Functional Bootstrapping for FV-style Cryptosystems

Dongwon Lee, Seonhong Min, and Yongsoo Song

Seoul National University, Seoul, Republic of Korea
{dongwonlee95,minsh,y.song}@snu.ac.kr

Abstract. Fully Homomorphic encryption (FHE) enables the computation of an arbitrary function over encrypted data without decrypting them. In particular, bootstrapping is a core building block of FHE which reduces the noise of a ciphertext thereby recovering the computational capability. This paper introduces a new bootstrapping framework for the Fan-Vercuteren (FV) scheme, called the *functional bootstrapping*, providing more generic and advanced functionality than the ordinary bootstrapping method. More specifically, the functional bootstrapping allows us to evaluate an arbitrary function while removing the error of an input ciphertext. Therefore, we achieve better depth consumption and computational complexity as the evaluation of a circuit can be integrated as part of the functional bootstrapping procedure. In particular, our approach extends the functionality of FV since it is even applicable to functions between different plaintext spaces.
At the heart of our functional bootstrapping framework is a novel homomorphic Look-Up Table (LUT) evaluation method where we represent any LUT using only the operations supported by the FV scheme. Finally, we provide a proof-of-concept implementation and present benchmarks. In concrete examples, such as delta and sign functions, our functional bootstrapping takes about 46.5s or 171.4s for 9-bit or 13-bit plaintext modulus, respectively.

1 Introduction

Fully Homomorphic Encryption (FHE) facilitates the evaluation of arbitrary functions on encrypted data without the need for decryption. It prevents private information from being revealed while evaluating data within an untrusted environment. The pioneering work on FHE was introduced by Gentry [19], and subsequent research has focused on leveraging the Learning With Errors (LWE) problem [42] or its ring variant, the Ring LWE (RLWE) problem [36]. Notable advancements in FHE construction such as FV [3, 17], GSW [24], BGV [4], TFHE [13], and CKKS [10], have emerged based on these foundational problems.

In these encryption schemes, a ciphertext accumulates a certain amount of noise and the noise increases while performing homomorphic operations. Since excessive noise can lead to incorrect decryption results, managing the noise is critical. While there are several well-known techniques such as the special modulus technique [23] to mitigate this issue, there exists no known solution to perfectly remove the noise growth from homomorphic operations. Consequently, to construct the FHE scheme, a procedure to lower the ciphertext noise is required after a substantial number of operations. Addressing this challenge, the *bootstrapping* technique was introduced by Gentry [20] to effectively lower the ciphertext noise and ensure the integrity of decryption results in the face of noise accumulation in homomorphic evaluation.

The bootstrapping technique involves homomorphically evaluating the decryption circuit to refresh the ciphertext, a process known for its computational expense due to the lack of straightforward support for the decryption circuit operations within FHE schemes. To achieve efficient bootstrapping, it becomes necessary to either implement optimizations for enhanced performance or reconfigure the decryption circuit for simplification. Notably, numerous studies on CKKS bootstrapping have focused on optimizing the approximation of the modulus operation within the decryption circuit [5, 32, 33]. For FV schemes, specific approaches have been undertaken to enhance bootstrapping efficiency. For instance, in the context of FV schemes, works such as [27, 6, 18] have proposed circuit designs aimed at eliminating certain least significant digits to reduce noise and thereby improving the overall performance of the circuit. This paper concentrates on FV bootstrapping, aiming to extend the functionality of preceding bootstrapping techniques.

Significant advancements in FV bootstrapping have been observed, with existing works predominantly emphasizing the reduction of ciphertext noise while preserving the integrity of the message value. Therefore, when evaluating a high-degree circuit, the primary role of the bootstrapping technique is to facilitate further computations by diminishing the error. Consequently, bootstrapping is not directly involved in altering the calculation of the circuit but serves as a means to enable additional operations by minimizing the error.

On the other hand, prior studies did not take into account the plaintext modulus and have proceeded with bootstrapping while maintaining this modulus. As a result, all data utilized in the evaluation must be encrypted using the same plaintext modulus. However, depending on the property of the data such as data, range, or the number of data, each data may have a suitable parameter for optimal performance. A proper parameter set tailored to the specific properties of the data can contribute to performance improvements and allow for more flexible packing within the ciphertext [4]. This adaptability may extend the usability of the FV scheme, as not all data necessarily needs to be encrypted using the same parameter set.

1.1 Our Contribution

In this work, we introduce a noble bootstrapping method for FV cryptosystem, called the *functional bootstrapping*. This conceptually new bootstrapping technique has two main advantages over the existing FV bootstrapping algorithms. Firstly, our new bootstrapping method enables us to use different modulus for the input and output ciphertext. Secondly, an arbitrary univariate function can be evaluated during our functional bootstrapping without any additional depth consumption, in contrast to the conventional bootstrapping outputting the identical message to the input ciphertext. As a result, we can achieve a better performance both in the bootstrapping and the circuit evaluation in the FV scheme. For example, the complexity of FV bootstrapping is heavily dependent on the size of the prime factor plaintext modulus of which the bootstrapping is performed, and therefore bootstrapping of a ciphertext with a large prime modulus was almost infeasible. However, with our innovative bootstrapping method, a smaller plaintext modulus can be utilized for the bootstrapping procedure and consequently, the complexity of the bootstrapping can be mitigated. Moreover, recall that any univariate function can be evaluated during our functional bootstrapping without any additional cost, the complexity of a large-depth circuit evaluation can be reduced almost for free by integrating the circuit into the bootstrapping itself.

We achieve these aforementioned functionalities from an evaluation algorithm for a general Look-Up Table (LUT) over the commutative ring $\mathbb{Z}_{p^r}$ for a prime p. In the existing bootstrapping method of a ciphertext with initial plaintext modulus p, the ciphertext modulus is firstly reduced into p^r for some $r > 0$ while guaranteeing the correct decryption. Then $r-1$ erroneous least significant bits (LSB) are iteratively eliminated. In contrast, our functional bootstrapping process is performed over a plaintext modulus q^r instead of p^r. This alteration makes the bootstrapping challenging since the error part cannot be simply removed by removing the LSB. To resolve this problem, we develop an algorithm which can compute an arbitrary LUT from $\mathbb{Z}_{p^r}$ to $\mathbb{Z}_p$.

We observe that there are a series of polynomial $\{u_j^i\}_{1<i\leq r, 0\leq j<q}$, which 'selectively' removes the LSB. Utilizing these polynomials, we can evaluate any LUTs with an iterative manner with respect to the LSBs of the input message. First, we provide an optimized LUT evaluation method specified for a simple LUT which has a form of step function. This simple LUT evaluation requires $\approx \frac{16}{3}\sqrt{r^3 p}$ key-switching operations and consumes $r \log p + \log r!$ depth asymptotically. This result is then generalized to an arbitrary LUT with a simple linear transformation. Consequently, it is capable of handling widely used functions that are not readily supported by FHE operations. In this work, we focus on the specific functions which are delta and sign functions. The adaptation of our algorithm to such functions allows an efficient computation in scenarios where the underlying functions present challenges within the FHE framework.

Finally, we implement our noble method for handling the delta and sign functions utilizing an open-source FHE library, Lattigo [37]. We also provide a benchmark analysis for both functions to assess the performance of our approach. In addition, we outline various applications where functional bootstrapping can be effectively employed. These applications demonstrate the practical utility of our approach and highlight its potential impact in real-world use cases involving the evaluation of arbitrary functions within the FHE framework.

1.2 Related Works

Liu et al. [35] recently improved the amortized bootstrapping of the TFHE scheme via the FV bootstrapping. Their method includes a very limited form of functional bootstrapping. In a nutshell, they utilized a 'fake' plaintext modulus of large size to encode a binary message in finite field $\mathbb{Z}_3$ to evaluate arbitrary binary gates with an addition and a bootstrapping [16]. Concurrently, Okada et al. suggested a TFHE-style functional bootstrapping in the FV scheme in [39] in a similar setting of leveraging a fake plaintext modulus. Their main idea is to perform a blind rotation over the exponent of the roots of unity, instead of the monomial. However, the functionality of these works is limited, since they only support a small modulus, or suffer from a high computational cost.

The homomorphic evaluation of the sign/delta function using the FV scheme is well studied in various previous researches, to realize comparison operation or SQL query homomorphically. Cheon et al. [11, 12] utilized a bivariate polynomial interpolation in order to compare two (large) integers. This approach is further refined by Tan et al. [43] by leveraging the finite field structure in order to compare the input integers digit-wise. On the other hand, a univariate polynomial based on the interpolation approach is also proposed in [38]. Later, Kim et al. [31] improved this method by encoding a large integer into a finite field $GF(p^d)$ and leveraging the Frobenius automorphism in order to evaluate a polynomial, thus not consuming modulus from homomorphic multiplications. Iliashenko and Zucca [29] achieved a significant speedup of both these two approaches, by observing that we can make the coefficient vector of the interpolation polynomials sparse.

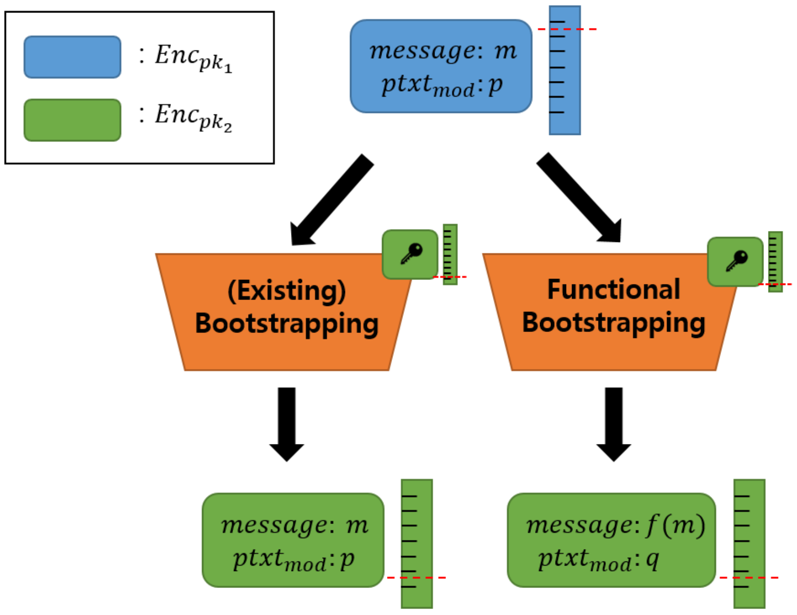

Fig. 1: Concept of functional bootstrapping.

2 Background

2.1 Notation

We denote the ring of integers of the $2N$-th cyclotomic field for some power of two N by $R = \mathbb{Z}[X]/\Phi_{2N}(X) = \mathbb{Z}[X]/(X^N + 1)$, and the residue ring of R modulo an integer $Q > 0$ by $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$. We use $\mathbb{Z} \cap (-Q/2, Q/2]$ as a representative of $\mathbb{Z}_Q$, and denote by $[a]_Q$ the reduction of a modulo Q. Throughout the paper, we write $x \leftarrow D$ to represent that x is sampled from the distribution D. We denote the uniform distribution over a finite set S by $\mathcal{U}(S)$. For $\sigma > 0$, D_σ denotes a distribution over R sampling N coefficients independently from the discrete Gaussian distribution of variance σ^2 and χ as a key distribution. We also use $\overline{a_{r-1}a_{r-2}\ldots a_0}$ to denote the base-p representation of $a \in \mathbb{Z}_{p^r}$ where $a_i \in \mathbb{Z}_p$, *i.e.*, $a = \sum_{i=0}^{r-1} a_i p^i$.

2.2 The FV Scheme and Bootstrapping

The FV scheme supports addition and multiplication over integers. A plaintext space is $R_p = \mathbb{Z}_p[X]/(X^N + 1)$ where p is a modulus of the plaintext space. A ciphertext consists of two polynomials from a ring $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$.

We make use of *gadget decompositon*, which is a commonly used technique in lattice-based HE cryptosystems for noise reduction [1, 9, 26]. For a modulus Q, real $B > 0$, and any $a \in R_Q$, the gadget decomposition $h : R_Q \to R^t$ and the gadget vector $\mathbf{g} = (g_0, g_1, \ldots, g_{t-1}) \in R_Q^t$ satisfy following equation where $\mathbf{b} = (b_0, b_1, \ldots, b_{t-1}) \leftarrow h(a)$:

$$\sum_{0 \le i < t} b_i \cdot g_i = a \pmod{Q} \qquad \text{and} \qquad \|\mathbf{b}\|_\infty \le B.$$

A detailed description of FV scheme is given in the below.

• $\underline{\texttt{FV.Setup}(1^\lambda)}$: Set the ring degree N, the plaintext modulus p, the ciphertext modulus Q, the key distribution χ over R, and the error parameter σ. Choose a gadget decomposition $h : R_Q \to R^\ell$ with a gadget vector $\mathbf{g} \in R_Q^\ell$. Output the parameter set $\mathsf{pp} = (m, p, Q, \chi, \sigma, h, \mathbf{g})$.

• $\underline{\texttt{FV.KeyGen}}$: Sample $s \leftarrow \chi$, $a \leftarrow \mathcal{U}(R_Q)$ and $e \leftarrow D_\sigma$. Set the secret and public keys as $\mathsf{sk} = s$ and $\mathsf{pk} = (b, a) \in R_Q^2$ where $b = -s \cdot a + e \pmod{Q}$. Sample $\mathbf{k}_1 \leftarrow \mathcal{U}(R_Q^\ell)$ and $\mathbf{e} \leftarrow D_\sigma^\ell$, and set the relinearization key as $\mathsf{rlk} = (\mathbf{k}_0, \mathbf{k}_1) \in R_Q^{\ell \times 2}$ where $\mathbf{k}_0 = -s \cdot \mathbf{k}_1 + \mathbf{e} + s^2 \cdot \mathbf{g} \pmod{Q}$.

• $\underline{\texttt{FV.Encode}(\mathbf{m})}$: Let k be the biggest power of two such that $2k | p - 1$. Then, given a message vector $\mathbf{m} \in \mathbb{Z}_p^k$, return a plaintext $\mu = \sigma^{-1}(\mathbf{m})$ where $\sigma : \mu \mapsto (\mu \mod X^{N/k} - \zeta^{2i+1})_{0 \le i < k})$ for $2k$-th root of unity ζ modulo p.

• $\underline{\texttt{FV.Decode}(\mu)}$: Given $\mu \in R_p$, return $\mathbf{m} = \sigma(\mu)$.

• $\underline{\texttt{FV.Enc}(\mathsf{pk}; \mu)}$: Sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. Given an encoding $\mu \in R_p$, output the ciphertext $\mathsf{ct} = w \cdot \mathsf{pk} + (\Delta \cdot \mu + e_0, e_1) \pmod{Q}$, for $\Delta = \lfloor Q/p \rceil$.

• $\underline{\texttt{FV.Dec}(\mathsf{sk}; \mathsf{ct})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_Q^2$ and associated secret key $\{\mathsf{sk}\}$, return $\mu = \lfloor (p/Q) \cdot (c_0 + c_1 \cdot s) \rceil \pmod{p}$.

• $\underline{\texttt{FV.Add}(\mathsf{ct}, \mathsf{ct}')}$: Given two ciphertexts $\mathsf{ct}, \mathsf{ct}' \in R_Q^2$, output $\mathsf{ct}_{add} = \mathsf{ct} + \mathsf{ct}' \pmod{Q}$.

• $\underline{\texttt{FV.Mult}(\mathsf{rlk}; \mathsf{ct}, \mathsf{ct}')}$: Given two ciphertexts $\mathsf{ct} = (c_0, c_1)$, $\mathsf{ct}' = (c_0', c_1') \in R_Q^2$ and the relinearization key rlk, let $(d_0, d_1, d_2) = \lfloor (p/Q) \cdot \mathsf{ct} \otimes \mathsf{ct}' \rceil$ such that $d_0 = \lfloor (p/Q) \cdot c_0 c_0' \rceil \pmod{Q}$, $d_1 = \lfloor (p/Q) \cdot (c_0 c_1' + c_0' c_1) \rceil \pmod{Q}$, and $d_2 = \lfloor (p/Q) \cdot c_1 c_1' \rceil \pmod{Q}$. Output the ciphertext $\mathsf{ct}_{mul} = (d_0, d_1) + (\langle h(d_2), \mathbf{k}_0 \rangle, \langle h(d_2), \mathbf{k}_1 \rangle) \pmod{Q}$.

The bootstrapping for the FV scheme is conducted in a similar manner to the bootstrapping of the BGV scheme [27, 6]. It consists of four steps which are modulus switching, homomorphic inverse

Discrete Fourier Transfromation (iDFT), polynomial evaluation (digit extraction), and homomorphic Discrete Fourier Transfromation (DFT). These procedures are essentially homomorphic evaluations of the decryption circuit.

Firstly, the input ciphertext is converted in order to make the decryption circuit as compact as possible. Then, iDFT is performed homomorphically to move the coefficients to the slots but with erroneous lower bits, which are removed during the polynomial evaluation, or so-called digit extraction procedure. Finally, we perform DFT homomorphically on the input ciphertext to move the coefficients back to the slots.

Among the above steps, the polynomial evaluation for digit extraction is the main bottleneck of FV/BGV bootstrapping. This procedure involves removing a certain number of least significant digits in base p where the plaintext modulus is the power of prime p. Let $w \in \mathbb{Z}_{p^r}$ be an input of the digit extraction algorithm which can be represented as follows:

$$w = \sum_{i=0}^{r} w_i p^i \quad (w_i \in \mathbb{Z}_p)$$

To achieve the functionality of digit extraction, previous methodologies calculate $w_{i,j}$ of where the least significant digit is w_i, and the next j least significant digits are all zeros. The goal is to remove $v < r$ least significant digits by subtracting $w_{0,r-1}, w_{1,r-2}, \ldots, w_{v-1,r-v}$ from the input w. In the work presented in [27], a lifting polynomial was a crucial building block. This polynomial outputs $w_{i,j+1}$ for an input $w_{i,j}$ and iterates lifting polynomial j times to obtain $w_{i,j}$.

Subsequent optimizations have been applied to the bootstrapping of the FV scheme, as outlined in further works such as [6, 18]. These optimizations primarily focus on modifying the digit extraction algorithm for improved efficiency. In contrast to the method presented in [27], which applies the lifting polynomial iteratively j times to evaluate $w_{i,j}$, the approach in [6] introduces efficient polynomials known as digit extraction polynomials, or lowest digit removal polynomials. It possesses a lower degree than the lifting polynomial used in previous methodologies, leading to enhanced bootstrapping performance. Moreover, with the digit extraction polynomial, there is no need to compute $w_{i,k}$ for all $0 \leq k \leq r-i-1$ to evaluate $w_{i,r-i-1}$. This optimization reduces the computational load by computing only the necessary $w_{i,j}$ terms, thereby improving the overall efficiency of the digit extraction process during bootstrapping.

In the work presented in [18], additional optimizations are introduced to enhance the digit removal procedure during bootstrapping. These optimizations leverage the properties of polyfunctions, resulting in a more efficient process. For example, the complexity of the digit extraction polynomial process is reduced by incorporating null polynomials. The authors also exploit the properties of even or odd functions depending on the value of the prime modulus p. Specifically, when $p = 2$, the digit extraction polynomials are designed to take advantage of the even function property. Similarly, when p is an odd prime, the polynomials capitalize on the odd function property. These tailored approaches enhance the evaluation of digit extraction polynomials, taking into account the specific characteristics of the polyfunctions.

In [6], the authors also introduce an optimized version of the bootstrapping for FV/BGV scheme named the *slim mode*. The slim mode is designed to reduce the number of digit extraction by applying batching isomorphism. Unlike the original bootstrapping, it performs homomorphic iDFT before modulus switching to batch the messages into a single polynomial. Consequently, it makes the bootstrapping about d times faster where d is the multiplicative order of p in $\mathbb{Z}_m^\times$. The slim mode bootstrapping procedure is provided in Fig. 2 and we only consider the slim mode bootstrapping in our work. More details for each phase of the slim bootstrapping are following:

• **Homomorphic DFT** Given an FV ciphertext ct which encrypts a message vector in $\mathbb{Z}_p^k$, return the product between DFT matrix (over $\mathbb{Z}_p$) and ct.

• **Modulus switching** Using the scale-invariant property of FV ciphertext (c_0, c_1), we change the ciphertext modulus to p^r to obtain an FV encryption $(c_0', c_1') \in R_{p^r}^2$. In FV, we can simply switch the modulus by scaling and rounding for each component of the ciphertext, *i.e.,* $c_i' = \lfloor p^r/Q \cdot c_i \rceil$. Then, we output $(\lfloor Q/p^r \rceil \cdot c_0', \lfloor Q/p^r \rceil \cdot c_1') \in R_Q^2$ for some large ciphertext modulus Q. Note that this resulting ciphertext is essentially an encryption of $M(X) \cdot p^{r-1} + e(X) \in R_{p^r}$ for some error polynomial e with $\|e\|_\infty < \frac{p^{r-1}}{2}$.

$$\begin{tikzcd}[row sep=large]
\mathsf{Enc}(m_0, \dots, m_{k-1}) \arrow[d, "\text{Homomorphic DFT}"] \\
\mathsf{Enc}\big(M(X) := m_0 + m_1X^d \cdots + m_{k-1}X^{(k-1)d}\big) \arrow[d, "\text{Modulus switching}"] \\
\mathsf{Enc}\big(M(X) \cdot p^{r-v} + e(X)\big) \arrow[d, "\text{Homomorphic iDFT}"] \\
\mathsf{Enc}\big(m_0 \cdot p^{r-v} + e_0, \dots, m_{k-1} \cdot p^{r-v} + e_{k-1}\big) \arrow[d, "\text{Digit Extraction}"] \\
\mathsf{Enc}(m_0, \dots, m_{k-1})
\end{tikzcd}$$

Fig. 2: Pipeline for the FV slim bootstrapping

• **Homomorphic iDFT** Given an FV ciphertext ct encrypting the messages in $\mathbb{Z}_{p^r}$, return the product between iDFT matrix (over $\mathbb{Z}_{p^r}$) and ct.

• **Digit Extraction** Finally, we homomorphically extract the highest digit of the base p representation of the message. To realize this, we leverage the lifting polynomials $\{F_i\}$ and the digit extraction polynomials $\{G_i\}$. A i-th lifting polynomial F_i is a polynomial defined over $\mathbb{Z}_{p_i}$ such that $F_i(x \cdot p^j + y) = y \pmod{p^{j+1}}$ for any $0 < j < i$, $0 \le x < p^{i-j}$ and $0 \le y < p$. A i-th digit extraction polynomial G_i is a polynomial defined over $\mathbb{Z}_{p_i}$ such that $G_i(x \cdot p + y) = y \pmod{p^i}$ for any $0 \le x < p^{i-1}$ and $0 < y < p$. We note that G_i is essentially equivalent to an iterative composition of F_i for i times. Now we can erase the lower bits iteratively by evaluating these polynomials and homomorphically dividing the plaintext by p. We will introduce the details in the later sections.

3 Functional Bootstrapping for FV

Bootstrapping is a technique employed to reduce the noise of a ciphertext, allowing a user to perform unlimited number of computations without compromising the privacy of the underlying message. In order to perform a large depth circuit, bootstrapping is almost unavoidable. However, in the existing works, bootstrapping is typically utilized as a black-box technique to reduce the noise bound, not directly influencing the performance of the circuit evaluation. In other words, the bootstrapping and the circuit evaluation were independent in the existing framework. We stress that there exists an opportunity for improvement of the circuit evaluation by strategically integrate the bootstrapping into the circuit itself.

In this section, we introduce a noble bootstrapping framework for the FV cryptosystem, called the *functional bootstrapping*, which provides a general functionality from the previous bootstrapping. To be precise, we integrate FV bootstrapping and function evaluation into a single procedure without any additional cost. From doing so, the overall circuit depth and complexity can be mitigated, since we can save time and depths from further evaluations after the bootstrapping.

Our new bootstrapping framework generalizes the functionality of the bootstrapping in two directions. First, it has a capability to evaluate an arbitrary function while reducing the noise of ciphertexts, unlike the conventional bootstrapping methods. Second, the class of the functions we can evaluate is now even broader. For example, our new method also supports an evaluation of the function between different domains (*i.e.,* message spaces with distinct plaintext modulus). This innovative approach holds the potential to enhance the overall performance of evaluations conducted on encrypted data, presenting a paradigm shift from traditional bootstrapping methodologies.

In the following subsections, we present an entire pipeline of functional bootstrapping and how to evaluate an arbitrary function possible between different spaces. Thereby the output plaintext modulus does not have to be identical to the input plaintext modulus. Furthermore, we also provide an overview of LUT evaluation corresponding to the arbitrary function.

3.1 Pipeline of the Functional Bootstrapping

Step	Encryption		Modulus	
	Message Vector	Plaintext	Ciphertext	Plaintext
	$\vec{\boldsymbol{m}} = (m_i)_{0\le i<k} \in \mathbb{Z}_p^k$		Q_{in}	p
Homomorphic DFT		$M(X) = \sum_{0\le i<k} m_i X^{d\cdot i} \in R_p$	Q_{in}	p
Modulus Switching		$\lfloor q^r/p \rceil \cdot M(X) + e(X) \in R_{q^r}$	Q	q^r
Homomorphic iDFT	$(\lfloor q^r/p \rceil \cdot m_i + e_i)_{0\le i<k} \in \mathbb{Z}_{q^r}^k$		Q	q^r
LUT Evaluation	$(f(m_i))_{0\le i<k} \in \mathbb{Z}_q^k$		Q	q

Fig. 3: Pipeline of the functional bootstrapping.

Let p and q be the input and output plaintext modulus, and Q_{in} and Q be the input and output ciphertext modulus, respectively. We also denote N, k, and d as the ring dimension, the number of the slots, and the multiplicative order of p, respectively, (*i.e.,* $k \cdot d = N$). Then, the goal of the functional bootstrapping is to evaluate an arbitrary function $f : \mathbb{Z}_p \mapsto \mathbb{Z}_q$ while bootstrapping an encryption of a message vector $\vec{m} = (m_i)_{0\le i<k} \in \mathbb{Z}_p^k$. Note that we aim to evaluate the function on the message, not on the coefficients, we choose to use the pipeline of the slim mode bootstrapping [6] because its digit extraction is performed on the messages rather than the coefficients as in the conventional bootstrapping algorithm. The overall pipeline of the functional bootstrapping is given in Figure 3, which consists of the following four steps. Here, r is a constant which will be determined in the later sections.

• **Homomorphic DFT** Given FV encryption of a message vector $(m_i)_{0\le i<k}$ where $m_i \in \mathbb{Z}_p$ $(0 \le i < k)$, we first perform DFT homomorphically to obtain encryption of the plaintext $M(X) := m_0 + m_1 X^d \cdots + m_{k-1}X^{(k-1)d}$.

• **Modulus switching** For the input ciphertext $\mathsf{ct} = (c_0, c_1) \in R_{Q_{in}}^2$ which is an encryption of the plaintext $M(X)$, convert the ciphertext modulus to q^r and obtain a new ciphertext $\mathsf{ct}' = (c_0', c_1') \in R_{q^r}^2$. Output $\mathsf{ct}^* = (c_0^*, c_1^*) \in R_Q^2$, an encryption of a plaintext $\lfloor q^r/p \rceil \cdot M(X) + e(X)$, where the plaintext modulus and ciphertext modulus are q^r and Q, respectively.

• **Homomorphic iDFT** We perform iDFT homomorphically and obtain an FV encryption of the message vector $(\lfloor q^r/p \rceil \cdot m_i + e_i)_{0\le i<k}$ of plaintext modulus q^r. Here, e_i is $(i\cdot d)$-th coefficient of the error polynomial $e(X)$ for $0 \le i < k$.

• **LUT Evaluation** For the input ciphertext $\mathsf{ct} \in R_{q^r}^2$, return the ciphertext $\mathsf{ct}' \in R_q^2$ which encrypting the operation result of the function f, *i.e.,* an encryption of the message vector $(f(m_i))_{0\le i<k}$.

Compared to the existing bootstrapping (slim) method summarized in Figure 2, homomoprhic DFT and iDFT proceed in a similar way as existing version and some distinctions arise in the following two steps: (1) modulus switching, and (2) LUT evaluation. Our new method extends the functionality compared to the previous bootstrapping approach from these phases.

During the modulus switching phase, we convert the ciphertext modulus into q^r and generate an erroness ciphertext ct^* with a large ciphertext modulus Q. This ciphertext is an encryption of $\lfloor q^r/p \rceil \cdot m + e$ which belongs to the new message space $\mathbb{Z}_{q^r}$. It ensures that the ciphertext is now compatible with the new modulus q^r. During this modulus switching, a single input message corresponds to multiple values in the converted message space. When there is an operation (proceeded in LUT evaluation) that can be carried out to have the same value for these multiple values, the output ciphertext is an encryption with converted message space which has a different plaintext modulus. For example, if there is an operation to have the identical value as an input for several values corresponding to the input, our bootstrapping is the same as the existing bootstrapping and the only difference is that the plaintext modulus has changed.

On the other hand, to evaluate an arbitrary function, multiple values corresponds to a single message should have resulted in the same value after the functional bootstrapping. Therefore, we need to evaluate LUT from $\mathbb{Z}_{q^r}$ to $\mathbb{Z}_q$ that satisfies the consistency of the output for consecutive values associated with a single input. More detailed descriptions are provided in the following subsections.

3.2 Modulus Switching

During the functional bootstrapping procedure, we scale the input ciphertext as long as the decryption is valid in order to make the decryption circuit as compact as possible. Let $\mathsf{ct} = (c_0, c_1) \in R^2_{Q_{in}}$ is an encryption of a message vector $\vec{m} \in \mathbb{Z}_p^k$. Then, since FV is a scale-invariant HE scheme, $\mathsf{ct}' = (\lfloor q^r/Q_{in} \cdot c_0 \rceil, \lfloor q^r/Q_{in} \cdot c_1 \rceil) \in R^2_{q^r}$ is a valid encryption of $\vec{m}$ as long as q^r is sufficiently large to accommodate the rounding error, regardless the initial ciphertext modulus Q_{in}.

Now, recall that we aim to remove the erroneous lower bits of the plaintext homomorphically in the remaining steps of the bootstrapping, this new ciphertext ct' is needed to be homomorphically decrypted under the plaintext modulus q^r. This can be achieved simply multiplying a constant $\lfloor Q/q^r \rceil$ where $Q >> p, q^r$ is the output ciphertext modulus. In other words, let $\mathsf{ct}^* := \lfloor Q/q^r \rceil \cdot \mathsf{ct}'$ is an encryption of the plaintext $c'_0 + c'_1 \cdot s \pmod{q}^r$ where $\mathsf{ct}' = (c'_0, c'_1)$. We describe the exact algorithm in Alg. 1.

Algorithm 1 Modulus Switching

```
Input: A ciphertext $\mathsf{ct} = (c_0, c_1) \in R^2_{Q_{in}}$
Output: A ciphertext $\mathsf{ct}^* = (c^*_0, c^*_1) \in R^2_Q$
1: $(c'_0, c'_1) \leftarrow (\lfloor q^r/Q_{in} \cdot c_0 \rceil, \lfloor q^r/Q_{in} \cdot c_1 \rceil) \in R^2_{q^r}$
2: $(c^*_0, c^*_1) \leftarrow (\lfloor Q/q^r \rceil \cdot c'_0, \lfloor Q/q^r \rceil \cdot c'_1) \in R^2_Q$
3: Return $\mathsf{ct}^* = (c^*_0, c^*_1) \in R^2_Q$
```

After the modulus switching, we obtain $c'_0 + c'_1 = \lfloor q^r/p \rceil \cdot M(X) + e'(X) \in R_{q^r}$ where M is the plaintext of the input ciphertext ct' and $\|e'\|_\infty$ is small. To guarantee the correct decryption after the modulus switching, we require $\|e'\|_\infty < q^r/2p$. As long as such condition is fulfilled, we stress that it is very important to choose the parameter q^r as small as possible to optimize the performance of the bootstrapping itself. Therefore, we investigate the probabilistic bound of e' thoroughly below.

The noise e' is a sum of two errors, e and e_{rnd} where e is the error of the input ciphertext and e_{rnd} is the rounding error obtained from the dividing-and-rounding. Assume that noise from the input ciphertext is smaller than $q^r/4p$, it is sufficient to bound the rounding error by $q^r/4p$ to guarantee the correct decryption. In other words, for $e_{rnd} = e^0_{rnd} + e^1_{rnd} \cdot s$ where

$$e^i_{rnd} = \frac{q^r}{Q_{in}} \cdot c_i - \left\lfloor \frac{q^r}{Q_{in}} \cdot c_i \right\rceil \ (i = 0, 1).$$

From the RLWE assumption, the coefficients of c_0 and c_1 is indistinguishable from uniformly sampled random numbers over $\mathbb{Z}_{Q_{in}}$. In practical scenarios, it is common to assume that the secret key s is uniformly sampled from the ternary set $\{-1, 0, 1\}$ with a hamming weight $\|s\|_1$. Hence, each coefficient of the error polynomial e can be considered as a sum of $\|s\|_1 + 1$ uniformly distributed variables over $[-0.5, 0.5]$. In prior works, the worst case bound $\|s\|_1 + 1$ is leveraged to estimate the rounding error bound [6].

We tighten this bound using a probabilistic bound as proposed in the works on CKKS bootstrapping [2, 32]. Lee et al. [32] pointed out that the sum of h uniformly distributed variables essentially follows the Irwin-Hall distribution, and can be bounded by $1.81\sqrt{h}$ with failure probability less than 2^{-15} [2]. Therefore, in a heuristic approach, we can refine the bound of the size of the coefficients of e_{rnd} as follows:

$$\|e_{rnd}\|_\infty \leq 1.81\sqrt{1 + \|s\|_1}.$$

Therefore, substituting this back to the error bound $\|e_{rnd}\|_\infty \leq q^r/4p$, we can obtain the following bound for the plaintext modulus q^r for LUT evaluation, as follows.

$$q^r > 7.24p \cdot \sqrt{1 + \|s\|_1} \tag{1}$$

3.3 LUT Evaluation

Now we briefly convey the basic idea of the LUT evaluation method of ours. In our functional bootstrapping pipeline, we evaluate the LUT on the input message residing in the commutative ring $\mathbb{Z}_{q^r}$ using the polynomials in order to obtain the result in the finite field $\mathbb{Z}_q$. However, evaluation of an arbitrary function over $\mathbb{Z}_{q^r}$ is generally a challenging task since only a small number of functions have polynomial representations. Therefore, in order to evaluate an arbitrary function, we leverage a similar idea to the digit extraction algorithm. Roughly speaking, we iteratively evaluate the polynomial while dividing the message homomorphically to evaluate the arbitrary function.

As will be explained in the next section, we take advantage of certain polynomials with a special property. To be precise, these polynomials 'selectively' remove the LSB, compared to the digit extraction polynomials which extract the LSB regardless of the value of LSB. With these (selectively) polynomials removing LSB, we can construct a homomorphic selector which iteratively operates on the LSB. A detailed description of this algorithm will be given in the next section.

4 Evaluation of look-up tables

Before we elaborate on the details of our LUT evlauation algorithm, let us discuss the homomorphic division by q over the commutative ring $\mathbb{Z}_{q^r}$. Generally, division by q is not a well-defined operation since it is a zero divisor. However, in the context of FV cryptosystem, if the message in $\mathbb{Z}_{q^r}$ is a multiple of q, it can be homomorphically divided by simply changing the plaintext modulus to q^{r-1}. Note that the resulting message will not be an element of the modulo ring $\mathbb{Z}_{q^r}$, but an element of $\mathbb{Z}_{q^{r-1}}$. To put it in another way, it is essentially a change of the base ring while dividing the input by q. In the later sections, we will abuse the notation of regular division for this operation. *i.e.,* , for $x \in \mathbb{Z}_{q^r}$, a multiple of q, x/q denotes the ring element $x/q \in \mathbb{Z}_{q^{r-1}}$.

Now, suppose that we are given an arbitrary LUT $F : \mathbb{Z}_{q^r} \mapsto \mathbb{Z}_q$ for a prime q. Then, evaluating F is ultimately obtaining $q^{r-1} \cdot F(m) \in \mathbb{Z}_{q^r}$ for the input message m as discussed above. Hence, if there exists a polynomial representation of this function $q^{r-1}F(x)$, the LUT can be evaluated directly through a single polynomial evaluation. However, only a small number of LUTs can be evaluated in such a way, since most of the functions defined over the commutative ring $\mathbb{Z}_{q^r}$ are not functions with polynomial representations, so-called the *polyfunctions*. For example, even LUT for digit extraction does not have an explicit polynomial representation. Therefore, we utilize the homomorphic division by q between the polynomial evaluations similar to the conventional bootstrapping method. By adopting such approach, we can finally evaluate an arbitrary LUT. A detailed explanation will be given later in this section.

In Sec. 4.1, we provide useful lemmas on the polyfunctions and investigate the structure of the polynomials in $\mathbb{Z}_{q^\ell}$ for some positive integer ℓ. We utilize these results to construct a noble LUT evaluation method for the functional bootstrapping in Sec. 4.2. Finally, we select certain widely-used functions, the delta and the sign function, to apply our functional bootstrapping method to evaluate them while bootstrapping in Sec. 4.3. For a better readability, we will use the unsigned representation for the integers over the commutative ring $\mathbb{Z}_{q^\ell}$ in the following subsections.

4.1 Polyfunctions over $\mathbb{Z}_{q^\ell}$

We introduce a necessary and sufficient condition for polyfunctions over $\mathbb{Z}_{q^\ell}$, established by Guha and Dukkipati [25].

Proposition 1 ([25]). *If a function f over $\mathbb{Z}_{q^\ell}$ can be represented with a linear combination of the following functions, then f is a polyfunction.*

1. $u_0^\ell(x) = \begin{cases} 0 & \text{for } q \nmid x \\ x & \text{for } q \mid x \end{cases}$
2. $u_i^\ell(x)$, *i-th shift of* $u_0^\ell(x)$, *i.e.*, $u_i^\ell(x) = u_0^\ell(x-i)$ $(0 \le i < q)$
3. *j-th powers of* $u_i^\ell(x)$, *i.e.*, $\left(u_i^\ell(x)\right)^j = \begin{cases} 0 & \text{for } q \nmid x \\ x^j & \text{for } q \mid x \end{cases}$ $(0 \le i < q,\ 0 \le j < \ell)$

Note that in our notation $u_i^\ell(x)$, i is an index for shifting while it was the exponent related to the output in [25]. We substitute it as an exponent j described in the third item and we also unify the expression of $u_i(x)$ (in [25]) which is divided into two cases where $i = 0$ and $1 \le i \le \ell - 1$ by changing the bound of the exponent j.

The implication of proposition 1 is that if a function f is a polyfunction, then restricting its domain to the congruence class of i modulo q is a polynomial with degree at most ℓ for any $0 \le i < q$. A very simple example will be the digit extraction polynomials $\{G_i\}$ which are commonly utilized in the state-of-the-art bootstrapping methods for FV/BGV scheme. In a nutshell, i-th digit extraction polynomial G_i is a polynomial which satisfies $G_i(x) = [x]_q \pmod{q^i}$ for any $x \in \mathbb{Z}$. Observe that G_i over $\mathbb{Z}_{q^i}$ is essentially a constant function at any congruence class of j modulo q, since $G_i(j + q \cdot x) = j$ regardless the value of x.

Now let us discuss the properties of the polynomial representation of polyfunctions and how they can be obtained. We first introduce the definition of the Smarandache function.

Definition 1 (Smarandache function). *The Smarandache function $\mu(\cdot)$ is defined as $\mu(x) = \min\{i \in \mathbb{N} : x \mid i!\}$.*

It is easy to show that $\mu(q^\ell) \le q\ell$ since the number of multiples of q is equal to or more than ℓ in successive $p\ell$ integers. Therefore, we use $q\ell$ as the upper bound of $\mu(q^\ell)$ in the later sections. *i.e.*, $\mu(q^\ell) = O(q\ell)$. Interestingly, it is known that any polyfunctions over $\mathbb{Z}_{q^\ell}$ can be represented with a polynomial with degree less than $\mu(q^\ell)$. We state it more formally below in Lem. 1.

Lemma 1 ([30]). *If $f : \mathbb{Z}_{q^\ell} \mapsto \mathbb{Z}_{q^\ell}$ is a polyfunction, there exists a polynomial representation of f with degree smaller than $\mu(q^\ell)$.*

In [18], the authors mentioned an efficient method to find such a 'compact' polynomial representation using the Newton interpolation. It is essentially a direct adoption of the divided difference method, which is a common interpolation technique in numerical analysis.

4.2 Our Method

In this subsection, we investigate the details of our noble LUT evaluation technique. In our LUT evaluation algorithm, we leverage the basis polynomials u_j^i. Observe that u_j^i is always a multiple of q, we can homomorphically divide the output by q. For the input x, $u_j^i(x)/q$ is $(x-j)/q$ only if $x = j \pmod{q}$ and zero if $x \neq j \pmod{q}$. Here, note that $(x-j)/q$ is essentially 'upper $i-1$ digits of x', this can be regarded as a homomorphic selector of the upper digit based on the last digit, in base-q representation. From this observation, we deduce that $(u_{j_{r-1}}^1)^0 \left(u_{j_{r-2}}^2 \left(\dots u_{j_1}^{r-1} \left(u_{j_0}^r(x)/q\right)/q \dots\right)/q\right)$ is a function which is zero except for one point $x = \overline{j_{r-1} \dots j_0}$ for $j_{r-1} \neq 0$. (Note that j_{r-1} should not be zero, otherwise it will always return one regardless of the input. To cover the case $j_{r-1} = 0$, we can simply add a constant to the input to make the most significant bit (MSB) nonzero.) Therefore, an arbitrary LUT can be homomorphically computed by evaluating this polynomial for every possible combination of $j_{r-1}, \dots, j_0$, multiply the LUT value and add them altogether.

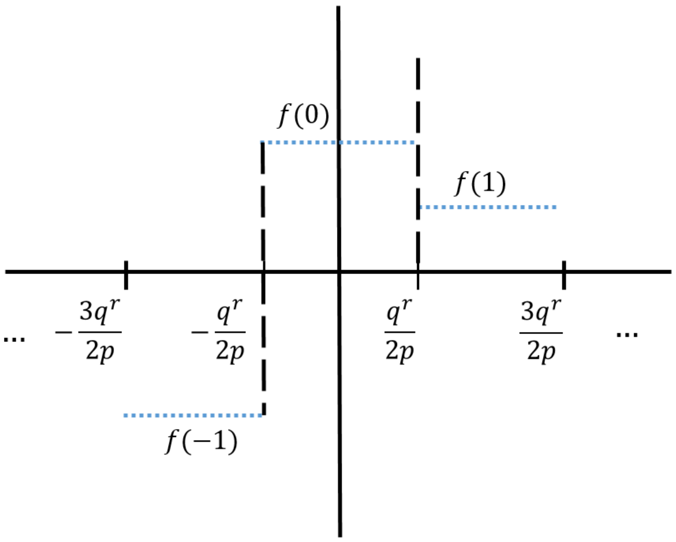

Fig. 4: The shape of LUT in functional bootstrapping. The dots denotes the function values at each integer points.

However, this general method suffers from a high computational cost and makes functional bootstrapping almost infeasible. In this work, we focus on optimizing a more specific case of LUT that is used in functional bootstrapping. After the modulus switch in the bootstrapping procedure, the input message m is associated with multiple values $\lfloor q^r/p \rceil \cdot m + e$ where $-q^r/2p \leq e \leq q^r/2p$. As $\lfloor q^r/p \rceil \cdot m$ is a fixed value for each m, $\lfloor q^r/p \rceil \cdot m + e$ are consecutive values in the integer domain. Since these values originate from a single input, the output of the LUT evaluation of the functional bootstrapping should be identical for all of these values. Consequently, the consistency in the output for consecutively associated values from a single input implies the LUT seems like a step function, regardless of the nature of the target function, as shown in Figure 4. From this fact, we devise a fast and optimized LUT evaluation method for our noble functional bootstrapping.

Step function We first commence by covering the LUT which has a form of step function, which is the most basic case of the step function-style LUT. Let the LUT $F : \mathbb{Z}_{q^r} \mapsto \mathbb{Z}_q$ is defined as follows,

$$F(x) = \begin{cases} 0 & \text{if } x < B \\ 1 & \text{otherwise} \end{cases}$$

for some bound $B \in \mathbb{Z}_{q^r}$. Without loss of generality, we suppose that $B \geq q^{r-1}$ since $F'(x) := 1 - F(x-B)$ is essentially another step function with bound $q^r - B \geq q^{r-1}$.

Now, let $\overline{b_{r-1}b_{r-2}\dots b_0}$ be the base q representation of B. Observe that if the LSB of the input x is smaller than b_0, the LUT F returns 0 if and only if its upper $r-1$ bits are less than $\overline{b_{r-1}\dots b_2(b_1+1)}$ and returns 1 otherwise. On the other hand, if the LSB of the input x is equal or bigger than b_0, the LUT F returns 0 if and only if its upper $r-1$ bits are less than $b_{r-1}\dots b_2 b_1$, and 1 otherwise. Therefore, we can evaluate the LUT F by dividing it into two sub-LUT's $F_1^{r-1}, F_2^{r-1} : \mathbb{Z}_{q^{r-1}} \mapsto \mathbb{Z}_q$, defined as follows:

$$F_1^{r-1} = \begin{cases} 0 & \text{if } x < \overline{b_{r-1}\dots(b_1+1)} \\ 1 & \text{otherwise} \end{cases},$$

$$F_2^{r-1} = \begin{cases} 0 & \text{if } x < \overline{b_{r-1}\dots b_1} \\ 1 & \text{otherwise} \end{cases}.$$

From this relation, we stress that the LUT F can be computed utilizing the polynomials u_i^r and evaluation of sub-LUTs F_1^{r-1} and F_2^{r-1}. Observe that $u_i^r(x)/q$ outputs the upper $r-1$ bits if the LSB of x is i, and zero otherwise for any input x. Subsequently, for any value x with LSB i, we can obtain $F(x)$ by evaluating $F_1^{r-1}(u_i^r/q)$ if $i < b_0$ and $F_2^{r-1}(u_i^r/q)$ if $b_0 \leq i$. Recall that we assumed that $B \geq q^{r-1}$, new bounds $\overline{b_{r-1}\dots(b_1+1)}$ and $\overline{b_{r-1}\dots b_1}$ are strictly bigger than zero, and thus $F_1^{r-1}(0) = F_2^{r-1}(0) = 0$. Hence, the LUT F can be evaluated by computing the sum of $F_1^{r-1}(u_i^r(x)/q)$ $(0 \leq i < b_0)$ and $F_2^{r-1}(u_i^r(x)/q)$ $(b_0 \leq i < q)$. Since we essentially evaluate the two identical LUTs for the cases $0 \leq i < b_0$ and $b_0 \leq i < q$ respectively, the sum of LUTs can be integrated as follows.

$$F(x) = F_1^{r-1}\left(\sum_{0 \leq i < b_0} u_i^r(x)/q\right) + F_2^{r-1}\left(\sum_{b_0 \leq i < q} u_i^r(x)/q\right)$$

Now, it remains to evaluate two sub-LUTs F_1^{r-1} and F_2^{r-1}. We remark that F_1^{r-1} and F_2^{r-1} are again step function over $\mathbb{Z}_{p^{r-1}}$ with bounds $B_1 := b_{r-1}\dots b_2(b_1+1)$ and $B_2 := b_{r-1}\dots b_2 b_1$. Hence, they can also be divided into sub-LUTs analogously. In a similar way that F_1^{r-1} and F_2^{r-1} do not contain any information on the LSB b_0 in them, the sub-LUTs of F_1^{r-1} and F_2^{r-1} are also independent from the LSB b_1+1 and b_1, of the bounds B_1 and B_2, respectively. As B_1 and B_2 only differ by the LSB, the sub-LUTs of F_1^{r-1} and F_2^{r-1} should be identical. Let us denote them by F_1^{r-2} and F_2^{r-2}. Then, they are defined as follows:

$$F_1^{r-2} = \begin{cases} 0 & \text{if } x < b_{r-1}\dots(b_2+1) \\ 1 & \text{otherwise} \end{cases}$$

$$F_2^{r-2} = \begin{cases} 0 & \text{if } x < b_{r-1}\dots b_2 \\ 1 & \text{otherwise} \end{cases}.$$

Analogous to the evaluation of F, F_1^{r-1} and F_2^{r-1} can be evaluated in a recursive manner. For a better readability, let us denote by $x_1 := \sum_{0 \leq i < b_0} u_i^r(x)/q$ and $x_2 := \sum_{b_0 \leq i < q} u_i^r(x)/q$. Then, it follows that

$$\begin{aligned} F(x) = F_1^{r-2}&\left(\sum_{0 \leq i \leq b_1} u_i^{r-1}(x_1)/q + \sum_{0 \leq i < b_1} u_i^{r-1}(x_2)/q\right) \\ + F_2^{r-2}&\left(\sum_{b_1 < i < q} u_i^{r-1}(x_1)/q + \sum_{b_1 \leq i < q} u_i^{r-1}(x_2)/q\right) \end{aligned}$$

since $F_1^{r-2}(0) = F_2^{r-2}(0) = 0$ due to the condition $B \geq q^{r-1}$.

Note that the input for the same LUTs are integrated and hence it only requires the evaluation of four polynomials $\sum_{0 \leq i \leq b_1} u_i^{r-1}(x_1)/q$, $\sum_{0 \leq i < b_1} u_i^{r-1}(x_2)/q$, $\sum_{b_1 < i < q} u_i^{r-1}(x_1)/q$ and $\sum_{b_1 \leq i < q} u_i^{r-1}(x_2)/q$, and

LUTs F_1^{r-2} and F_2^{r-2} over $\mathbb{Z}_{q^{r-2}}$. These LUTs F_1^{r-2} and F_2^{r-2} can be iteratively computed via four polynomial evaluations and two LUT evaluations in the smaller dimension, in a similar manner. (Note that the condition $B \geq q^{r-1}$ plays an important role here, making the value of the LUT zero for the zero input.) Subsequently, at the end of the iteration, it remains to evaluate two LUTs F_1^1 and F_2^1 over $\mathbb{Z}_q$, defined as follows.

$$F_1^1 = \begin{cases} 0 & \text{if } x < b_{r-1} + 1 \\ 1 & \text{otherwise} \end{cases}$$

$$F_2^1 = \begin{cases} 0 & \text{if } x < b_{r-1} \\ 1 & \text{otherwise} \end{cases} .$$

We stress that any function over $\mathbb{Z}_q$ is a polyfunction, both of them have polynomial representations with integer coefficients and each LUT can be evaluated with one polynomial evaluation. From these recurrence relations, we can devise an algorithm for step function evaluation. Below, the exact algorithm based on this approach is presented in Alg. 2.

Algorithm 2 Polynomial evaluation for step function

```
Input: An input $x \in \mathbb{Z}_{q^r}$, LUT $F$ of a step function with bound $B$.
Output: $F(x) \in \mathbb{Z}_q$
1: parse $B = b_{r-1}b_{r-2}\dots b_0$ in digit $q$ representation.
2: $x_1, x_2 \leftarrow 0 \in \mathbb{Z}_{q^r}, x \in \mathbb{Z}_{q^r}$
3: for $i = 0; i < r-1; i+ = 1$ do
4:     $x_1 \leftarrow \sum_{0 \leq j \leq b_i} u_j^{r-i}(x_1) + \sum_{0 \leq j < b_i} u_j^{r-i}(x_2) \pmod{q^{r-i}}$
5:     $x_2 \leftarrow \sum_{b_i < j < q} u_j^{r-i}(x_1) + \sum_{b_i \leq j < q} u_j^{r-i}(x_2) \pmod{q^{r-i}}$
6:     $x_1, x_2 \leftarrow x_1/q \in \mathbb{Z}_{q^{r-i-1}}, x_2/q \in \mathbb{Z}_{q^{r-i-1}}$
7: end for
8: Return $F_1^1(x_1) + F_2^1(x_2) \in \mathbb{Z}_q$
```

Depth and Time complexity Analysis Now we analyze the depth consumption and the time complexity of our method. For the time complexity, we consider the number of multiplication (key-switching) which takes a large portion of the computations time in the functional bootstrapping. We remark that they are solely dependent on the degree of the polynomials utilized during the evaluation. It is known that any polynomial of degree d can be evaluated with $2\sqrt{d}$ non-scalar multiplications while consuming $\lceil \log d \rceil$ levels, utilizing the Paterson-Stockmeyer algorithm [21, 41]. Based on this fact, we conduct the time complexity and the depth consumption analysis for our method.

In the i-th iteration, four polynomials over $\mathbb{Z}_{q^{r-i}}$ is evaluated. As every polynomial over $\mathbb{Z}_{q^{r-i}}$ has degree at most $\mu(q^{r-i}) \approx q(r-i)$ by Lemma 1, their evaluation requires $\approx 4 \cdot 2\sqrt{q(r-i)} = 8\sqrt{q(r-i)}$ key-switching operations and $\approx \log(qr)$ levels of depth consumption. Hence, during r iterations of the algorithm, we perform $\sum_{i=0}^{r-1} 8\sqrt{q(r-i)} \approx \frac{16}{3}\sqrt{r^3 q}$ key-switching operations and consume $\sum_{i=0}^{r-1} \log(q(r-i)) = r\log q + \log r!$ multiplicative depths.

Optimization Our method can be optimized under certain circumstances. If $b_i = 0$ for some $0 \leq i < r$, the polynomial $\sum_{0 \leq j < b_i} u_j^{r-i}(x_1)$ is essentially zero and thus its evaluation can be skipped. Moreover, if B has consecutive ℓ zeros in the LSB, *i.e.,* $b_0 = b_1 = \cdots = b_{\ell-1} = 0$, x_1 is essentially zero for ℓ iterations. In this case, only one polynomial evaluation and two polynomial evaluations are necessitated at the beginning and in each iteration of the algorithm, respectively.

We also remark that there exists a time-depth trade-off for our LUT evaluation as well. Our method essentially compares each digit of the bound and the input, it can be realized as a multivariate function over $\mathbb{Z}_q$ with digits of the input message as variables. Note that each digit can be obtained while consuming

$i \cdot \log q$ during the digit extraction algorithm of the conventional FV bootstrapping, the multivariate polynomial can be evaluated with output depth $(r+1)\log q$. This depth asymptotically improves the depth consumption of the aforementioned method, while its asymptotic time complexity is increased.

Evaluation of arbitrary LUT The evaluation of step function can be naturally extended to the case of LUT with multiple intervals. Naïvely, we can decompose a LUT with k intervals into a linear combination of k step functions. In particular, let the LUT with k intervals is defined as follows:

$$F(x) = \begin{cases} \alpha_1 & \text{if } x < B_1 \\ \alpha_2 & \text{if } B_1 \le x < B_2 \\ \ldots \\ \alpha_{k-1} & \text{if } B_{k-2} \le x < B_{k-1} \\ \alpha_k & \text{otherwise} \end{cases}$$

Then, we can represent it as $F(x) = \alpha_1 + \sum_{i=1}^{k-1}(\alpha_{i+1} - \alpha_i)F_i(x)$ such that $F_i(x)$, step function, is defined as follows:

$$F_i(x) = \begin{cases} 0 & \text{if } x < B_i \\ 1 & \text{otherwise} \end{cases} \quad \text{for } 1 \le i < k$$

Hence, we have the capability to evaluate an arbitrary step function during functional bootstrapping. However, employing naïve approach entails performing k time-consuming evaluation of k step functions. We remark that the time complexity can be mitigated by constructing a recurrence relation akin to the step function case. Specifically, we categorize the cases based on the LSB of the given bounds $B_1, \ldots, B_{k-1}$. Nonetheless, this approach involves an exhaustive classification of edge cases, given the existence of $k \cdot (k-1)! = k!$ possible orderings of the LSB of the input value and $k-1$ bounds. While it reduces the number of polynomials evaluated throughout the LUT evaluation, this classification should be considered carefully. Therefore, for better scalability, it is advisable to use the naïve method except for certain use-cases demand optimization.

4.3 Concrete Examples: Delta & Sign Functions

In this section, we apply our functional bootstrapping technique to some selected functions, namely delta and sign function. The delta function is a special function that returns 1 when the input is zero and 0 for other cases. To put it in a functional form,

$$\mathit{Delta}(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$$

After modulus switching from q^r to Q in the modulus switching phase, the ciphertext can be regarded as an encryption of $\lfloor q^r/p \rceil \cdot m + e$ where $-q^r/2p \le e < q^r/2p$, as discussed in the beginning of this section. Therefore, the evaluated LUT during the functional bootstrapping of the delta function should be

$$F(x) = \begin{cases} 0 & \text{if } x < -q^r/2p \\ 1 & \text{if } -q^r/2p \le x < q^r/2p \\ 0 & \text{otherwise.} \end{cases}$$

Even though the function F seems like a LUT , we can convert it into a step function case by shifting a domain. Since the message space of FV is $(-p/2, p/2]$ for the plaintext modulus p, the delta function is converted to the following function when we substitute the input x to $x - \lfloor p/2 \rceil$:

$$\mathit{Delta}(x) = \begin{cases} 0 & \text{if } x < p-1 \\ 1 & \text{otherwise} \end{cases}$$

Then, the LUT F is also changed as follows:

$$F(x) = \begin{cases} 0 & \text{if } x < q^r - q^r/2p \\ 1 & \text{otherwise} \end{cases}$$

As evaluating a functional bootstrapping with a LUT with three intervals takes more complexity than a step function, we can reduce the complexity by transforming the delta function as we mentioned above. We can exploit the delta function to extract items with the same attribute as the target value. A more detailed scenario is described in the next section.

Another useful function is a sign function which returns -1 when the input is negative and 1 for other cases. It also has various ways to utilize this function. For example, The sign function is the main building block of comparison which is a sign value of subtraction of two inputs and the comparison is a useful function used in several applications such as decision trees, sorting algorithms, or SQL queries in the database.

The sign function is a typical step function that can be written as follows:

$$Sign(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

Similar to the delta function, the LUT F from $\mathbb{Z}_{q^r}$ to $\mathbb{Z}$ is as following:

$$F(x) = \begin{cases} -1 & \text{if } x < -q^r/2p \\ 1 & \text{otherwise} \end{cases}$$

As both LUTs have the same form as the step function, we can easily compute them during the functional bootstrapping according to the Algorithm 2. Note that the depth and time complexity are asymptotically the same with analysis in Section 4.2 since the evaluation process is not different from the Algorithm 2. As a result, it requires $\sum_{i=0}^{r-1} 8\sqrt{q(r-i)} \approx \frac{16}{3}\sqrt{r^3 q}$ key-switching operations and consumes $\sum_{i=0}^{r-1} \log(q(r-i)) = r\log q + \log r!$ depths. In the following section, a concrete performance of these two functions are provided under an appropriate parameter set.

5 Experiments & Applications

Employing the functional bootstrapping algorithm, specifically tailored for the delta and sign functions, we present a proof-of-concept level implementation within the context of the FV scheme.

We apply functional bootstrapping to address specific functions within practical applications, notably in the domains of PSI [8, 7, 15] or circuit PSI [28], and PIR [14, 40, 22]. Remark that the delta and sign functions described in Section 4 have extensive utility in real-world scenarios.

The subsequent sections present a comprehensive benchmark analysis of the performance of functional bootstrapping and introduce its efficacy in addressing the specified functions within the FV scheme. Additionally, we present practical use cases and demonstrate the utilization of functional bootstrapping.

5.1 Implementation

We provide a proof-of-concept implementation of our LUT evaluation for the delta and sign functions. We implemented our method in Lattigo v5 [37]. All experiments were performed on a machine with Intel(R) Xeon(R) Platinum 8268 @ 2.90GHz CPU and 192GB RAM running Ubuntu 20.04.2 LTS. We used two ring dimensions $N = 2^{15}$ and $N = 2^{16}$, which are commonly used for an evaluation of a circuit with a sufficiently large depth such as bootstrapping. Each parameter set $(p, q, r, ||s||_1)$ satisfies the condition of equation 1 to guarantee the low bootstrapping failure probability and achieves an estimated security level of $\geq$128-bits. Table. 1 summarizes the parameter set that we used in the implementation.

ID	p	q	r	$\|s\|_1$	N	$\log PQ$
I	≤ 700	17	4	256	2^{15}	840
II	≤ 12000	17	5	256	2^{16}	1700

Table 1: Parameter set for the implementation. p, q denote the input and output plaintext modulus, respectively.

In Lattigo, the choice of the plaintext modulus is confined to a power-of-two NTT-friendly prime number with moderate size. Such a limitation causes unexpected behavior of the existing baseline algorithms in the Lattigo library. Therefore, we implemented several (unoptimized) algorithms such as the Paterson-Stockmeyer polynomial evaluation algorithm, matrix multiplication, encoding and decoding. Due to such a reason, we believe that there still is room for further optimization in our code. An illustrative example is the application of the lazy polynomial Baby-Step Giant-Step (BSGS) algorithm [34], which holds the promise of minimizing the relinearization operations during polynomial evaluation. In addition, we recognize the opportunity to enhance the time complexity of polynomial evaluation from a sublinear to a logarithmic scale through the utilization of the polynomial evaluation technique proposed by Okada et al. [39]. We expect that these potential optimizations will refine the efficiency and overall performance of our code.

5.2 Benchmarks

We present experimental results derived from our implementation of the functional bootstrapping method. The execution time associated with functional bootstrapping is presented in Table 2. As described in Section 4.3, the LUTs employed for the computation of delta and sign functions exhibit identical characteristics, sharing the same number of intervals and the ranges within those intervals. The only difference is the output values of each interval. Consequently, the evaluation time for the functional bootstrapping with both functions remains consistent. During the measurement of elapsed time, we constrained the dataset size to ensure that the data could be efficiently packed within a single ciphertext. Note that the initial plaintext modulus p, correlated with the number of messages packable in the ciphertext, does not affect the execution time within the scope of our experimental setup.

ID	Elapsed Time	BTS	Remaining Levels
I	46.5 sec	$\approx$ 595 bits	$\approx$ 44 levels
II	171.4 sec	$\approx$ 840 bits	$\approx$ 104 levels

Table 2: Evaluation time, noise consumption of functional bootstrapping (BTS), and remaining level after functional bootstrapping for delta/sign function

In addition, Table 2 also shows the noise consumption and remaining level, resulting from it, of the functional bootstrapping. The ring dimension N affects the magnitude of noise generated during homomorphic evaluation, with a parameter set featuring $N = 2^{16}$ leading to a greater bit consumption during the bootstrapping process. In addition, we calculate the remaining level in the situation where the plaintext modulus is q which is smaller than p. Therefore, the remaining noise level is markedly higher in comparison to situations where the modulus remains p.

5.3 Applications

The application of functional bootstrapping with delta or sign functions holds significant potential in various real-world scenarios. One prominent example is Circuit-based Private Set Intersection (Circuit

PSI) protocol [28], which is a generalization of the plain PSI. PSI is a 2PC protocol that computes the intersection $X \cap Y$ of private sets X and Y of each participating party, without leaking any information on X and Y rather than the intersection. The functionality of Circuit PSI is essentially the same, while a function is evaluated over the intersection privately instead of just computing the intersection.

To realize this Circuit PSI in HE-based PSI protocols, evaluation of delta function is necessary, since the output is zero if the item is contained in the intersection $X \cap Y$, and a random non-zero integer otherwise in the current HE-based PSI protocols [8, 7, 15]. However, a naïve evaluation of the delta function is almost infeasible, since it requires an evaluation of a polynomial with a size of the plaintext which is typically large in these settings moreover, the remaining level after the PSI protocol is low and thus it requires a bootstrapping. We stress that our functional bootstrapping can be utilized, by proceeding the functional bootstrapping with delta function evaluation.

<Database 'DB'>

User ID	Age	...	Grade
1545	25	...	5
2418	43	...	3
197	16	...	5
1886	37	...	4

Query: SELECT GRADE FROM 'DB' WHERE ID=1886
$\rightarrow$ For each row, return "delta(ID - 1886) * GRADE"

Fig. 5: PIR example for the functional bootstrapping

A PIR [14, 40, 22] is another application scenario of how functional bootstrapping can be effectively employed. In a PIR setting, a user seeks to retrieve specific information from a server without disclosing the identity of the item being retrieved. If we apply HE to achieve this functionality, it requires an equality check that can be facilitated through the utilization of the delta function.

Consider the scenario depicted in Figure 5, where the goal is to extract attributes (e.g., Grade) of entities with a certain index from the database. The process involves subtracting a value of certain index from each row and selecting entities for which the result is zero. By using the delta function, we can transform these results into 1 when the subtraction is zero and 0 for others. Subsequently, obtaining attributes for specific indexes becomes straightforward by multiplying the delta function result with the corresponding attribute values. This application underscores the versatility of functional bootstrapping in privacy-preserving information retrieval scenarios.

Moreover, when we evaluate these queries, functional bootstrapping may enhance the flexibility of the computation. In the previous bootstrapping method, all messages needed to be encrypted with the same plaintext modulus since there were no operations that changed the plaintext modulus. However, different data may have different domains (ranges) and there are appropriate parameters for encrypting each of them. For instance, as the range of 'User ID' is much wider than 'Grade' in Figure 5, we can encrypt the latter data with a smaller plaintext modulus and it might improve both time and space complexity during the overall circuit evaluation.

6 Conclusion & Future Works

In this paper, we introduce a new bootstrapping method called the functional bootstrapping which generalizes the existing bootstrapping techniques for FV scheme. It extends the functionality of the bootstrapping by allowing us to perform an arbitrary function during the bootstrapping, without any additional depth consumption than the bootstrapping depth. As a result, the consumed depth and the computational complexity of a large depth circuit can be mitigated asymptotpically. It also gives more flexibility in the parameter selection by allowing us the plaintext modulus conversion.

It is achieved by a development of a new algorithm to evaluate the arbitrary look-up tables over the commutative ring $\mathbb{Z}_{p^\ell}$. An optimized LUT evaluation algorithm for a specific class of LUT, namely the step-function, is also proposed. In addition, we demonstrate its application in handling a special functions like delta and sign functions. We implement this approach using the open-source FHE library, Lattigo [37] and provide benchmark analyses.

It is worth noting that our method can also be applied to the bootstrapping for BGV cryptosystem, or even CKKS cryptosystem. Recall that in our LUT evaluation method, the input message does not need to be structured. In other words, our method can be utilized for any kind of packing method as long as we choose the correct parameter. Although similar to the discussion given in [6], we expect that the functional bootstrapping for BGV ciphertext will be asymptotically worse than the functional bootstrapping for FV ciphertext due to the rounding error bound and limited choice of p and q. We believe that optimizing our functional bootstrapping framework for these cases can be an interesting research topic in the future.

References

1. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full rns variant of fv like somewhat homomorphic encryption schemes. In: International Conference on Selected Areas in Cryptography. pp. 423–442. Springer (2016)
2. Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 587–617. Springer (2021)
3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
5. Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 34–54. Springer (2019)
6. Chen, H., Han, K.: Homomorphic lower digits removal and improved fhe bootstrapping. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 315–337. Springer (2018)
7. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled psi from fully homomorphic encryption with malicious security. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1223–1237 (2018)
8. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1243–1255 (2017)
9. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full rns variant of approximate homomorphic encryption. In: International Conference on Selected Areas in Cryptography. pp. 347–368. Springer (2018)
10. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)
11. Cheon, J.H., Kim, M., Kim, M.: Optimized search-and-compute circuits and their application to query evaluation on encrypted data. IEEE Transactions on Information Forensics and Security **11**(1), 188–199 (2015)
12. Cheon, J.H., Kim, M., Kim, M.: Search-and-compute on encrypted data. In: Financial Cryptography and Data Security: FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers. pp. 142–159. Springer (2015)
13. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: international conference on the theory and application of cryptology and information security. pp. 3–33. Springer (2016)
14. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. Journal of the ACM (JACM) **45**(6), 965–981 (1998)
15. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled psi from homomorphic encryption with reduced computation and communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 1135–1150 (2021)
16. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 617–640. Springer (2015)

17. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. **2012**, 144 (2012)
18. Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On polynomial functions modulo pe and faster bootstrapping for homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 257–286. Springer (2023)
19. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
20. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
21. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 129–148. Springer (2011)
22. Gentry, C., Halevi, S.: Compressible fhe with applications to pir. In: Theory of Cryptography Conference. pp. 438–464. Springer (2019)
23. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
24. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference. pp. 75–92. Springer (2013)
25. Guha, A., Dukkipati, A.: An algorithmic characterization of polynomial functions over $\mathcal{F}_{p^n}$. Algorithmica **71**(1), 201–218 (Jun 2013). https://doi.org/10.1007/s00453-013-9799-7, http://dx.doi.org/10.1007/s00453-013-9799-7
26. Halevi, S., Polyakov, Y., Shoup, V.: An improved rns variant of the bfv homomorphic encryption scheme. In: Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings. pp. 83–105. Springer (2019)
27. Halevi, S., Shoup, V.: Bootstrapping for helib. Journal of Cryptology **34**(1), 7 (2021)
28. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS (2012)
29. Iliashenko, I., Zucca, V.: Faster homomorphic comparison operations for bgv and bfv. Proceedings on Privacy Enhancing Technologies **2021**(3), 246–264 (2021)
30. Keller, G., Olson, F.R.: Counting polynomial functions (mod pn) (1968)
31. Kim, M., Lee, H.T., Ling, S., Wang, H.: On the efficiency of fhe-based private queries. IEEE Transactions on Dependable and Secure Computing **15**(2), 357–363 (2016)
32. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In: Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40. pp. 618–647. Springer (2021)
33. Lee, Y., Lee, J.W., Kim, Y.S., Kim, Y., No, J.S., Kang, H.: High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 551–580. Springer (2022)
34. Lee, Y., Lee, J.W., Kim, Y.S., Kim, Y., No, J.S., Kang, H.: High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 551–580. Springer (2022)
35. Liu, Z., Wang, Y.: Amortized functional bootstrapping in less than 7ms, with $\tilde{O}(1)$ polynomial multiplications. Cryptology ePrint Archive, Paper 2023/910 (2023), https://eprint.iacr.org/2023/910, https://eprint.iacr.org/2023/910
36. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) **60**(6), 1–35 (2013)
37. Mouchet, C.V., Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Lattigo: A multiparty homomorphic encryption library in go. In: Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography. pp. 64–70. No. CONF (2020)
38. Narumanchi, H., Goyal, D., Emmadi, N., Gauravaram, P.: Performance analysis of sorting of fhe data: integer-wise comparison vs bit-wise comparison. In: 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA). pp. 902–908. IEEE (2017)
39. Okada, H., Player, R., Pohmann, S.: Homomorphic polynomial evaluation using galois structure and applications to bfv bootstrapping. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 69–100. Springer (2023)
40. Park, J., Tibouchi, M.: Shecs-pir: somewhat homomorphic encryption-based compact and scalable private information retrieval. In: European Symposium on Research in Computer Security. pp. 86–106. Springer (2020)

41. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing **2**(1), 60–66 (1973)
42. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
43. Tan, B.H.M., Lee, H.T., Wang, H., Ren, S., Aung, K.M.M.: Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. IEEE Transactions on Dependable and Secure Computing **18**(6), 2861–2874 (2020)