trace}(1) = N \) and \( \text{Trace}(X^i) = 0 \) for \( i \neq 0 \). An efficient recursive homomorphic trace evaluation in [11] is described in Algorithm 7. We refer the details and correctness of this algorithm to [11].

With the homomorphic trace algorithm, we could transfer an LWE into a RLWE with the following algorithm. The resulting RLWE ciphertext encrypts the message as the constant term.

The correctness is shown in [11], and we refer the readers to that paper. The error analysis of this algorithm is given in Lemma 5 in Appendix B.

3 Functional Bootstrapping

Functional bootstrapping is a generalized version of the original bootstrapping procedure, which enables to refresh the ciphertext while performing some pre-determined function on the plaintext, simultaneously. We follow the FHEW-like bootstrapping [20] and generalize it to handle multiple bits in a direct way. We note that the generalized functional bootstrapping procedure is still subject to the constraint that \( f(x) = -f(x + N) \).

Consider an LWE ciphertext \( \mathbf{ct} \in \text{LWE}_{e,2N}(m) \). View it as a noiseless encryption of \( \tilde{m} = m + \varepsilon \), where \( \varepsilon \) is the error term of the original ciphertext. The functional bootstrapping procedure refreshes it into a ciphertext of \( f(m + \varepsilon) \) with error term independent of \( \mathbf{ct} \). It seems not like the “bootstrapping” procedure in the literature, because the original error term is still in the output. We will explain that it is sufficient for our purpose to handle fixed-precision numbers.
Moreover, it is straightforward to remove the error term $e$. Consider the ciphertext is of the form $\text{ct} \in \text{LWE}_{2N}^N(\text{encode}(m) + e)$, define an extended function $F = f \circ \text{decode}$, where $\circ$ means the composition of the two functions. The functional bootstrapping on $F$ will remove $e$, and the error term in the resulting ciphertext is independent of $\text{ct}$.

Let $Z_{2N}$ lie in $[0, 2N]$, we describe the functional bootstrapping procedure in Algorithm 9.

**Algorithm 9** $\text{FBS}_{(\tilde{r}, 2N)\rightarrow(N,q)}(\text{ct}, f, \text{BK})$

**Input:** An LWE ciphertext $\text{ct} = (a, b) \in \text{LWE}_{2N}^N(m)$ under key $\tilde{s} = (\tilde{s}_0, \ldots, \tilde{s}_{\tilde{n} - 1})$.

where $\{(a, \tilde{s}) + b\}_{2N} = m + e = \tilde{m}$;

A bootstrapping key $\text{BK} = \{\text{BK}_{i} \in \text{RGSW}^N_{2,q}(\tilde{s}_i)\}, 0 \leq i \leq \tilde{n} - 1$;

$f : Z_{2N} \rightarrow Z_q$ satisfying $f(x + N) = -f(x)$.

**Output:** $\text{ct}' \in \text{LWE}_{N}^N(f(\tilde{m}))$, where $\tilde{s} = \text{ToCoeff}(s)$.

1. Let $\sigma = [Qf(0)/q] - [Qf(N - 1)/q] X - \cdots - [Qf(1)/q] X^{N - 1} \in \mathbb{R}_Q$.
2. Let $\text{acc} \leftarrow (0, X^s \cdot v(X)) \in \mathbb{R}_Q \times \mathbb{R}_Q$.
3. for $i = 0$ to $\tilde{n} - 1$ do
4. \hspace{1em} $\text{acc} = \text{acc} + \text{BK}_i \sqcap ((X^{\tilde{s}_i} - 1) \cdot \text{acc})$.
5. end for
6. $\text{ct}' = \text{Extract}(\text{acc})$
7. return $\text{ct}' = \text{Mod}_{Q \rightarrow q}(\text{ct}')$

Denote $\text{SimpleFBS}_{(\tilde{r}, 2N)\rightarrow(N,q)}(\text{ct}, f, \text{BK})$ as a simplified version, which is exactly the same as the above $\text{FBS}_{(\tilde{r}, 2N)\rightarrow(N,q)}(\text{ct}, f, \text{BK})$ except running the modulus switching procedure in step 7, whose output $\text{ct}'$ is still on modulus $Q$.

**Remark 1.** The functional bootstrapping procedure does not switch the secret key back to $\tilde{s}$. This is because in our main construction, the secret key is $s$, and $\tilde{s}$ is only viewed as an intermediate key for efficiency reasons.

**Remark 2.** For CKKS-like encryption schemes, they usually encounter the case when $f$ is “somewhat continuous”. For example, activate functions in machine learning models. The extra error $e$ will not change the result too much, i.e., $f(\tilde{m}) \approx f(m)$, which is acceptable in real applications.

**Remark 3.** For BFV-like encryption schemes, the input ciphertext is of the form in $\text{LWE}_{2N}^N(\frac{2N}{b} \cdot m + e)$, where $t$ is the plaintext module, and we assume $t|2N$ for simplicity. Algorithm 9 could handle BFV ciphertexts by extending $f(x)$ into $f(\frac{1}{2N}(x))$, and the the constraint of $f$ is changed into $f(x) = -f(x + t/2)$.

**Theorem 1.** Let $\text{ct} \in \text{LWE}_{2N}^N(m)$ with error term $e$, $f : Z_{2N} \rightarrow Z_q$ be any function satisfying $f(x + N) = -f(x)$ and $Q \geq q$. Then $\text{ct}'$ output in Algorithm 9 is a ciphertext in $\text{LWE}_{N}^N(f(\tilde{m}))$ with error variance $\sigma^2$, where $\tilde{m} = m + e \in Z_{2N}$.

Furthermore, Let the error variance in $\text{BK}$ be $\sigma_r^2$, then

$$\sigma^2 \leq 2(q/Q)^2 \tilde{m} N \sigma^2 + N/24 + 1/6,$$

where $q$ is defined as follows: If base power gadget is used in $\text{BK}$, $g$ is the decomposition base. If RNS gadget is used then $g = \max\{y_{g_0}, ..., y_{g_{t-1}}\}$.

**Proof.** In step 2, $\text{acc}$ is a ciphertext in $\text{RLWE}_{2,N}^N(X^b \cdot v(X))$. In each iteration of step 4, the updated $\text{acc}$ is a ciphertext in $\text{RLWE}_{2,N}^N(X^{\sum_{j=0}^{k}a_j} \cdot v(X))$. Therefore, $\text{acc}$ is a ciphertext in $\text{RLWE}_{2,N}^N(X^{b+\sum_{j=0}^{k}a_j} \cdot v(X))$ after step 5. Since $[b + \langle a, \tilde{s} \rangle]_{2N} = \tilde{m}$, then we have $X^{b+\langle a, \tilde{s} \rangle} \cdot v(X) = X^m \cdot v(X)$. By the definition of $v(X)$
and \( f(x + N) = -f(x) \), the constant term of \( X^\tilde{m} \cdot v(X) = [Qf(\tilde{m})/q] \). Due to the modulus switching procedure, \( \mathbf{ct}' \) is a ciphertext in \( \text{LWE}^N_\tilde{q}(f(\tilde{m})) \).

Let \( \sigma_i^2 \) for \( 0 \leq i < 2\tilde{n} - 1 \) be the error variances of \( \text{acc} \) in each iteration, according to Lemma 1 we have
\[
\sigma_{i+1}^2 \leq 2\ell N^q g^2 \sigma_{bk}^2 + \tilde{\sigma}_i \cdot \sigma_i^2 \leq 2\ell N^q g^2 \sigma_{bk}^2 + \sigma_i^2.
\]
Since \( \sigma_0 = 0 \), then the error variance in \( \mathbf{ct}' \) is bounded by \( 2\ell \tilde{n} N^q g^2 \sigma_{bk}^2 \). According to Lemma 2, we have
\[
\sigma^2 \leq 2(q/Q)^2 \ell \tilde{n} N^q g^2 \sigma_{bk}^2 + N/24 + 1/6.
\]

### 3.1 Fully Functional Bootstrapping

Since function \( f \) in Algorithm 9 has to satisfy the negacyclic condition, it is not sufficient for our construction. This is because the functional bootstrapping procedure could not handle all the \( \log(2N) \) bits and the most significant bit is out of control in this setting.

We propose the fully functional bootstrapping algorithm, which could deal with any function \( f \), whose domain is \( \mathbb{Z}_N \) (instead of \( \mathbb{Z}_{2N} \)), on a ciphertext in \( \text{LWE}^N_\tilde{q}(m) \) by reducing the modulus from \( 2N \) into \( N \). We note that we homomorphic decryption process is still performed on the group with order \( 2N \).

In a nutshell, given a ciphertext in \((a, b) \in \text{LWE}^N_\tilde{q}(m)\), the fully functional bootstrapping procedure refresh it into a ciphertext in \( \text{LWE}^N_{\tilde{q}}(f(\tilde{m})) \), for any \( f : \mathbb{Z}_N \mapsto \mathbb{Z}_q \). The intuition of this algorithm is as follows. Viewing \((a, b)\) as a ciphertext with modulus \( 2N \), then \([a, b] + b_{2N} = m + e + k \cdot N\), for \( k \in \{0, 1\} \).

We first apply the functional bootstrapping procedure in \((a, b)\) to get a ciphertext of \(-kN\), and add it to get a ciphertext of \( m + e \) under modulus \( 2N \). Since \( m + e \) is less than \( N \), we could invoke the functional bootstrapping again on a function extended from \( f \) to obtain the required ciphertext. Details are given in Algorithm 10.

### Algorithm 10 FullyFBS\(_{(6, N)}\rightarrow\!(N, q)\) (\(\mathbf{ct}, f, \text{BK}, \tilde{\text{BK}}, \text{KSK}^\tilde{Q}_{s\rightarrow\tilde{s}}\))

**Input:** \(\mathbf{ct} = (a, b) \in \text{LWE}^N_\tilde{q}(m)\) under key \(\tilde{s} = (\tilde{s}_0, \ldots, \tilde{s}_{2\tilde{n} - 1})\);
- A bootstrapping key \(\text{BK} = \{\text{BK}_i \in \text{RGSW}^{\tilde{q}}_{\tilde{q}, s}(\tilde{s}_i) \}_{i=0}^{\tilde{n} - 1}\);
- A bootstrapping key \(\tilde{\text{BK}} = \{\tilde{\text{BK}}_i \in \text{RGSW}^{\tilde{q}}_{\tilde{q}, \tilde{s}}(\tilde{s}_i) \}_{i=0}^{\tilde{n} - 1}\);
- A key-switching key \(\text{KSK}^\tilde{Q}_{s\rightarrow\tilde{s}}\);
- Any function \( f : \mathbb{Z}_N \mapsto \mathbb{Z}_q \).

**Output:** \(\mathbf{ct}' \in \text{LWE}^N_{\tilde{q}}(f(\tilde{m} + \tilde{e}))\) for some \(\tilde{e}\), where \(s = \text{ToCoef}(s)\) and \(\tilde{m} = [b + (a, \tilde{s})]_N\).
1. Lift \((a, b)\) into \(\mathbb{Z}_{2N}\) to get \([a, b] + b_{2N} = m + e + kN\) with \(k \in \{0, 1\}\). Viewing \(\mathbf{ct} \in \text{LWE}^N_{\tilde{q}, s}(m + e + kN)\).
2. Perform \(\text{fix} = \text{SimpleFBS}_{(8, 2N)}\rightarrow\!(N, 2N)\)(\(\mathbf{ct}, f, \text{BK}\)) with function \(f_c(x) = N/2\) for \(x \in [0, N]\), \(f_c(x) = -N/2\) for \(x \in [N, 2N)\). Obtaining \(\text{fix} \in \text{LWE}^N_{\tilde{q}}((-1)^k \cdot \tilde{Q}/4)\).
3. \(\text{fix} = \text{ModS}_{\tilde{Q}\rightarrow\tilde{N}}(\text{DimS}_{N\rightarrow\tilde{N}}(\text{fix}, \text{KSK}^\tilde{Q}_{s\rightarrow\tilde{s}}))\).
4. \(\text{let fix} = \text{fix} + (0, -N/2)\), \(\triangleright \text{fix} \in \text{LWE}^N_{\tilde{q}}(-kN)\).
5. \(\text{let } \mathbf{ct}' = [\text{ct} + \text{fix}]_{2N} \in \text{LWE}^N_{\tilde{q}, 2N}(\tilde{m} + \tilde{e})\), \(\triangleright \text{fix} \) is the error term in \(\text{fix}\).
6. Let \(F(x) = f(x)\) for \(0 \leq x < N\), and \(F(x) = -f(x - N)\) otherwise.
7. Perform \(\mathbf{ct}'' = \text{FBS}_{(8, 2N)}\rightarrow\!(\mathbb{N}, q)\)(\(\mathbf{ct}'\), \(f, \text{BK}\)) \(\triangleright \mathbf{ct}' \in \text{LWE}^N_{\tilde{q}}(f(\tilde{m} + \tilde{e}))\).

**Remark 4.** We note that in step 2, the functional bootstrapping procedure does not switch the modulus from \(\tilde{Q}\) into \(2N\), which will be done in step 3. But we still use \([\tilde{Q}f_c(i)/(2N)]\) in the lookup table in step 2. The reason of doing this is to control the errors, since modulus switching will reduce the magnitude of the errors.

**Remark 5.** We use a smaller modulus \(\tilde{Q}\) in \(\tilde{\text{BK}}\) for efficiency reasons. \(\tilde{Q}\) could be much smaller than \(Q\). We note that if RNS gadget matrix is used, \(\tilde{\text{BK}}\) will not increase the key size, because it is only part of \(\text{BK}\).
Then which is also the error variance of $ct$. In this case, the error terms $e, \tilde{e}$ in Algorithm 10 are removed.

**Remark 6.** Algorithm 10 could be extended for BFV-like ciphertext similarly as in Remark 3. The plaintext modulus is reduced into $t/2$. Namely, the ciphertext is in $LWE_{q,N}^N(\frac{N}{t^2} \cdot m + e)$ and function $f : \mathbb{Z}_{q/2} \rightarrow \mathbb{Z}_q$ could be any functions. Step 2 remains unchanged, and $f(x)$ in step 6 is replaced by $f(\frac{1}{2N} \cdot x)$. Thanks to the analysis of Theorem 1, the error variance of $ct$ is very close to $\tilde{m} + \tilde{e} = m + e + \tilde{e} \in \mathbb{Z}$ and $\tilde{e}$ is some error term with variance $\tilde{\sigma}^2$.

**Theorem 2.** Let $ct \in LWE_{q,N}^N(m)$ with error term $e, f : \mathbb{Z}_N \rightarrow \mathbb{Z}_q$ be any function and $Q \geq \tilde{Q} \geq q > 2N$. Then $ct'$ output in Algorithm 10 is a ciphertext in $LWE_{q,N}^N(f([\tilde{m} + \tilde{e}]_N))$ with error variance $\tilde{\sigma}^2$, where $\tilde{m} + \tilde{e} = m + e + \tilde{e} \in \mathbb{Z}$ and $\tilde{e}$ is some error term with variance $\tilde{\sigma}^2$.

Furthermore, suppose $BK$ contains $2l$ RLWE ciphertexts with error variance $\sigma_{bk}^2$, $BK$ contains $2l$ RLWE ciphertexts with error variance $\sigma_{bk}^2$. Then

$$\tilde{\sigma}^2 \leq (2N/\tilde{Q})^2(N\ell_{ksk}g_{ksk}^2\sigma_{skk}^2 + 2\tilde{m}N\tilde{g}^2\sigma_{bk}^2) + \tilde{m}/24 + 1/6,$$

$$\sigma^2 \leq 2(q/Q)^2\ell nN\sigma^2 + N/24 + 1/6,$$

where $g$ is defined as follows: If base power gadget is used in $BK$, $g$ is the decomposition base. If RNS gadget is used then $g = \max\{g_0, ..., g_{t-1}\}$. $g, g_{ksk}$ are similarly defined in $BK$ and $KSK_{q \rightarrow s'}$. 

**Proof.** Since $m + e \in \mathbb{Z}_N$, then $m + e + N \in [N, 2N)$. Then $k$ identifies $m + e + kN$ belongs to $[0, N)$ or $[N, 2N)$. Since $f_c$ is negacyclic, due to Theorem 1 and SimpleFBS, $fix \in LWE_{\tilde{N}, \tilde{Q}}((-1)^k \cdot \tilde{Q}/4)$ in step 2, where $s = \text{ToCoef}(s)$. After dimension switching and modulus switching, $fix$ is transferred into a ciphertext in $LWE_{q,2N}^n((-1)^k \cdot N/2)$. Note that $(-1)^k - 1 = -2k$ for $k \in \{0, 1\}$. $ct''$ is a ciphertext in $LWE_{q,2N}^n(f([\tilde{m} + \tilde{e}]_N))$, where $\tilde{e}$ is the error term in $ct''$.

According to the definition of $x$, the modulus switching updates the error variance of $fix$ into

$$\sigma_{skk}^2 \leq (2N/\tilde{Q})^2(N\ell_{ksk}g_{ksk}^2\sigma_{skk}^2 + 2\tilde{m}N\tilde{g}^2\sigma_{bk}^2) + \tilde{m}/24 + 1/6$$

which is also the error variance of $ct''$, i.e., the variance of $\tilde{e}$.

According to Theorem 1, we have

$$\sigma^2 \leq 2(q/Q)^2\ell nN\sigma^2 + N/24 + 1/6.$$  

This completes the proof. \qed

**Remark 7.** Thanks to the modulus switching procedure, we will keep the error variance $\sigma^2$ and $\sigma'^2$ very close to $\tilde{m}$ and $N$, respectively.

## 4 Homomorphic Truncation for Fixed-Precision Numbers

In this section, we describe our algorithms to homomorphically evaluate multiplication of fixed-precision numbers. Namely, we show how to homomorphically evaluate the truncation procedure. The key ingredient is a bits extraction algorithm that heavily relies on the fully functional bootstrapping procedure.

The bits extraction procedure enables to homomorphically extract the bits blocks of the plaintext from LWE ciphertexts and reduces the errors simultaneously. Given positive integers $d, \delta, \delta'$, the bits extraction procedure homomorphically extracts $d$ bits at the position $\delta$ (counting from the least significant bit) of the message, and puts it into the position $\delta'$. With this algorithm, one could easily drop the undesired bits
and refresh the effective bits. Let $Z_{2d}$ lie in $[0, 2^d)$, assume $q$ is a power of 2. The algorithm is described in Algorithm 11.

Algorithm 11 BitsExtract$^{d}_{\tilde{f}, N}$(ct, δ, δ′, BK, $\tilde{B}$K, $KSK^{\tilde{Q}}_{\tilde{s} \rightarrow \tilde{s}}$, $KSK^{Q}_{s \rightarrow s}$)

**Parameters:** $d = \log N - \ell_e$, where $\ell_e$ is to be determined; $\tilde{Q}$: an intermediate modulus with $\tilde{Q} > \tilde{Q} > q$; $\tilde{s}$: an intermediate secret key derived from $s$; $\tilde{n}$: an intermediate dimension.

**Input:** ct $\in \text{LWE}^N_{\tilde{n}}(m)$, where $2^d | m$, and error term $|e| < 2^{d-1}$; $\delta$: the position being extracted; $\delta'$: the target position; A bootstrapping key $\tilde{B}$K = $\{\tilde{B}$Ki $\in \text{RGSW}^N_{\tilde{s}, Q}(\tilde{s}_i)\}_{i=0}^{\tilde{n} - 1}$; A bootstrapping key $\tilde{B}$K = $\{\tilde{B}$Ki $\in \text{RGSW}^N_{s, \tilde{Q}}(\tilde{s}_i)\}_{i=0}^{\tilde{n} - 1}$; Two key-switching keys $KSK^{\tilde{Q}}_{\tilde{s} \rightarrow \tilde{s}'}$, $KSK^{Q}_{s \rightarrow s'}$.

**Output:** ct$'$ $\in \text{LWE}^N_{s}(m')$, where $m' = 2^d | m^* |_{2d}$, and $m^* = m / 2^d$;
1: Let $r = \log q - \delta + \ell_e$.
2: Let ct$'$ = DimS$_{N \rightarrow \tilde{n}}$(ct, KSK$^{\tilde{Q}}_{\tilde{s} \rightarrow \tilde{s}'}$)
3: Let ct = ModR$_{s \rightarrow N}$ (ModS$_{s \rightarrow \tilde{n}}$(ct$'$)) // ct = LWE$^N_{\tilde{n}}(2^d | m^* |_{2d})$
4: Perform FullyFBS$^{(\tilde{s}, N) \rightarrow (N, \tilde{n})}$(ct, f, BK, $\tilde{B}$K, KSK$^{\tilde{Q}}_{\tilde{s} \rightarrow \tilde{s}'}$) with $f$ being

\[
f : \mathbb{Z}_N \rightarrow \mathbb{Z}
\]

\[
x \mapsto 2^{\delta'} \cdot \left[\frac{x}{2^{\ell_e}}\right]_{2^d},
\]

obtaining ct$'$ = LWE$^N_{\tilde{n}}(m')$;
5: return ct$'$.

**Remark 8.** As shown in our implementation, we could extract about $(\log N) / 2$ bits once. $\ell_e \approx (\log \tilde{n}) / 2$ bits are reserved to handle the errors to make sure the effective bits in the plaintext are not destroyed. Algorithm 11 could be extended for BFV-like ciphertexts, the analysis is similar to Remark 3 and 6.

**Remark 9.** Note that the constraint on $m$ such that $2^d | m$ is dedicated for our exact truncation algorithm, which is required for IND-CPA$^D$ security. This constraint could be removed if only approximate truncation is needed, as in the IND-CPA case.

**Remark 10.** In our setting, we also need to preserve the higher bits when move the bits blocks into lower positions. This is crucial in our truncation algorithm, when handling negative numbers. Thus, define an algorithm SignBitsExtract$^{d}_{(s, N)}$ be the same as in Algorithm 11 except that $f$ is defined as

\[
f(x) = \begin{cases} 2^{\delta'} \cdot \left[\frac{x}{2^\ell_e}\right]_{2^d}, & x \in [0, \frac{N}{2} - 2^{\ell_e-1}) \cup [N - 2^{\ell_e-1}, N) \\ q - 2^{\delta'+d} + 2^{\delta'} \cdot \left[\frac{x}{2^\ell_e}\right]_{2^d}, & x \in [\frac{N}{2} - 2^{\ell_e-1}, N - 2^{\ell_e-1}). \end{cases}
\]

If the most significant bit of the extracted $d$ bits is 1 (i.e., negative), then the extraction algorithm will preserve the two’s complement representation of this $d$-bits number.

**Theorem 3.** Suppose BK contains $2l$ RLWE ciphertexts with error variance $\sigma_{bk}^2$. BK contains $2\ell$ RLWE ciphertexts with error variance $\sigma_{bk}^2$, KSK$^{\tilde{Q}}_{\tilde{s} \rightarrow \tilde{s}'}$ contains $2\ell$ RLWE ciphertexts with error variance $\sigma_{ck}^2$, KSK$^{Q}_{s \rightarrow s'}$ contains $2\ell$ RLWE ciphertexts with error variance $\sigma_{ck}^2$. If base power gadget is used in BK, $g$ is the decomposition base. If RNS gadget is used then $g = \max\{g_0, \ldots, g_{\ell-1}\}$. $g_{\ell}, g_{\ell_{\text{RNS}}}, g_{\text{RNS}}$ are similarly defined in BK, KSK$^{\tilde{Q}}_{\tilde{s} \rightarrow \tilde{s}'}$ and KSK$^{Q}_{s \rightarrow s'}$.
Let \( ct \in \text{LWE}_{\kappa,q}(m) \) with \( 2^d | m \) and error variance \( \sigma^2 \), \( q \) is a power of 2. Let

\[
\tau = 2^{2(\ell_1 - \delta)}(Nt_{\kappa,k}\|g\|_k\sigma_{\kappa,k}^2 + \sigma^2) + \left(\frac{2N}{Q}\right)^2(Nt_{\kappa,k}\|g\|_k\sigma_{\kappa,k}^2 + 2\bar{m}N\|g\|_b\sigma_b^2) + \frac{n}{12t} + \frac{1}{3}.
\]

If \( 6\sqrt{7} + \frac{1}{2} < 2^{\ell_1 - 1} \). Then \( ct' \) output in Algorithm 11 is a ciphertext in \( \text{LWE}_{\kappa,q}(m') \) with error variance \( \sigma'^2 \) and satisfies

\[
\sigma'^2 \leq 2(q/Q)^2\bar{m}N\|g\|_b\sigma_b^2 + N/24 + 1/6.
\]

**Proof.** According to Lemma 4, the error variance \((\sigma'))^2\) of \( ct' \) is bounded by \( Nt_{\kappa,k}\|g\|_k\sigma_{\kappa,k}^2 + \sigma^2 \). After running the modulus switching procedure in step 3, \( ct' \) is transferred into a ciphertext encrypting \( \lfloor \frac{N}{q}m \rfloor \) and the error variance is bounded by \( (2^d/q)^2 \cdot (\sigma')^2 + \bar{m}/24 + 1/6 \). We have

\[
\lfloor (2^d/q)m \rfloor = \frac{2^d \cdot m + e'}{q} = \frac{2^d \cdot m + e'}{q} = 2^d \cdot m' + e'
\]

where \( e' \) is the rounding error. Let \( e'' \) be the error term after modulus switching, and this error term stays the same after modular reduction as long as \(|e''| \leq N\), which is guaranteed by the chosen parameters. Denote \( e'' = e' + e'' \). Since \( d = \log N - \ell_1 \), then \( (2^d \cdot m') \mod N = 2^d \cdot [m']_{2^d} \). Thus \( ct \) is a ciphertext in \( \text{LWE}_{\kappa,q}(2^d [m']_{2^d} + e') \) with error term 0.

Due to Algorithm 10 and Theorem 2, we know that \( ct' \) is a ciphertext in \( \text{LWE}_{\kappa,q}(f(2^{2\ell_1} [m']_{2^d} + e' + \bar{c})_N) \) with error variance \( \sigma'^2 \leq 2(q/Q)^2\bar{m}N\|g\|_b\sigma_b^2 + N/24 + 1/6 \). Where \( \bar{c} \) has variance

\[
\bar{c}^2 \leq (2N/Q)^2(Nt_{\kappa,k}\|g\|_k\sigma_{\kappa,k}^2 + 2\bar{m}N\|g\|_b\sigma_b^2) + \bar{m}/24 + 1/6.
\]

Therefore, we have \(|e'' + \bar{c}| \leq 6\sqrt{7}, \) which implies \(|e'' + \bar{c}| \leq |e'' + \bar{c}| + 1/2 < 2^{\ell_1 - 1} \). By the definition of \( f \), we have for some \( k \in \mathbb{Z} \) such that

\[
f\left(\lfloor 2^{2\ell_1} [m']_{2^d} + e' + \bar{c} \rfloor_N\right) = f\left(2^{2\ell_1} [m']_{2^d} + e' + \bar{c} + kN\right)
\]

\[
= 2^{2\ell_1} \cdot \left[\left(\lfloor 2^{2\ell_1} [m']_{2^d} + e' + \bar{c} + kN\rfloor / 2^{2\ell_1}\right)\right]_{2^d}
\]

\[
= 2^{2\ell_1} \cdot \left([m']_{2^d} + k \cdot 2^d + [e' + \bar{c}] / 2^{2\ell_1}\right)_{2^d}
\]

\[
= 2^{2\ell_1} \cdot \left([m']_{2^d} + k \cdot 2^d\right)_{2^d} = 2^{2\ell_1} \cdot [m']_{2^d}.
\]

This completes the proof.

**Corollary 1.** Assuming the conditions hold in Theorem 3. Then the extraction algorithm defined in Remark 10 SignBitsExtract_{\ell,\delta}(ct, \delta, \delta', BK, BK, KSK_{\kappa,s+2^d}, KSK_{q-s+2^d}) outputs a ciphertext \( ct' \) encrypts \( m' \) such that \( m' = 2^{2\ell_1} [m']_{2^d} \) if \( 0 \leq 2^{2\ell_1} [m']_{2^d} < N/2 \), and \( m' = q - 2^{2\ell_1 + d} + 2^{2\ell_1} [m']_{2^d} \), if \( N/2 \leq 2^{2\ell_1} [m']_{2^d} < N \).

**Proof.** Similar to the analysis of Theorem 3, we have the ciphertext \( ct \) encrypts plaintext \( 2^{2\ell_1} [m']_{2^d} \) with error term \( e' + \bar{c} \), where \(|e' + \bar{c}| < 2^{\ell_1 - 1} \). The fully functional bootstrapping returns a ciphertext of \( f(2^{2\ell_1} [m']_{2^d} + e' + \bar{c})_N \) with \( f \) defined in Remark 10.

If \( 0 \leq 2^{2\ell_1} [m']_{2^d} < N/2 \), then \( 0 \leq 2^{2\ell_1} [m']_{2^d} \leq 2^{2\ell_1} (2^{2\ell_1 - 1} - 1) = N/2 - 2^{\ell_1} \). Since \(|e' + \bar{c}| < 2^{\ell_1 - 1} \), then

\[
\left[2^{2\ell_1} [m']_{2^d} + e' + \bar{c}\right]_N \in \left[0, \frac{N}{2} - 2^{\ell_1 - 1}\right) \cup [N - 2^{\ell_1 - 1}, N),
\]

14
Corollary 2. Assuming the conditions hold in Theorem 3, except that the input message \( m \parallel 2^d \) in Algorithm 11. Let \( m = \bar{m} \cdot 2^d + \tilde{m} \), where \( \bar{m} \) and \( \tilde{m} \) are integers and \( \bar{m} = [m]_{2^d} \). Then, \( \text{ct}^f \) is a ciphertext of \( 2^d \cdot [\bar{m} + \tilde{e}]_{2^d} \) with error variance \( \sigma^2 \) for some integer \( \tilde{e} \in \{0, 1\} \). Further, we have

\[
\sigma^2 = 2(q/Q)^2\ell N g^2 \sigma^2_{\beta k} + N/24 + 1/6.
\]

In other words, if \( \bar{m} \in \mathbb{Z}_{2^d} \), then the only possible case that \( [\bar{m} + \tilde{e}]_{2^d} \neq \bar{m} + \tilde{e} \) is \( \bar{m} = 2^d - 1 \).

Proof. Similar to Theorem 3, the message \( m^* \) is now defined as \( m^* = \bar{m} + \tilde{m}/2^d \). Then the fully functional bootstrapping procedure in Algorithm 11 results in a ciphertext that encrypts

\[
2^d \left[ \left[ \bar{m} + \tilde{m}/2^d + (e^* + \tilde{e})/2^{f_e} \right] \right]_{2^d} = 2^d \left[ \left[ \bar{m} + \tilde{m}/2^d + (e^* + \tilde{e})/2^{f_e} \right] \right]_{2^d}
\]

Let \( \tilde{e} = \lfloor \tilde{m}/2^d + (e^* + \tilde{e})/2^{f_e} \rfloor \). Note that \( 0 < \tilde{m}/2^d < 1 \) and \( |e^* + \tilde{e}|/2^{f_e} < 1/2 \), then \(-1/2 < \tilde{m}/2^d + (e^* + \tilde{e})/2^{f_e} < 3/2 \). We have \( \tilde{e} \in \{0, 1\} \). The analysis of \( \sigma^2 \) is the same as in Theorem 3. \( \square \)

4.1 Homomorphic Truncation

With algorithm 11, we could perform the truncation by repeatedly extracting the last \( d \) bits of the current message \( m \) and subtract it from \( m \). Recall that the truncation algorithm takes as input an integer \( \Delta^2 \cdot m \), where \( m \in \mathbb{FP}_{2p} \) and \( \log \Delta > 2p \), and outputs \( \Delta \cdot \text{Trunc}_p(m) \). It drops the last \( p \) bits of \( m \), and rescale the factor from \( \Delta^2 \) into \( \Delta \).

In the algorithm, we assume \( d | p \) and \( d | (\log q - 2\log \Delta + p) \) for simplicity, the general case is essentially the same. The input of this algorithm takes as input a RLWE ciphertext that only encrypts the scaled fixed-precision number as the constant term. It is easy to extend the packing version of RLWE ciphertexts. Details are given in Algorithm 12.
In other words, it only refreshes the errors. In this algorithm, it only performs step 7-14, the position complexity will increase.

Remark 11. In step 9, only the SimpleFBS procedure is applied, and the modulus switching procedure is applied in step 10. This slight modification is to manage the errors and keep them small.

Remark 12. It is easy to modify Algorithm 12 into a simplified version HomTrunc_p'. Its input ciphertext is \( \text{ct} \in \text{RLWE}_{\sigma}^{\Delta m}(\Delta m') \) with \( m \in \mathbb{F}_{p^2} \) and the plaintext in the output ciphertext is exactly the same. In other words, it only refreshes the errors. In this algorithm, it only performs step 7-14, the position \( \gamma - \log \Delta + i \cdot d \) is replaced with \( \gamma + i \cdot d \) and the \text{SignBitsExtract}_d^{\ell}(s, N) \) algorithm is replaced with the original \text{BitsExtract}_d^{\ell}(s, N) \) algorithm. This is because we do not need to drop bits blocks, and the position of the effective bits are not changed.

Remark 13. Algorithm 12 could be extended to the coefficient packing version of RLWE ciphertexts. The message is a ring element whose coefficients are in the form of \( \Delta^2 m_i \) for \( 0 \leq i \leq N - 1 \) and \( m_i \in \mathbb{F}_{p^2} \).

Theorem 4. Suppose BK contains \( 2\ell \) RLWE ciphertexts with error variance \( \sigma_{bk}^2 \). BK contains \( 2\ell \) RLWE ciphertexts with error variance \( \tilde{\sigma}_{bk}^2 \). BK contains \( 2\ell \) RLWE ciphertexts with error variance \( \sigma_{k}^2 \).

Algorithm 12 HomTrunc_p(\text{ct}, BK, B̄K, KSK_\tilde{\sigma}_{2→2}, KSK_\sigma_{2→2}, AK)

Parameters: \( d = \log N - \ell_c; \)
\( p \): The amount of bits being dropped, assuming \( dp \).

Input: \( \text{ct} \in \text{RLWE}_{\sigma}^{\Delta m}(\Delta^2 m) \), where \( m \in \mathbb{F}_{p^2} \), and the error satisfies \( |e| \leq \frac{\Delta^2}{2^{dp}} \);

A bootstrapping key BK = \{BK_i \in \text{RGSW}_{\sigma}^{\Delta m}(\tilde{\sigma}_i) \}_{i=0}^{\Delta - 1};
A bootstrapping key B̄K = \{B̄K_i \in \text{RGSW}_{\sigma}^{\Delta m}(\tilde{\sigma}_i) \}_{i=0}^{\Delta - 1};

Two key-switching keys KSK_\tilde{\sigma}_{2→2}, KSK_\sigma_{2→2};
An automorphism key AK.

Output: \( \text{ct}' \in \text{RLWE}_{\sigma}^{\Delta m}(\Delta m') \), where \( m' = \text{Trunc}_p(m) \in \mathbb{F}_{p} \)

1. Let acc = \text{Extract}(ct) \hspace{1cm} \triangleright acc \in \text{LWE}_{\sigma}^{\Delta m}(\Delta^2 m)
2. Denote \( \eta = 2\log \Delta - 2p \)
3. for \( i = 0, 1, \ldots, \frac{p}{2} - 1 \) do
4. Let cbits = \text{BitsExtract}_d^{\ell}((\text{acc}, \eta + i \cdot d, \eta + i \cdot d, BK, B̄K, KSK_\tilde{\sigma}_{2→2}, KSK_\sigma_{2→2})
5. acc = acc - cbits
6. end for
7. Let \( \text{ct}^* = (0, 0) \in \text{LWE}_{\sigma}^{\Delta m}(0), \kappa = \eta + p \) and \( \gamma = \log q - d \).
8. for \( i = 0, 1, \ldots, (\log q - \kappa)/d - 2 \) do
9. Let cbits = \text{BitsExtract}_d^{\ell}((\text{acc}, \kappa + i \cdot d, \kappa - \log \Delta + i \cdot d, BK, B̄K, KSK_\tilde{\sigma}_{2→2}, KSK_\sigma_{2→2})
   \hspace{1cm} \triangleright \text{Here in BitsExtract, SimpleFBS is applied, i.e. cbits} \in \text{LWE}_{\sigma}^{\Delta m}.
10. acc = acc - ModS_{\Delta q}(\Delta \cdot \text{cbits})
11. \text{ct}^* = \text{ct}^* + \text{cbits}
12. end for
13. \text{cbits} = \text{SignBitsExtract}_d^{\ell}(\text{acc}, \gamma - \log \Delta, BK, B̄K, KSK_\tilde{\sigma}_{2→2}, KSK_\sigma_{2→2})
   \hspace{1cm} \triangleright \text{Preserve the two's complement representation.}
14. \text{ct}^* = \text{ct}^* + \text{cbits}.
15. return \( \text{ct}' = \text{ModS}_{\Delta q}(\text{LWE-to-RLWE}(\text{ct}^*, AK)) \)

16
used then \( g = \max\{g_0, \ldots, g_{t-1}\} \). \( \bar{g}, g_{kak}, g_{kak} \) and \( g_{ak} \) are similarly defined in \( \bar{\mathcal{K}}, KSK_{s'\rightarrow s'}, KSK_{s'\rightarrow s'} \) and \( \mathcal{K} \).

If \( ct \in RLWE_{s,q}^{N}(\Delta^2m) \) with error variance \( \sigma^2 \), where \( m \in \mathbb{F}_{2p} \). Let

\[
\tau_{1,k} = 2^{2(\ell_{i} - 2 \log \Delta + 2p - i - d)} \left( N^2 g_{kak}^2 g_{kak}^2 k + \left( 2(\frac{q}{Q})^2 \bar{g} \bar{N} g_{kak}^2 \sigma_{bk}^2 + \frac{N}{24} + \frac{1}{6} \right) + \sigma^2 \right) + \left( \frac{2N}{Q} \right)^2 \left( N^2 g_{kak}^2 g_{kak}^2 2\bar{g} \bar{N} g_{kak}^2 2\sigma_{bk}^2 + \frac{n}{12} + \frac{1}{3} \right),
\]

where \( 0 \leq k \leq p/d - 1 \). Let

\[
\tau_{2,t} = 2^{2(\ell_{i} - \log \Delta - t - d)} \left( N^2 g_{kak}^2 g_{kak}^2 + \frac{p}{d} \left( 2(\frac{q}{Q})^2 \bar{g} \bar{N} g_{kak}^2 \sigma_{bk}^2 + \frac{N}{24} + \frac{1}{6} \right) + \right. \]

\[
\left. t \cdot \left( 2(\frac{q}{Q})^2 \bar{g} \bar{N} g_{kak}^2 \sigma_{bk}^2 + \frac{N}{24} + \frac{1}{6} \right) + \sigma^2 \right) + \left( \frac{2N}{Q} \right)^2 \left( N^2 g_{kak}^2 g_{kak}^2 2\bar{g} \bar{N} g_{kak}^2 2\sigma_{bk}^2 + \frac{n}{12} + \frac{1}{3} \right),
\]

where \( 0 \leq t \leq \log \frac{q - k}{d} - 1 \).

Suppose \( \theta_{\text{max}} \{ \{ \tau_{1,k} \}, \{ \tau_{2,t} \} \} + 1/2 < 2^{\ell_{i} - 1}, q \) is a power of 2, \( d/p, d/(\log q - k) \) and \( \log \Delta > p \). Then \( ct' \) output in Algorithm 12 is a ciphertext in \( RLWE_{s,q}^{N}(\Delta m') \) with error variance \( \sigma'^2 \), where \( m' = \text{Trunc}_{p}(m) \). Moreover,

\[
\sigma'^2 = \left( \frac{q}{Q} \right)^2 \left( \bar{g} \bar{N} g_{kak}^2 \sigma_{bk}^2 + \frac{\log q - \bar{\kappa}}{d} \cdot 2\bar{g} \bar{N} g_{kak}^2 \sigma_{bk}^2 + \frac{N}{24} + \frac{1}{6} \right).
\]

Proof. Since \( m \in \mathbb{F}_{2p} \) and \( \log \Delta > p \), we know that \( 2^p | \Delta^2 m \). We consider two cases: \( \Delta^2 m \geq 0 \) and \( \Delta^2 m < 0 \). For the first case, split \( \Delta^2 m \in \mathbb{Z}_{\geq 0} \) into bits blocks with each block \( d \) bits,

\[
\Delta^2 m = 2^q m_0 + 2^q m_1 + \cdots + 2^q m_{\frac{d}{2} - 1} + 2^q m_\frac{d}{2} + \cdots + 2^q m_{\frac{d}{2} + \left( \frac{q - \log q}{d} - 1 \right) d} + 2^q m_{\frac{d}{2} + \left( \frac{q - \log q}{d} - 1 \right) d},
\]

where \( 0 \leq m_i < 2^d \). Consider the loop from step 3 to step 6. For \( 0 \leq i \leq p/d - 1 \), let \( \text{cbits}_i \) be the output of the \( i \)-th iteration of step 4 and \( \sigma_i^2 \) be the error variance. Let \( \text{acc}_i \) be the output of the \( i \)-th iteration of step 5 and \( \sigma_i^2 \) be the error variance. We claim \( \text{cbits}_i \) is a ciphertext of \( 2^{q + d} \cdot m_i \) and \( \sigma_i^2 \leq 2(q/Q)^2 \bar{g} \bar{N} g_{kak}^2 \sigma_{bk}^2 + N/24 + 1/6 \). \( \text{acc}_i \) is a ciphertext of \( \Delta^2 m - \sum_{j=0}^{i} 2^q m_j \) and \( \sigma_i^2 \leq \sigma^2 + (i + 1) \cdot \sigma_0^2 \).

According to the chosen parameters and Theorem 3, \( \text{cbits}_0 \) is a ciphertext of \( 2^q m_0 \) and \( \sigma_0^2 \leq 2(q/Q)^2 \bar{g} \bar{N} g_{kak}^2 \sigma_{bk}^2 + N/24 + 1/6 \). \( \text{acc}_0 \) is a ciphertext of \( \Delta^2 m - 2^q m_0 \) and \( \sigma_0^2 \leq \sigma^2 + \sigma_0^2 \).

Suppose \( \text{cbits}_i \) and \( \text{acc}_i \) satisfies the above properties, we have \( \sigma_i^2 \leq \sigma^2 + (i + 1) \cdot \sigma_0^2 \) for \( 0 < i < p/d - 1 \). Then due to the chosen parameters and the constraint of \( \{ \tau_{1,k} \} \), it is easy to verify that the constraint of \( \tau \) in Theorem 3 is satisfied. Invoke Algorithm 11 on \( \text{acc}_i \), will obtain \( \text{cbits}_{i+1} \), which is a ciphertext of \( 2^{q + (i + 1) d} \cdot N^2 m - \sum_{j=0}^{i} 2^{q + j d} \cdot m_j \), and the error variance \( \sigma_{i+1}^2 = \sigma_i^2 \leq 2(q/Q)^2 \bar{g} \bar{N} g_{kak}^2 \sigma_{bk}^2 + N/24 + 1/6 \) due to Theorem 3. Therefore, \( \text{acc}_{i+1} \) is a ciphertext of \( \Delta^2 m - \sum_{j=0}^{i+1} 2^{q + j d} \cdot m_j \) and the error variance \( \sigma_{i+1}^2 \leq \sigma^2 + (i + 2) \cdot \sigma_0^2 \).

After step 6, \( \text{acc} \) is a ciphertext of \( \Delta^2 m' \) with error variance less than \( \sigma^2 + (p/d) \cdot \sigma_0^2 \). The following steps will refresh \( \text{acc} \) into a ciphertext \( \text{ct}' \) of \( [(Q/q) \Delta m'] \) with error variance independent of \( \sigma \). The
analysis from step 8 to step 14 is similar as above, because the `SignBitsExtract` algorithm outputs the same as `BitsExtract`. For $0 \leq i \leq (\log q - \kappa)/d - 1$, the $i$-th iteration of step 9 outputs a ciphertext $c_b$, which encrypts $[(Q/q)^{2\log \Delta - p + i/d} \cdot m_{\bar{z}_{i+1}}]$, this is because the modulus switching procedure is not performed in the functional bootstrapping procedure. The $i$-th iteration of step 10 outputs a ciphertext of $\Delta^2 m' - \sum_{j=0}^{i} 2^{\gamma + p + i/d} m_{\bar{z}_{i+j}} = \Delta^2 m' - \Delta (\sum_{j=0}^{i} 2^{\gamma p + i/d} m_{\bar{z}_{i+j}})$. Therefore, the final $ct^*$ is a ciphertext of $[(Q/q)\Delta m']$. The LWE-to-RLWE algorithm and modulus switching procedure convert $ct^*$ into an RLWE ciphertext that encrypts $\Delta m'$.

For the case that $\Delta^2 m < 0$, the message is in the form $q + \Delta^2 m > 0$. Since $2^p|q$, then $2^p|(q + \Delta^2 m)$. In this case, the most significant bit of the highest $d$-bits in step 13 is 1. Due to Corollary 1, the message contains an additional term $q - 2^{\gamma - \log \Delta - d}$. Combining with the analysis of the first case, the resulting message is of the form

$$q - 2^{\gamma - \log \Delta - d} + \Delta \left(\text{Trunc}_p \left( \frac{q + \Delta^2 m}{\Delta^2} \right) \right) = q - \frac{q}{\Delta} + \Delta \left( \frac{2^p (q + \Delta^2 m)}{\Delta^2} \right) / 2^p = q + \Delta \left( \frac{2^p \cdot m}{\Delta^2} \right) / 2^p = q + \Delta \cdot \text{Trunc}_p(m),$$

which is desired.

Considering the error variance, due to Theorem 3 without using modulus switching, we know that the error variance of the final $ct^*$ satisfies $(\sigma^*)^2 \leq (\log q - \kappa)/d \cdot 2\ell n Q g^2 \sigma_0^2$, which is independent of $\sigma$. By Lemma 5 and 2, the LWE-to-RLWE and modulus switching procedures ensure that $ct'$ is a ciphertext of $\Delta m'$ with error variance

$$\sigma'^2 \leq \left( \frac{q}{Q} \right)^2 \left( \ell_{\text{sk}} N^2 q_{\text{sk}}^2 \sigma_{\text{sk}}^2 + \frac{\log q - \kappa}{d} \cdot 2\ell n N g^2 \sigma_0^2 \right) + N/24 + 1/6.$$

It remains to prove that the parameters are satisfied the constraints in Theorem 3 when invoking the bits extraction procedure in step 9. After step 6, $acc$ is a ciphertext of $\Delta^2 m'$ with error variance less than $\sigma^2 + (p/d) \cdot \sigma_0^2$. In the $i$-th iteration, the error variance of $acc$ is less than

$$\sigma^2 + (p/d)\sigma_0^2 + \sigma_0^2 \cdot \left( 2(q\Delta/Q)^2 \ell n N g^2 \sigma_0^2 + N/24 + 1/6 \right).$$

By the choice of $\{\tau_{2,t}\}_t$ for $0 \leq t \leq (\log q - \kappa)/d - 1$, the constraint in Theorem 3 is satisfied. This completes the proof. □

**Approximate Homomorphic Truncation.** The homomorphic truncation algorithm is dedicated for fixed-precision numbers to achieve IND-CPA security. When approximate encryption for fixed-point or floating-point numbers is considered, we could modify the homomorphic truncation algorithm into an approximate version. The resulting fully homomorphic encryption scheme is only IND-CPA secure.

In the exact homomorphic truncation algorithm, $p$ is much smaller than $\log \Delta$. This is because we have to reserve enough space for the errors that will not destroy the effective bits. In the worst case, the bit length of the error produced in the homomorphic operations could not exceed $2\log \Delta - 2p$. In the approximate homomorphic truncation algorithm, $p$ could be very close to $\log \Delta$. The bits extraction algorithm begins at the position $2\log \Delta - p$ instead of $2\log \Delta - 2p$, this is where the approximate value occurs. Details are given in Algorithm 13.
Algorithm 13  ApproxTruncₚ(ct, BK, BK, KSKₗ₋₂, KSKₗ₋₂, AK)

**Parameters:**  
\( d = \log N - \ellₙ; \)  
\( p: \) A parameter related to precision, assuming \( dp \).

**Input:**  
\( ct \in \text{RLWE}^N_q(\lfloor \Delta^2m \rfloor), \) where \( m \in \mathbb{R} \);  
A bootstrapping key \( BK = \{BK_i \in \text{RGSW}^{N}_{q, Q}((s_i))\}_{i=0}^{\tilde{\alpha}-1} \);  
A bootstrapping key \( BK = \{BK_i \in \text{RGSW}^{N}_{q, Q}((s_i))\}_{i=0}^{\tilde{\alpha}-1} \);  
Two key-switching keys \( \text{KSK}^Q_{\rightarrow \rightarrow 1}, \text{KSK}^Q_{\rightarrow \rightarrow 2}; \)  
An automorphism key \( AK \).

**Output:**  
\( ct' \in \text{RLWE}^N_q(\Delta m'), \) where \( \Delta m' \in \mathbb{Z} \) and \( m' \approx m \).

1: Let \( \text{acc} = \text{Extract}(ct) \land \text{acc} \in \text{LWE}^N_q(\lfloor \Delta^2m \rfloor) \)
2: Denote \( \eta = 2\log \Delta - 2p \) and \( \kappa = \eta + p, \gamma = \log q - d. \)
3: Let \( \text{ct}^* = (0, 0) \in \text{LWE}^N_q(0) \).
4: for \( i = 0, 1, \ldots, (\log q - \kappa)/d - 2 \) do  
5: Let \( \text{cbits} = \text{BitsExtract}^d_{(s,N)}(\text{acc}, \kappa + i \cdot d, \kappa - \log \Delta + i \cdot d, BK, BK, KSK^Q_{\rightarrow \rightarrow 2}, KSK^Q_{\rightarrow \rightarrow 2}) \)  
6: \( \text{acc} \leftarrow \text{acc} - \text{ModSQ}_{-q}(\Delta \cdot \text{cbits}) \)
7: \( \text{ct}^* \leftarrow \text{ct}^* + \text{cbits} \)
8: end for  
9: \( \text{cbits} = \text{SignBitsExtract}^d_{(s,N)}(\text{acc}, \gamma, \gamma - \log \Delta, BK, BK, KSK^Q_{\rightarrow \rightarrow 2}, KSK^Q_{\rightarrow \rightarrow 2}) \)  
10: \( \text{ct}^* \leftarrow \text{ct}^* + \text{cbits} \)
11: return \( \text{ct}' = \text{ModSQ}_{-q}(\text{LWE-to-RLWE}(\text{ct}^*, AK)) \)

**Remark 14.** Algorithm 13 could be extended to a simplified version \( \text{ApproxTrunc}^c_{\tilde{p}} \). Its input ciphertext is \( \text{ct} \in \text{RLWE}^N_q(\lfloor \Delta m \rfloor) \) with \( m \in \mathbb{R} \) and the output ciphertext is of the form \( \text{ct}' \in \text{RLWE}^N_{q, d}(\Delta m') \). In this algorithm, the position \( \kappa - \log \Delta + i \cdot d \) is replaced with \( \kappa + i \cdot d \) and the \( \text{SignBitsExtract}^d_{(s,N)} \) algorithm is replaced with the original \( \text{BitsExtract}^d_{(s,N)} \) algorithm.

**Theorem 5.** Let all the conditions holds as in Theorem 4, then \( \text{ct}' \) is a ciphertext of \( \Delta m' \) with error variance \( \sigma'^2 \). Moreover, we have \( |\Delta m' - \Delta m| \leq \frac{d}{8} + \frac{1}{2}, \) and  
\[ \sigma'^2 \leq \left( \frac{q}{Q} \right)^2 \cdot \left( \ellₙN^2g^2σ^2_{\text{sk}} + \frac{\log q - \kappa}{d} \cdot 2\ellₙN^2g^2σ^2_{\text{sk}} \right) + \frac{N}{24} + \frac{1}{6}. \]

**Proof.** Similar to the proof of Theorem 4, we could write \( |\Delta^2m| \) in the form  
\[ |\Delta^2m| = m^* + 2q + p + \cdots + 2q + p + \frac{(q^2 - q^2 - 1)}{2} \cdot m \log \frac{q - \eta}{d}, \]  
where \( 0 \leq m^* < 2q + p, 0 \leq m_{\log \frac{q - \eta}{d}} < 2d \) for \( 0 \leq i < (\log q - \eta)/d. \)

Denote \( M \) as the event that  
\[ \exists i \in \left[ 0, \frac{\log q - \eta - p}{d} \right], \text{ s.t. } 0 \leq j \leq i, \quad m_{\log \frac{q - \eta}{d}} = 2^d - 1. \]

If \( M \) does not happen, i.e., there is no consecutive \( m_{\log \frac{q - \eta}{d}} \)'s that equal to \( 2^d - 1 \) starting from the first block, then according to Corollary 2 and Theorem 4, we have the output ciphertext encrypts \( \Delta m' \). Thus \( |\Delta^2m| - |\Delta^2m'| = m^* < 2q + p, \) which implies \( |\Delta^2m - \Delta^2m'| < 2q + p + |\epsilon| < 2q + p + 1/2, \) where \( \epsilon \) is the rounding error. We get that \( |\Delta m - \Delta m'| < \frac{d}{8} + \frac{1}{24}. \)
If \( M \) happens for some \( i^* \), we have \( m_{\xi+j} = 2^d - 1 \) for all \( 0 \leq j \leq i^* \). Due to Corollary 2, we know that the BitsExtract algorithm will obtain ciphertexts of 0 when extracting \( m_{\xi+j} \) for \( 0 \leq j \leq i^* \). The extracted ciphertext of the block \( m_{\xi+(i^*+1)} \) is of the form \((m_{\xi+(i^*+1)})^d + 1\). Since the \((i^*+1)\)-th block does not overflow, then all the remaining message blocks will be extracted exactly. This is because subtracting \((m_{\xi+(i^*+1)})^d + 1\) will reserve enough 0’s (in fact \( d - 1 \) 0’s) for the next extraction. Thus we have

\[
|\Delta m - \Delta m'| = |m^* + \sum_{j=0}^{i^*} (2^d + j \cdot 2^d - 1) - 2^d + (i^*+1) \cdot 2^d |
\]

\[
= |m^* + 2^d \cdot 2^d - 1 \cdot \sum_{j=0}^{i^*} 2^d - 2^d + (i^*+1) \cdot 2^d |
\]

\[
= |m^* + 2^d \cdot 2^d - 2^d - 2^d + (i^*+1) \cdot 2^d |
\]

\[
= |m^* - 2^d|
\]

Since \( 0 \leq m^* < 2^{d+1} \), thus we have \( |\Delta m - \Delta m'| < \frac{1}{2d} + \frac{1}{d} \). The analysis of \( \sigma' \) is exactly the same as in Theorem 4. This completes the proof. \( \square \)

**Remark 15.** Note that \( p \approx \log \Delta \) in this case, and the resulting message \( m' \) is very close to \( m \). Therefore, the bootstrapping error dominates the approximation error in this procedure. This is the main reason our IND-CPA scheme preserves high precision while keeping the dimension small.

**Remark 16.** Note that in the approximate truncation algorithm, the modulus \( q \) is not required to be a power of 2 since approximate errors are allowed. One could choose \( q \) be a product of distinct prime factors and close to a power of 2.

## 5 The TOTA Algorithms

We are ready to describe our FHE scheme for fixed-precision numbers and for fixed-point numbers. Since our approximate version is very similar, we describe them in one algorithm and point out the differences.

The fully homomorphic encryption TOTA consists of the following five algorithms: (\text{TOTA.KeyGen, TOTA.Enc, TOTA.Dec, TOTA.Add, TOTA.Mult}). The plaintext space is defined in \( \mathbb{F}_p \) or \( \mathbb{R} \).

- **\text{TOTA.KeyGen}(\lambda)\)**
  - It takes as input the security parameter \( \lambda \), chooses integers \( N, q, Q, \tilde{Q}, \tilde{n}, \Delta \), where \( N, \tilde{n}, q \) and \( \Delta \) are powers of \( 2 \). Let \( \chi_\sigma \) be the error distribution for encryption and \( \sigma > 0 \) be the standard variance.
  - It samples a secret key \( s \leftarrow \{0, 1\}^N \), \( a \leftarrow \mathbb{R}_q \) and \( c \leftarrow \chi_\sigma \). Let \( s = \text{ToRing}(s) \), set the secret key \( \text{SK} = s \) and the public key as
    \[
    \text{PK} = (a, -as + c) \in \mathbb{R}_q^2.
    \]
  - It generates a key switching key \( \text{RLK} \in \text{RGSW}^N_{s,q}(s^2) \), which will be used to relinearize the ciphertext. It also generates an automorphism key \( \text{AK} = \{K_i \in \text{RGSW}^N_{s,q}(\phi_i(s))\}, i \in \{2^k + 1|k = 1, \ldots, \log N\} \).
  - It samples an intermediate key \( \tilde{s} = \{\tilde{s}_0, \ldots, \tilde{s}_{n-1}\} \leftarrow \{0, 1\}^n \) and let \( \tilde{s} = \text{ToRing}(\tilde{s}) \), generates bootstrapping keys \( \text{BK} = \{\text{BK}_i \in \text{RGSW}^N_{s,q}(\tilde{s}_i)\}_{i=0}^{n-1} \), \( \tilde{\text{BK}} = \{\tilde{\text{BK}}_i \in \text{RGSW}^N_{s,q}(\tilde{s}_i)\}_{i=0}^{n-1} \), and two switching keys \( \text{KSK}^q_{\tilde{s} \rightarrow \tilde{s}} \) and \( \text{KSK}^q_{s \rightarrow \tilde{s}} \) as defined in Algorithm 5.

Let \( \text{EK} = (\text{BK}, \tilde{\text{BK}}, \text{KSK}^q_{\tilde{s} \rightarrow \tilde{s}}, \text{KSK}^q_{s \rightarrow \tilde{s}}, \text{AK}, \text{RLK}) \), it outputs \( (\text{SK}, \text{PK}, \text{EK}) \).
– TOTA.Enc(PK, m). Given m ∈ \mathbb{F}_p (or m ∈ \mathbb{R}) set m' = Δ ⋅ m (or m' = |Δm|). Sample two \mathcal{R} elements e_0, e_1 ← χ_σ and u ← \{0, 1\}^{N_σ}, set u = ToRing(u). Output the ciphertext as PK ⋅ u + (e_0, e_1 + m').

– TOTA.Dec(SK, ct). Given a ciphertext ct = (a, b), if the plaintext is defined in \mathbb{F}_p, it outputs

\[
 m = \frac{1}{2p} \cdot \left[ \frac{2p}{\Delta} (b + a ⋅ s) \right]_0 ∈ \mathbb{F}_p .
\]

If the plaintext is defined in Ω, it outputs m = [(b + as)/Δ] ∈ Ω. Where (⋅)0 denotes the constant term of the ring element.

– TOTA.Add(EK, ct₁, ct₂). Given two ciphertexts ct₁ = (a₁, b₁), ct₂ = (a₂, b₂). It outputs ct' = (a₁ + a₂, b₁ + b₂). If the plaintext is defined in \mathbb{F}_p, it outputs the refreshed ciphertext ct = HomTrunc_p( ct', BK, BK, KSK_{1→2}, KSK_{2→3}, AK). If the plaintext is defined in Ω, it outputs the refreshed ciphertext ct = ApproxTrunc_p (ct', BK, BK, KSK_{1→2}, KSK_{2→3}, AK).

– TOTA.Mult(EK, ct₁, ct₂). Given two ciphertexts ct₁ = (a₁, b₁), ct₂ = (a₂, b₂). It computes (f, g, h) = (a₁a₂, a₁b₂ + a₂b₁, b₁b₂). Relinearize the ciphertext by setting ct' = Keys_{1→2}((0, f), RLK) + (g, h). If the plaintext is defined in \mathbb{F}_p, it outputs ct = HomTrunc_p( ct', BK, BK, KSK_{1→2}, KSK_{2→3}, AK). If the plaintext is in Ω, it outputs ct = ApproxTrunc_p( ct', BK, BK, KSK_{1→2}, KSK_{2→3}, AK).

**Theorem 6.** Suppose BK contains 2l RLWE ciphertexts with error variance \( σ^2_{bk} \). BK contains 2l RLWE ciphertexts with error variance \( σ^2_{bk} \). KSK_{1→2} contains 2l_{sk} RLWE ciphertexts with error variance \( σ^2_{sk} \). KSK_{2→3} contains 2l_{rk} RLWE ciphertexts with error variance \( σ^2_{rk} \). RLK contains \( ℓ_{rk} \) RLWE ciphertexts with error variance \( σ^2_{rk} \). If base power gadget is used in BK, g is the decomposition base. If RNS gadget is used then \( g = \max\{g_0, ..., g_{l−1}\} \). g, g_{sk}, g_{rk}, g_{ak} and g_{ak} are similarly defined in BK, KSK_{1→2}, KSK_{2→3}, AK and RLK.

Let \( σ^2 \) be the error variance of ciphertexts in TOTA. \( τ_{1,k} \) and \( τ_{2,t} \) are defined as in Theorem 4 except that \( σ^2 \) is replaced by \( 3Δ^2 β^2 σ^2_0 + ℓ_{rk} N g^2_{sk} σ^2_{rk} \). Suppose

\[
 σ^2_0 < \max \left\{ \frac{Δ^2}{144 ⋅ 2^p}, \frac{Δ^2}{108 β^2 ⋅ 2^p} - \frac{ℓ_{rk} N g^2_{sk} σ^2_{rk}}{3Δ^2 β^2} \right\}, \quad (N + 1) \cdot σ^2 ≤ σ^2_0,
\]

\[
 \left( \frac{q}{Q} \right)^2 \cdot (ℓ_{sk} N^2 g^2_{ak} σ^2_0 + \frac{log q − k}{d} \cdot 2ℓ_{rk} N g^2 σ^2_{sk}) + \frac{N}{24} + \frac{1}{6} ≤ σ^2_0.
\]

Then TOTA defined in \mathbb{F}_p decrypts the ciphertext correctly and is fully homomorphic.

**Proof.** Let ct = (a, b) ∈ RLWE_{n,s,q}(Δm) be a fresh ciphertext. It is easy to verify that \( b + as = e_0s + e_1 + Δm \). Denote \( e^* = e_0s + e_1 \), we have the variance of \( e^* \) is less than \( (N + 1)σ^2 ≤ σ^2_0 \). When invoking the decryption algorithm on ct, we have TOTA.Dec(SK, ct) = \( \left[ \frac{2p}{\Delta} (b + a ⋅ s) \right] / 2p = \frac{2p}{\Delta} e^* \). Since \( σ^2_0 ≤ Δ^2/(144 ⋅ 2^p) \), then \( |e^*| ≤ 6σ_0 \) and \( \left| \frac{2p}{\Delta} e^* \right| < 1/2 \). TOTA.Dec(SK, ct) = m ∈ \mathbb{F}_p.

Because addition is similar and simple, we only consider multiplication here. Let ct₁ ∈ RLWE_{n,s,q}(Δm₁), ct₂ ∈ RLWE_{n,s,q}(Δm₂) be fresh ciphertexts with error term \( e_1, e_2 \), respectively. Denote the multiplication ciphertext ctₘₐₓₜ = TOTA.Multi(ΕK, ct₁, ct₂). Before relinearizing the ciphertext, the error term of the multiplication is of the form \( Δm₁e₂ + Δm₂e₁ + e₁e₂ \), and the last term is much smaller. Heuristically, the variance is less than \( 3Δ^2 β^2 σ^2_0 + ℓ_{rk} N g^2_{sk} σ^2_{rk} \). In order to invoke HomTrunc_p and decrypt the
exact message, the error term in $ct'$ will not distort the first $2p$ bits of the fraction part. By the setting of $\sigma_0$, it ensures that $6r_{\text{relin}} \leq \Delta^2/2^{2p+1}$.

By the setting of $\tau_1,k$, $\tau_2,l$ and Theorem 4, we have $ct_{\text{mult}}$ is a ciphertext of $\text{Trunc}_p(m_1m_2)$ with error variance $\sigma'^2$. It satisfies

$$\sigma'^2 \leq \left(\frac{q}{Q}\right)^2 \cdot \left(\ell_k N^2 q_j^2 \sigma^2_{\text{sk}} + \frac{\log q - \kappa}{d} \cdot 2\ell_k N q^2 \sigma^2_{\text{sk}}\right) + \frac{N}{24} + \frac{1}{6} \leq \sigma_0.$$ 

This completes the proof. □

Remark 17. The analysis for the approximate encryption is similar, we omit the detailed description.

**Corollary 3.** Suppose all the conditions hold as in Theorem 6, $\text{EK}$ and $\text{PK}$ in $\text{TOTA}$ are indistinguishable from uniformly random, then $\text{TOTA}$ performed on $\mathbb{F}_p$ and $\mathbb{R}$ achieves IND-CPA$^0$ and IND-CPA security, respectively.

**Proof.** It is straightforward from [35]. □

## 6 Parameters and Implementation

We choose concrete parameters for $\text{TOTA}$ and implement it with C++ from scratch. All the parameters are chosen to provide at least 127-bit security according to [1]. For simplicity, we set the magnitude of all the messages to be less than 1. The parameters are given in Table 1. Set I and II are chosen for IND-CPA$^0$ security and Set III, IV and V are chosen for IND-CPA security.

Note that we could choose the dimension $N$ to $2^{13}$ and $2^{12}$ for $p = 14$ and $p = 12$ in the IND-CPA$^0$ case, respectively. In the IND-CPA case, we could use $N = 2^{13}$ and preserve about 39-bit precision. This is the first construction of this kind.

<table>
<thead>
<tr>
<th>Param.</th>
<th>$N$</th>
<th>$q$</th>
<th>$\sigma$</th>
<th>$\ell$</th>
<th>$\Delta$</th>
<th>$\text{EK}$</th>
<th>$\text{BK}$</th>
<th>$\text{KSK}_{\ell+1,2}^0$</th>
<th>$\text{KSK}_{\ell,2}^0$</th>
<th>$\text{AK}$</th>
<th>$p$/precision</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set I</td>
<td>$2^{18}$</td>
<td>$2^{109}$</td>
<td>3.2</td>
<td>$2^{41}$</td>
<td>$2^{51}$</td>
<td>$Q = {54,55,50,50}$</td>
<td>$g : \text{rms}$</td>
<td>$\sigma_{\text{sk}} : 2^7$</td>
<td>$\sigma_{\text{sk}} : 2^7$</td>
<td>$g_{\text{sk}} : \text{rms}$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
</tr>
<tr>
<td>Set II</td>
<td>$2^{12}$</td>
<td>$2^{14}$</td>
<td>3.2</td>
<td>$2^{11}$</td>
<td>$2^{14}$</td>
<td>$Q = {54,54}$</td>
<td>$g : \text{rms}$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$g_{\text{sk}} : 2^7$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
</tr>
<tr>
<td>Set III</td>
<td>$2^{13}$</td>
<td>$39$</td>
<td>3.2</td>
<td>$2^{10}$</td>
<td>$2^{10}$</td>
<td>$Q = {54}$</td>
<td>$g_{\text{sk}} : 2^7$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$g_{\text{sk}} : 2^7$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
</tr>
<tr>
<td>Set IV</td>
<td>$2^{12}$</td>
<td>${54}$</td>
<td>3.2</td>
<td>$2^{11}$</td>
<td>$2^{12}$</td>
<td>$Q = {54}$</td>
<td>$g_{\text{sk}} : 2^7$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$g_{\text{sk}} : 2^7$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
</tr>
<tr>
<td>Set V</td>
<td>$2^{13}$</td>
<td>${54,54}$</td>
<td>3.2</td>
<td>$2^{11}$</td>
<td>$2^{13}$</td>
<td>$Q = {54,55,50,50}$</td>
<td>$g_{\text{sk}} : \text{rms}$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$g_{\text{sk}} : 2^7$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
<td>$\sigma_{\text{sk}} : 3.2$</td>
</tr>
</tbody>
</table>

Table 1. Parameters for $\text{TOTA}$ with different security. The numbers of the modulus mean the bit length of each prime factor. $\text{rms}$ means that we choose the largest generator in the RNS representation. Set I and II set $p = 14$ and $p = 12$, respectively. Set III, IV and V achieves precision about 11, 21 and 39, respectively. Note that in the IND-CPA case, $p$ in the algorithm is not the final precision.

We also test all the parameter sets and the concrete performance is given in Table 2. The experiments are run on a machine with Intel(R) Core(TM) i7-10510U 2304 MHz CPU and 16GB RAM.

As shown in Table 2, we could truncate the plaintext homomorphically in about $77s/50s$ for Set I/II and in about $8s/23s/68s$ for Set III/IV/V. The key size of our construction is relatively large. One could use the methods in [30] to compact the key size and perform the functional bootstrapping procedure on the fly. We leave it to the future work.

Note that the theoretical analysis of the parameters chosen are relatively loose, the concrete parameters chosen are tighter. We also note that $q$ is not required to be a power of 2 in the IND-CPA case.
<table>
<thead>
<tr>
<th>Security</th>
<th>Parameter Set</th>
<th>Truncation time (s)</th>
<th>Public Key Size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IND-CPAD</td>
<td>Set I</td>
<td>77.6</td>
<td>≈ 8.006</td>
</tr>
<tr>
<td></td>
<td>Set II</td>
<td>50.3</td>
<td>≈ 3.007</td>
</tr>
<tr>
<td>IND-CPAD</td>
<td>Set III</td>
<td>8.52</td>
<td>≈ 0.878</td>
</tr>
<tr>
<td></td>
<td>Set IV</td>
<td>23.5</td>
<td>≈ 3.001</td>
</tr>
<tr>
<td></td>
<td>Set V</td>
<td>68.3</td>
<td>≈ 8.004</td>
</tr>
</tbody>
</table>

Table 2. Concrete Performance of Different Parameter Sets

7 Conclusions

We propose TOTA, a fully homomorphic encryption for fixed-precision arithmetic with IND-CPAD security. Along with that, we also extend it into an FHE scheme for approximate encryption with IND-CPAD security. The resulting approximate FHE preserves high precision with only small ring dimension. The underlying core technique is a new fully functional bootstrapping methods that enables to perform arbitrary function with in domain \([0, N]\). We believe our fully functional bootstrapping is of independent interest.

References

Appendix

A Details on RGSW Ciphertext and External Product

The gadget matrix consists of multiple gadget vectors. Let \( g = \{g_0, ..., g_l\} \) be a gadget vector. Two types of gadget vectors are considered. If base power is used, i.e., \( g = \{1, g, g^2, ..., g^{l-1}\} \), we call it a base power gadget, denoted as \( g_{\text{power}} \). For some \( Q = q_0 \times q_1 \times \cdots \times q_{\ell-1} \) with distinct primes, we call \( g \) a RNS gadget, which is denoted as \( g_{\text{RNS}} \), if \( g_i = [(Q/q_i)^{-1}]_q \), for \( 0 \leq i \leq \ell - 1 \).

**Definition 3 (Gadget Matrix).** Let \( g = \{g_0, g_1, ..., g_{\ell-1}\} \subset \mathbb{Z} \) be the gadget vector, then the gadget matrix \( G \) corresponding to it is the following matrix in \( \mathbb{R}^{2\ell \times 2} \):

\[
\begin{pmatrix}
g_0 & 0 \\
\vdots & \\
g_{\ell-1} & 0 \\
0 & g_0 \\
\vdots & \\
0 & g_{\ell-1}
\end{pmatrix}
\]

Also we define the decomposition procedure corresponding to the gadget basis.

**Definition 4.** Let \( g \) be a gadget vector, \( f \in \mathcal{R} \), then define

\[
\text{Decomp}_g(f) = (f_0, ..., f_{\ell-1}) \in \mathcal{R}^\ell,
\]

such that \( \sum f_i g_i = f \). More specifically, if \( g = g_{\text{power}} \), then \( (f_0, ..., f_{\ell-1}) \) is the power decomposition on base \( g \), i.e., \( \sum_{i=0}^{\ell-1} f_i g^i = f \). If \( g = g_{\text{RNS}} \), then \( (f_0, ..., f_{\ell-1}) \) is the RNS decomposition, i.e., \( f_i = f \mod q_i \).

For \((a, b) \in \mathcal{R} \times \mathcal{R} \), define

\[
\text{Decomp}_g((a, b)) = (\text{Decomp}_g(a), \text{Decomp}_g(b)) \in \mathbb{R}^{2\ell}
\]

It is easy to verify that

\[
\text{Decomp}_g((a, b)) \cdot G = (a, b)
\]

**Definition 5 (External Product).** Let \( C \in \text{RGSW}^{N}_{s,t}(m_1) \), and \( b \in \text{RLWE}^{N}_{s,t}(m_2) \), then the external product on them is

\[
C \boxtimes b = \text{Decomp}_g(b) \cdot C
\]

**Lemma 1.** Let \( C \in \text{RGSW}^{N}_{s,t}(m_1) \) with error variance \( \sigma_1^2 \), \( b \in \text{RLWE}^{N}_{s,t}(m_2) \) with error variance \( \sigma_2^2 \). Let \( ct = C \boxtimes b \), then \( ct \in \text{RLWE}^{N}_{s,t}(m_1 \cdot m_2) \) with error variance \( \sigma_{\text{ext}}^2 \leq 2\ell Ng^2\sigma_1^2 + \|m_1\|^2\sigma_2^2 \). Where \( g \) is defined as follows: If base power gadget is used, \( g \) is the decomposition base. If RNS gadget is used then \( g = \max\{g_0, ..., g_{\ell-1}\} \).

**Proof.** Let \( b = (a, b) \), and \( e \) be the error term of \( b \). Let \( \{e_i\}_{i=0}^{2\ell-1} \) be the error terms of RLWE samples in \( C \). Let \( (a_0, ..., a_{\ell-1}, b_0, ..., b_{\ell-1}) = \text{Decomp}_g((a, b)) \). We have \( C \boxtimes b = ct + m_1 \cdot (a, b) \), where \( ct \in \text{RLWE}^{N}_{s,t}(0) \). Since \( m_2 \) is the message of \((a, b)\), then \( C \boxtimes b \in \text{RLWE}^{N}_{s,t}(m_1 \cdot m_2) \).

The error term of \( C \boxtimes b \) is of the form \( \sum_{i=0}^{\ell-1}(a_i \cdot e_i + b_i \cdot e_{i+\ell}) + m_1 \cdot e \). Simple analysis shows that the variance \( \sigma_{\text{ext}}^2 \) of the error term is bounded by \( 2\ell Ng^2\sigma_1^2 + \|m_1\|^2\sigma_2^2 \). \( \square \)
B Noise Analysis

Lemma 2. Let $\mathbf{ct} \in LWE_{q}^{N}(m)$ with error variance $\sigma^{2}$, and $\mathbf{ct}' = \text{Mod}_{q} \rightarrow q' (\mathbf{ct})$, then $\mathbf{ct}' \in LWE_{q}^{N}(\lfloor (q'/q) \cdot m \rfloor)$ with error variance $\sigma_{\text{max}}^{2} = (q'/q)^{2}\sigma^{2} + N/24 + 1/6$.

Proof. Let $(a, b) = LWE_{q}^{N}(m)$ under key $s$, where $(a, s) + b = m + e \in \mathbb{Z}_{q}$. Let $(a', b') = \mathbf{ct}' = \text{Mod}_{q} \rightarrow q' (\mathbf{ct})$, then

$$
\langle (q'/q) \cdot a, s \rangle + \langle (q'/q) \cdot b \rangle = \langle (q'/q) \cdot a, s \rangle + (q'/q) \cdot b + \langle r, s \rangle + \epsilon_{1} \in \mathbb{Z}
$$

$$
= \langle (q'/q) \cdot (m + e) + (r, s) + \epsilon_{1} = \langle (q'/q) \cdot m \rangle + \langle q'/q \cdot e + (r, s) \rangle + \epsilon_{1} + \epsilon_{2} = ((q'/q) \cdot m') + e'
$$

where $r := (q'/q)a - \langle (q'/q) \cdot a \rangle$ and $\|r\|_{\infty} < 1/2, |\epsilon_{1}| < 1/2, |\epsilon_{2}| < 1/2$.

Similar to [20], the variance of $(r, s)$ is $\|s\|_{2}^{2}/12$ according to the central limit heuristic and the randomness of $a$. Then by the distribution of $s$ we have the variance of $e'$ being $(q'/q)^{2}\sigma^{2} + (N/2)/12 + 1/12 + 1/12 = (q'/q)^{2}\sigma^{2} + N/24 + 1/6$, where $\sigma^{2}$ is the variance of $e$. \hfill $\square$

Lemma 3. Let $\mathbf{ct} \in RLWE_{s,q}^{N}(m)$ with error variance $\sigma^{2}$, $\text{KSK}$ be a RGSW ciphertext with error variance $\sigma_{\text{kSK}}^{2}$. Then $\mathbf{ct}' = \text{Key}_{S_{s \rightarrow s'}}(\mathbf{ct}, \text{KSK})$ is a ciphertext in $RLWE_{s,q}^{N}(m)$ with error variance $\sigma'^{2}$, and

$$
\sigma'^{2} \leq \ell N g^{2} \sigma_{\text{kSK}}^{2} + \sigma^{2},
$$

where $g$ is defined as follows: If base power gadget is used in BK, $g$ is the decomposition base. If RNS gadget is used then $g = \max\{g_{0}, ..., g_{t-1}\}$.

Proof. The key switching procedure is essentially an external product except only $\ell$ RLWE ciphertexts are needed. For correctness, $(\mathbf{a}', \mathbf{b'})$ in Algorithm 4 is of the form $\mathbf{ct} + (a, b)$, where $\mathbf{ct} \in RLWE_{s,q}^{N}(0)$. Thus $\mathbf{ct} \in RLWE_{s,q}^{N}(m)$. The error term of $\mathbf{ct}'$ is $\sum_{i=0}^{t} a_{i} \cdot e_{i}$, where $(a_{0}, ..., a_{t-1})$ is the decomposition of $a$, and $(e_{0}, ..., e_{t-1})$ are error term of the first RLWE ciphertext in $\text{KSK}$. Similar to Lemma 1, the error variance of $\mathbf{ct}$ satisfies $\sigma'^{2} \leq \ell N g^{2} \sigma_{\text{kSK}}^{2} + \sigma^{2}$. This completes the proof. \hfill $\square$

Lemma 4. Let $\mathbf{ct} \in LWE_{s,q}^{N}(m)$ with error variance $\sigma^{2}$, $\text{KSK}_{s \rightarrow s'}$ be a set of RGSW ciphertexts with error variance $\sigma_{\text{kSK}}^{2}$. Then $\mathbf{ct}' = \text{Dim}_{S_{N \rightarrow n'}}(\mathbf{ct}, \text{KSK}_{s \rightarrow s'})$ is a ciphertext in $LWE_{s,q}^{N'}(m)$ with error variance $\sigma'^{2}$, and

$$
\sigma'^{2} \leq \ell N g^{2} \sigma_{\text{kSK}}^{2} + \sigma^{2},
$$

where $g$ is defined as follows: If base power gadget is used in BK, $g$ is the decomposition base. If RNS gadget is used then $g = \max\{g_{0}, ..., g_{t-1}\}$.

Proof. The correctness directly follows from the fact that the constant term of $\sum_{i=0}^{N/n-1} a_{i} \cdot s_{i}$ is the same as $(a, s) \in \mathbb{Z}_{q}$. The analysis of the error variance is similar to Lemma 3. It involves in $N/n$ key switching procedures in the ring $\mathbb{Z}_{q}[X]/(X^{n} + 1)$, and the extract procedure will not change the error variance. Therefore $\sigma'^{2} \leq (N/n)(\ell n g^{2} \sigma_{\text{kSK}}^{2}) + \sigma^{2} = (N g^{2} \sigma_{\text{kSK}}^{2}) + \sigma^{2}$. \hfill $\square$

Lemma 5. Let $\mathbf{ct} \in LWE_{s,q}^{N}(m)$ with error variance $\sigma^{2}$, $\text{AK}$ be a set of RGSW ciphertexts with error variance $\sigma_{\text{ak}}^{2}$. Then $\mathbf{ct}' = \text{LWE-to-RLWE}(\mathbf{ct}, \text{AK})$ is a ciphertext of $RLWE_{s,q}^{N}(m)$ with error variance $\sigma'^{2}$, and

$$
\sigma'^{2} \leq \ell N g^{2} \sigma_{\text{ak}}^{2} + \sigma^{2},
$$

where $g$ is defined as follows: If base power gadget is used in AK, $g$ is the decomposition base. If RNS gadget is used then $g = \max\{g_{0}, ..., g_{t-1}\}$.

27
Proof. In Algorithm 8, denote \([a, s] + b|q = m + e =: \tilde{m}\), then \(\mathsf{ct}^*\) can be seen as a noiseless ciphertext on \(m^* \in \mathcal{R}_q\) whose constant term is \(\tilde{N}\tilde{m}\). The process of evaluating trace invokes \(\log N\) times \(\mathsf{HomAuto}\) procedures, and each involves a key switching procedure. This brings a new noise of variance \(\sigma_{\text{ext}}^2 \leq \ell N g^2 \sigma_{\text{sk}}^2\). Let the error variance of the output ciphertext in step 3 is \(\sigma_j^2\) in the \(j\)-th iteration, for \(1 \leq j \leq \log N\). We have \(\sigma_{j+1}^2 = \sigma_j^2 + (\sigma_j^2 + \sigma_{\text{ext}}^2)\) and \(\sigma_1^2 = 0\). Thus \(\sigma_{\log N}^2 = \sum_{i=0}^{\log N - 1} 2^i \cdot \sigma_{\text{ext}}^2 \leq N \sigma_{\text{ext}}^2\).

The resulting message of trace is \([N \cdot \tilde{N}\tilde{m}]_q = \tilde{m} = m + e\). If we view \(m\) as the plaintext, then the noise of Algorithm 8 is of variance \(\sigma'^2 \leq \ell N^2 g^2 \sigma_{\text{sk}}^2 + \sigma^2\).

□