Leveled Functional Bootstrapping via External Product Tree

Zhihao Li[1], Xuan shen[2,3], Xianhui Lu[2,3], Ruida Wang[2,3], Yuan Zhao[1], Zhiwei Wang[2,3], and Benqiang Wei[2,3]

[1] Ant Group
[2] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. Multi-input and large-precision lookup table (LUT) evaluation pose significant challenges in Fully Homomorphic Encryption (FHE). Currently, two modes are employed to address this issue. One is tree-based functional bootstrapping (TFBS), which uses multiple blind rotations to construct a tree for LUT evaluation. The second is circuit bootstrapping, which decomposes all inputs into bits and utilizes a CMux tree for LUT evaluation. In this work, we propose a novel mode that is leveled functional bootstrapping. This mode utilizes the external product tree to perform multi-input functional bootstrapping. We implement TFBS and LFBS within the OpenFHE library. The results demonstrate that our method outperforms TFBS in both computational efficiency and bootstrapping key size. Specifically, for 12-bit and 16-bit input LUTs, our method is approximately two to three orders of magnitude faster than TFBS. Finally, we introduce a novel scheme-switching framework that supports large-precision polynomial and non-polynomial evaluations. The framework leverages digital extraction techniques to enable seamless switching between the BFV and LFBS schemes.

Keywords: Lookup Table (LUT) · Functional Bootstrapping · External Product · Scheme Switching.

1 Introduction

Fully homomorphic encryption (FHE) is a powerful cryptographic primitive that enables performing computations over encrypted data without decryption. In 2009, Gentry proposed the first fully homomorphic encryption scheme [22] and designed a bootstrapping procedure. This procedure can homomorphically compute the decryption function to reduce the noise of ciphertexts. Since then, FHE has undergone significant advancements and innovations, making it applicable to various privacy-related computing scenarios.

Typically, FHE schemes are typically categorized into three types. The first type includes the BGV [9] and BFV [8,21] schemes. These support exact polynomial evaluations with the 'Single Instruction Multiple Data' (SIMD) mode. The second type focuses on approximate arithmetic evaluations and is represented by the CKKS [14] scheme. This approach treats noise as part of the message, which offers advantages in performance and allows for residual multiplicative levels. The third type supports bit-level operations and achieves efficient bootstrapping procedures, represented by the FHEW [19] and TFHE [15] schemes.

Among them, the FHEW and TFHE schemes stand out for their efficiency and flexibility in bootstrapping. They support different computational modes, such as FHE mode based on the functional bootstrapping (FBS) [33,17], and LHE mode based on circuit bootstrapping (CBS) [16,36]. In the FHE mode, a key advantage is that an arbitrary function can be embedded into bootstrapping, which is widely utilized in numerous applications, such as evaluating non-linear functions in neural networks [7,31]. However, the precision of the FBS is limited by $4 \sim 5$ bits on the underlying parameters since the large-precision LUT evaluation needs to increase the parameters of the cryptosystem, which leads to significant performance degradation.

To deal with the muti-input LUT[4], Guimarãe et al. [24] proposed tree-based functional bootstrapping (TFBS). The TFBS algorithm decomposes the multi-input LUT into multiple small LUTs, which are associated together in a tree-based construction via functional bootstrapping and the special LWE-to-RLWE packing method (i.e., base-aware key switching). Although this method addresses the problem of parameter increases in functional bootstrapping, its exponential complexity relative to the large inputs is still prohibitive. On the other hand, scheme [5] presents the LHE mode to deal with the large precision LUT. The LHE mode needs to split all input into bits and then conduct the circuit bootstrapping. Following that, the output format of the ciphertext from circuit bootstrapping can be used to carry out LUT evaluation with the CMux gates tree. However, this approach still suffers from two inherent limitations: the first is the loss of parallelism during the functional bootstrapping phase, and the second is the inability to perform aggregation operations efficiently after bitwise decomposition.

1.1 Our Contributions

We propose a new LUT evaluation mode to address this issue. Our contributions can be divided into three main parts. Firstly, we introduce a conversion method from LWE to RGSW ciphertexts with multi-bits and develop a new multi-valued[5] bootstrapping algorithm. Secondly, we present the leveled functional bootstrapping, which enhances the efficiency of evaluating large circuits. We propose the new LUT Evaluation mode to solve this problem. Finally, we apply the proposed LFBS algorithm in scheme switching framework that can support large precision polynomial and non-polynomial evaluations.

[4] Multi-input FBS refers to a process in which the multivariate function $f(m_0, ..., m_{l-1})$ is evaluated during the bootstrapping procedure.

[5] Multi-valued FBS describes the case where multiple uni-variate functions are computed during bootstrapping.

Ciphertext Conversion (CC). We develop a new ciphertext conversion that can switch ciphertext LWE(m) to RGSW$(X^{\mathsf{Encode}(m)})$ instead of RGSW(m). This ciphertext conversion supports multi-bit message m for LWE. It's output and (multiple) test polynomials, through external product, enable multi-valued functional bootstrapping. Additionally, we tailor a multi-valued algorithm for this conversion to improve efficiency, requiring only one blind rotation. This ciphertext conversion is of independent interest and can serve as a new building block to enhance FHEW-like schemes.

Leveled Functional Bootstrapping. We design the leveled functional bootstrapping mode as illustrated in Fig. 1. The process is divided into three steps. Step 1 involves converting the input LWE ciphertext, while operating in FHE mode. Step 2 performs tree-based LUT evaluation through external products, executing functional bootstrapping in LHE mode. Finally, step 3 outputs the result of the LUT evaluation. Furthermore, we improve the RLWE packing operation used in the TFBS and LFBS schemes by utilizing the homomorphic trace technique, which significantly reduces the computational cost.

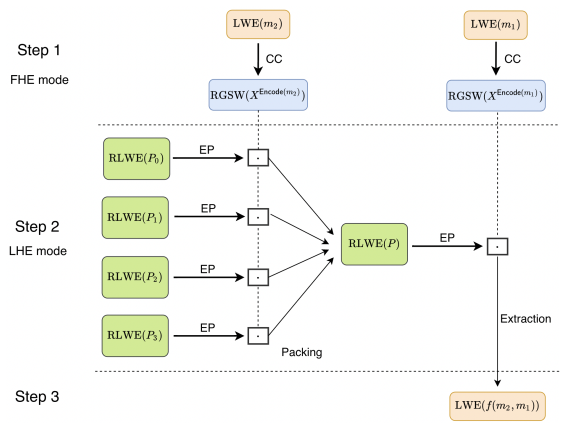


Fig. 1. The workflow of leveled functional bootstrapping

We compare the scheme [24] based on the multivariate function, and the computational complexity is shown in Tab. 1. Compared to the TFBS scheme [24], the results indicate that our method can reduce the number from exponential growth to linear growth.

– **Implementations**: We implement our proposed method in OpenFHE [3] library. Typically, for a 12-bit-to-12-bit LUT, the LFBS algorithm outperforms the scheme [24] by a factor of **180**.
– **Key Sizes**: Our approach reduces the key size of scheme [24] from the GB level to the MB level. For $B = 4$, and $B = 16$, we achieve reductions of **94**% and **97**%, respectively.

Method	Blind Rotation	External Product	Key Switching	Gadget Product
Scheme [24]	$1+\frac{B^{l-1}-1}{B-1}$	-	$\mathcal{O}(B^{l-1}NB)$	$\mathcal{O}(nB^{l-1}+B^{l-1}NB)$
Our	l	$\mathcal{O}(B^{l-1})$	$\mathcal{O}(B^{l-2}\log N)$	$\mathcal{O}(nl+B^{l-1})$

Table 1. We show the computational cost for the muti-input LUT: $f(m_0,...,m_{l-1})$, for $m_i \in [0, B-1]$, where N is the degree of the polynomial, n is the dimension of the LWE, and $\log N < B < N$. Blind rotation involves $2n$ gadget products and the RLWE key switching needs one gadget product.

Scheme Switching. We propose a novel framework for switching between the BFV and our LFBS algorithm, allowing for large precision computations, including both polynomial and non-polynomial operations. Our approach utilizes homomorphic digit extraction (HDE), which was originally used in the bootstrapping processes of BGV and BFV, to separate the plaintext space with SIMD operations for the first time. Compared to the extraction method [29] that employs FBS, our technique significantly reduces the number of homomorphic multiplications from $\mathcal{O}(nlN)$ to $\mathcal{O}(l^2B)$.

1.2 Technical overview

Ciphertext Conversion. In this conversion variant, we aim to convert the LWE(m) into RGSW$(X^{\mathsf{Encode}(m)})$. Specifically, for the gadget vector $\mathbf{g} = (B_0, ..., B_{d-1})$, the ciphertext RGSW$(X^{\mathsf{Encode}(m)})$ consists of $2d$ RLWE ciphertexts: $B_i \cdot X^{\mathsf{Encode}(m)}$ and $B_i \boldsymbol{s} \cdot X^{\mathsf{Encode}(m)}$. The native approach involves using d FBS to obtain RLWE$(B_i \cdot X^{\mathsf{Encode}(m)})$. Subsequently, the secret key $\boldsymbol{s}$ can be embedded in these ciphertexts through RLWE key switching. We note that the multi-valued bootstrapping method proposed in scheme [18] can generate all RLWE ciphertexts regarding $B_i \cdot X^{\mathsf{Encode}(m)}$ in the original circuit bootstrapping process. However, this technique does not apply to the new RGSW ciphertext format with cyclic encoding.

In this work, we design a new multi-valued bootstrapping method to obtain the gadget RLWE ciphertext with the cyclic group encoding using only one blind rotation. In detail, we propose a new encoding during blind rotation that pads the gadget vector into RLWE as follows

$$\mathsf{RLWE}(B_0 X^{\mathsf{Encode}(m)} + \cdots + B_{d-1} X^{\mathsf{Encode}(m)+d-1}).$$

Furthermore, to separate the terms for B_i from the RLWE, we perform special modulus switching on the input LWE(m) in advance, ensuring that the terms $B_0, ..., B_{d-1}$ reside within the subring and correspond to different residue classes. In other words, for a polynomial of degree N, we divide these terms into d subrings. Then, the ciphertext of B_0 can be obtained by evaluating the trace from K_N to $K_{N/d}$, as shown in the scheme [12,36]. Similarly, we can rotate the

B_i to the position of B_0 and repeat the HomoTrace operation to obtain the RLWE ciphertext of $B_i X^{\mathsf{Encode}(m)}$. Finally, we only need to perform d RLWE key switching operations to deal with the secret key, which in turn completes the process.

Leveled Functional Bootstrapping. Tree-based bootstrapping [24] involves two core operations: blind rotation and base-aware key switching. For a muti-input LUT: $B^l \to B^{l'}$, TFBS first splits and encodes it into $l' \cdot B^{l-1}$ test polynomials. It then employs blind rotation to perform a multiplication between the input LWE and the test polynomials for the computation of the sub-LUTs. Our main observation is that the ciphertext conversion can convert the (R)LWE into RGSW ciphertext with the cyclic group encoding, and multiplication is interpreted as an external product operation. Thus, we can use the external product to reconstruct the tree-based LUT with LHE mode. The ciphertext conversion is independent of the tree-based LUT construction, which means that the result of the ciphertext conversion procedure can be reused by all sub-LUTs. Thus, our scheme only involves l blind rotations in total.

RLWE Packing. Furthermore, for each level of the tree, TFBS uses sample extraction to obtain the desired results and packs them into ciphertexts corresponding to the test polynomials through base-aware key switching. However, base-aware key switching is an extremely laborious operation, which involves $\mathcal{O}(NdB)$ gadget products. We design a new packing algorithm that eliminates the switching step between LWE and RLWE. In detail, we can integrate the constant terms of two RLWEs into the constant and $N/2$ terms by using only one automorphism. Similarly, given B RLWEs, we can pack their constant terms into $i \cdot \frac{N}{B}$ terms of a new ciphertext for $i \in [0, B-1]$ with B automorphisms. Additionally, other redundancies can be removed through trace evaluation (HomTrace), while target items are replicated using rotation operations. This new packing algorithm requires only $\mathcal{O}(\log N)$ gadget products, which is a significant improvement for the packing process.

1.3 Related Work

The circuit bootstrapping technique, as described in [16], is closely related to our proposed conversion of LWE ciphertext to RGSW ciphertext. Specifically, circuit bootstrapping transforms a 1-bit LWE ciphertext into a 1-bit RGSW ciphertext while also refreshing the noise. This process is particularly useful in the leveled evaluation mode of the TFHE scheme. This process consists of two main steps. The first step, functional bootstrapping, aims to refresh the noise in the LWE ciphertext. The second step uses key switching to transform the LWE ciphertext into its corresponding RGSW form, enabling subsequent circuit evaluations.

Later, Wang et al. [36] proposed a more efficient bootstrapping scheme. Their main improvement is the use of RLWE key switching and homomorphic trace techniques, which eliminate the key switching operation from LWE to RLWE that was required in the second step of the original circuit bootstrapping. Additionally, they implemented the algorithm in the OpenFHE [3] library based on the Number Theoretic Transform (NTT) on the FHEW scheme. Recently, Ha et al. [25] corrected an error analysis and extended the ciphertext modulus from a prime modulus to a power-of-two modulus. They implemented their algorithm in the TFHE-rs library [10] and used Fast Fourier Transform (FFT) for polynomial multiplication, incorporating AVX instructions for enhanced performance. Several works [28] have extended functional bootstrapping to the BFV and CKKS schemes. These solutions enable the execution of batch lookup table (LUT) operations; however, their precision is somewhat limited.

Scheme switching allows the conversion of ciphertext to evaluate arithmetic and boolean circuits for FHE schemes. CHIMERA [6] is the first framework capable of performing exact arithmetic operations on the BFV side, approximate arithmetic operations on the CKKS side, and look-up table (LUT) evaluations on the TFHE side. The second framework, PEGASUS [31], supports both CKKS and FHEW schemes and optimizes the underlying computing processes by utilizing Residue Number System (RNS) and Number Theoretic Transform (NTT) techniques. Moreover, PEGASUS demonstrates its practicality by implementing a viable application that runs K-means clustering on thousands of encrypted samples within a few minutes.

2 Preliminary

2.1 Notation

The lower-case regular bold letters indicate vectors, e.g., $\mathbf{a}$, while italic bold letters represent polynomials, such as $\boldsymbol{a}$. For a real number r, we write the floor, ceiling, and round functions as $\lfloor r \rfloor$ $\lceil r \rceil$ $\lfloor r \rceil$, respectively. We denote the infinity norm $||\mathbf{u}||$ for a vector $\mathbf{u}$ and $\mathbb{Z}_q$ the integer ring $\mathbb{Z}/q\mathbb{Z}$ and the scope is $[-q/2, q/2) \cap \mathbb{Z}$, and sometimes $[x]_Q$ is used to denote $x \mod Q$. We use $\leftarrow$ to denote randomly choosing an element from uniform and Gaussian distributions. Let N be a power of 2, we denote the $2N$-th cyclotomic ring by $\mathbb{Z}[X]/(X^N + 1)$, and the quotient ring is $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$ with coefficients in $\mathbb{Z}_Q$. Sometimes, we also use $\mathcal{R}_{N,Q}$ to denote the quotient ring. For a polynomial $\boldsymbol{s}$, we denote $\boldsymbol{\phi}(\boldsymbol{s}) = (s_0, ..., s_{N-1}) \in \mathbb{Z}_q^N$ as the vector of coefficient. Furthermore, we denote σ^2 as the variance of the Gaussian Distribution for $a \in \mathbb{Z}_q$ or $\boldsymbol{a} \in \mathcal{R}$. Finally, we use $*$ to indicate terms that are considered redundant.

2.2 Gadget Decomposition

For a modulus Q and a polynomial $\boldsymbol{a} \in \mathcal{R}_Q$, we define the gadget vector as $\mathbf{g} = \left(B^0, B, \cdots, B^{d-1}\right)$, and the the signed exact decomposition in base B is

$$\mathbf{g}^{-1}(\boldsymbol{a}) = \left([\boldsymbol{a}]_B, \left[\left\lfloor \frac{\boldsymbol{a}}{B} \right\rceil\right]_B, \cdots, \left[\left\lfloor \frac{\boldsymbol{a}}{B^{d-1}} \right\rceil\right]_B\right) \in \mathcal{R}_B^d,$$

where each term belongs to $[-B/2, B/2]$, and the decomposition length satisfies $d = \lceil \log_B Q \rceil$. It is easy to see that $\langle \mathbf{g}^{-1}(\boldsymbol{a}), \mathbf{g} \rangle \equiv \boldsymbol{a} \mod Q$. Furthermore, for $B^d < Q$, we can also define the approximate gadget decomposition with the gadget vector $\mathbf{g}_{\mathsf{AG}} = \left(\left\lceil \frac{Q}{B^d} \right\rfloor, \left\lceil \frac{Q}{B^d} \right\rfloor \cdot B, ..., \left\lceil \frac{Q}{B^d} \right\rfloor \cdot B^d \right)$, which decompose $\boldsymbol{a}$ and outputs $(\boldsymbol{a}_0, ..., \boldsymbol{a}_{d-1})$. The approximate gadget decomposition introduces an approximate error ϵ, i.e., $\langle \mathbf{g}_{\mathsf{AG}}^{-1}(\boldsymbol{a}), \mathbf{g}_{\mathsf{AG}} \rangle \equiv \boldsymbol{a} + \epsilon$, where $|\epsilon| \le \frac{1}{2} \left\lceil \frac{Q}{B^d} \right\rfloor$.

2.3 Hard Problems and Ciphertexts

- **LWE [34].** Given the positive integers n and q, the LWE encryption of the message $m \in \mathbb{Z}$ is a vector $(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, where $b = \mathbf{a} \cdot \mathbf{s} + m + e \mod q$. The vector $\mathbf{a}$ is uniformly sampled from $\mathbb{Z}_q^n$, secret key $\mathbf{s}$ is sampled from a key distribution χ, error e is sampled from the Gaussian Distribution χ'.
- **RLWE [32].** RLWE is a ring version of LWE on $\mathcal{R}_Q$. The RLWE encryption of the message $\boldsymbol{m} \in \mathcal{R}_Q$ is a pair $(\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_Q^2$, where $\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{m} + \boldsymbol{e} \mod Q$. The polynomial $\boldsymbol{a}$ is uniformly sampled from $\mathcal{R}_Q$, secret key $\boldsymbol{s}$ is sampled from a key distribution χ, and each coefficient of the error $\boldsymbol{e}$ is sampled from the Gaussian Distribution χ'.
- **RLWE′[33].** Given the gadget vector $\mathbf{g} = (B_0, ..., B_{d-1})$, the gadget RLWE ciphertext is defined by:

$$\text{RLWE}'(\boldsymbol{m}) = (\text{RLWE}(B_0 \cdot \boldsymbol{m}), \cdots, \text{RLWE}(B_{d-1} \cdot \boldsymbol{m})).$$

- **RGSW [20].** The RGSW encryption of the message $\boldsymbol{m}$ is defined by:

$$\text{RGSW}(\boldsymbol{m}) = (\text{RLWE}'(-\boldsymbol{s} \cdot \boldsymbol{m}), \text{RLWE}'(\boldsymbol{m})).$$

This paper adopts the most significant bit (MSB) encoding by default. The encoding and decoding functions are defined as follows $\mathsf{Encode}: \varphi = \left\lfloor \frac{q}{t} \right\rfloor \cdot m + e$, $\mathsf{Decode}: \left\lfloor \frac{t}{q} \cdot \varphi \right\rceil \mod t$. In addition, we use superscripts and subscripts to define the parameters of the ciphertext, such as $\text{LWE}^n_{\mathbf{s},t/q}(m)$, $\text{RLWE}_{\boldsymbol{s},N,t/Q}(\boldsymbol{m})$. In certain contexts, we may omit related parameters for brevity.

Gadget Product. For $\boldsymbol{t} \in \mathcal{R}_Q$, $(\boldsymbol{t}_0, ..., \boldsymbol{t}_{d-1})$ denotes the gadget decomposition of $\boldsymbol{t}$ with respect to the gadget vector $\mathbf{g} = (B_0, ..., B_{d-1})$. The gadget product $\odot$ is defined by:

$$\begin{aligned} \boldsymbol{t} \odot \text{RLWE}'_{\boldsymbol{s},Q}(\boldsymbol{m}) &:= \sum_{i=0}^{d-1} \boldsymbol{t}_i \cdot \text{RLWE}_{\boldsymbol{s},Q}(B_i \cdot \boldsymbol{m}) \\ &= \text{RLWE}_{\boldsymbol{s},Q}\left(\sum_{i=0}^{d-1} B_i \cdot \boldsymbol{t}_i \cdot \boldsymbol{m} \right) \\ &= \text{RLWE}_{\boldsymbol{s},Q}(\boldsymbol{t} \cdot \boldsymbol{m}). \end{aligned}$$

We employ the approximate decomposition in the gadget product because its length is shorter than that of the exact decomposition when considering the same level of noise growth. The error variance of the gadget product result is bounded by $\sigma^2_{\mathsf{out}} = \varepsilon_1 \sigma^2_{\mathsf{in}} + \varepsilon_2$, where $\varepsilon_1 = \frac{B^2}{12} dN, \varepsilon_2 = \frac{N}{12} \cdot \left(\left\lceil \frac{Q}{B^d} \right\rceil \right)^2$.

External Product. Given ciphertexts $\mathrm{RLWE}_{s,Q}(\boldsymbol{m}_1) = (\boldsymbol{a}, \boldsymbol{b})$ and $\mathrm{RGSW}_{\boldsymbol{s},Q}(\boldsymbol{m}_2)$, where $||\boldsymbol{m}_2||_2^2 \le 1$, the external product $\boxdot$ is defined by:

$$\begin{aligned}\mathrm{RLWE}_{\boldsymbol{s},Q}(\boldsymbol{m}_1) \boxdot \mathrm{RGSW}_{\boldsymbol{s},Q}(\boldsymbol{m}_2) &= \boldsymbol{a} \odot \mathrm{RLWE}'_{\boldsymbol{s},Q}(-\boldsymbol{s} \cdot \boldsymbol{m}_2) + \boldsymbol{b} \odot \mathrm{RLWE}'_{\boldsymbol{s},Q}(\boldsymbol{m}_2) \\ &= \mathrm{RLWE}_{\boldsymbol{s},Q}(-\boldsymbol{a} \cdot \boldsymbol{s} \cdot \boldsymbol{m}_2 + \boldsymbol{b} \cdot \boldsymbol{m}_2) \\ &= \mathrm{RLWE}_{\boldsymbol{s},Q}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \boldsymbol{e}_1 \cdot \boldsymbol{m}_2).\end{aligned}$$

The external product outputs the RLWE encryption of the product of $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ with the error variance bounded by $\sigma^2_{\mathsf{EP}} = \varepsilon_{\mathsf{EP1}} \cdot \sigma^2_{\mathrm{RGSW}} + \varepsilon_{\mathsf{EP2}} + \sigma^2_{\mathrm{RLWE}}$, where $\varepsilon_{\mathsf{EP1}} = \frac{B^2 dN}{6}$, $\varepsilon_{\mathsf{EP2}} = \frac{N}{12} \cdot \left\lceil \frac{Q}{B^d} \right\rceil^2$, σ^2_{RGSW} denotes the error variance of input RGSW ciphertext, and σ^2_{RLWE} is the error variance of the input RLWE ciphertext.

2.4 Ciphertext Switching

2.4.1 LWE Switching

[**LWE Modulus Switching**]. Given an LWE ciphertext $\mathsf{ct} = (\mathbf{a}, b) \in \mathrm{LWE}^n_{\mathbf{s},Q}(m)$ with error variance σ^2_{in}, the modulus switching algorithm computes

$$\mathsf{ModSwitch}_{Q,q}(\mathsf{ct}) = (\left\lfloor \frac{q}{Q} \cdot \mathbf{a} \right\rceil, \left\lfloor \frac{q}{Q} \cdot b \right\rceil),$$

which outputs the LWE ciphertext under modulus q, and its variance satisfies $\sigma^2_{\mathsf{out}} \le (\frac{q}{Q})^2 \cdot \sigma^2_{\mathsf{in}} + \frac{n+2}{24}$ as shown in Appendix A.2.

[**LWE Key Switching**]. Given an LWE ciphertext $\mathsf{ct} = (\mathbf{a}, b) \in \mathsf{LWE}^N_{\mathbf{z},Q}(m)$ with error variance σ^2_{in}, and the switching keys $\mathsf{LK}_{i,j,v} \in \mathsf{LWE}^n_{\mathbf{s},Q}\left(vz_i B^j_{\mathsf{LK}}\right)$ with error variance σ^2_{LK}, where $v \in \{0, \dots, B_{\mathsf{LK}} - 1\}$, $d_{\mathsf{LK}} = \lceil \log_{B_{\mathsf{LK}}} Q \rceil$, for all $0 \le i \le N-1$, $0 \le j \le d_{\mathsf{LK}} - 1$, the LWE key switching algorithm computes

$$\mathsf{LWE.KeySwitch}(\mathsf{ct}) = (\mathbf{0}, b) - \sum_{i,j} \mathsf{LK}_{i,j,a_{i,j}},$$

which outputs a new LWE ciphertext $\mathsf{ct}' \in \mathsf{LWE}^n_{\mathbf{s},Q}(m)$, and its variance satisfies $\sigma^2_{\mathsf{out}} \le \sigma^2_{\mathsf{in}} + N d_{\mathsf{LK}} \cdot \sigma^2_{\mathsf{LK}}$ as shown in Appendix A.4.

2.4.2 RLWE Switching

[**RLWE Key Switching**]. The RLWE key switching can converts the ciphertext $\mathrm{RLWE}_{\boldsymbol{s_1}}(\boldsymbol{m})$ to a new ciphertext $\mathrm{RLWE}_{\boldsymbol{s_2}}(\boldsymbol{m})$. The process need the key switching key $\mathsf{RK} = \mathrm{RLWE}'_{\boldsymbol{s}_2}(\boldsymbol{s_1})$. Given the $\mathrm{RLWE}_{\boldsymbol{s_1}}(\boldsymbol{m}) = (\boldsymbol{a}, \boldsymbol{b})$, it computes

$$\mathsf{RLWE.KeySwitch}(\mathsf{ct}, \mathsf{RK}) = (0, \boldsymbol{b}) - \boldsymbol{a} \odot \mathsf{RK},$$

which outputs the ciphertext $\mathrm{RLWE}_{\boldsymbol{s_2}}(\boldsymbol{m})$. The correctness of the RLWE key switching algorithm can be derived directly from LWE key switching, and error variance σ^2_{RKS} is the same as that of the gadget product.

In addition, the process can embed the secret key $\boldsymbol{s}$ into the message, which we refer to as secret key switching. Given the key switching key $\mathsf{RSK} = \mathrm{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$, and the ciphertext $\mathsf{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathrm{RLWE}_{\boldsymbol{s}}(m)$, we can compute

$$\mathsf{RLWE.SKSwitch}(\mathsf{ct}, \mathsf{RS}) = \boldsymbol{a} \odot \mathsf{RSK} + (\boldsymbol{b}, 0)$$

that outputs ciphertext $\mathrm{RLWE}_{\boldsymbol{s}}(-\boldsymbol{s} \cdot \boldsymbol{m})$ as shown in scheme [36]. Note that the secret key switching introduces an additional error of $\boldsymbol{e} \cdot \boldsymbol{s}$, where the error variance satisfies $\sigma^2_{\mathsf{out}} \leq \frac{N}{2}\sigma^2_{\mathrm{in}} + \sigma^2_{\mathrm{RKS}}$.

[**Automorphism**]. The automorphism operation can change the plaintext $\boldsymbol{m}(X)$ to $\boldsymbol{m}(X^j)$ in RLWE ciphertext. For $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$, there are N automorphisms as $\tau_j : \boldsymbol{m}(X) \to \boldsymbol{m}(X^j)$ for $j \in \mathbb{Z}^*_{2N}$. Given ciphertext $\mathsf{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}^2_Q$, and the RLWE key switching key $\mathsf{ATK}_{\tau_j} \in \mathrm{RLWE}'_{\boldsymbol{s},Q}(\boldsymbol{s}(X^j))$, the automorphism $\mathsf{HomAuto}_{\tau_j}(\mathsf{ct}, \mathsf{ATK}_{\tau_j})$ is divided two steps:

– Let $\tau_j(\mathsf{ct}(X)) = (\boldsymbol{a}(X^j), \boldsymbol{b}(X^j) \in \mathcal{R}^2_Q$.
– Apply the RLWE key switching from the secret key $\boldsymbol{s}(X^j)$ to $\boldsymbol{s}(X)$.

The first step outputs an RLWE encryption of $\boldsymbol{m}(X^j)$ under the secret key $\boldsymbol{s}(X^j)$. In the second step, RLWE key switching is used to switch the secret key from $\boldsymbol{s}(X^j)$ to $\boldsymbol{s}(X)$, where the error variance of the automorphism operation is equal to the RLWE key switching since the first step does not increase noise.

[**HomoTrace**]. For the tower of finite fields $E = K_N \geq K_{N/2} \geq \cdots \geq K_1 = F$, there are $\log N$ special automorphisms as $\psi_j : \boldsymbol{m}(X) \to \boldsymbol{m}(X^{2^j+1})$ for $j \in \{1, \log N\}$, where K_d denotes the $(2d)$-th cyclotomic field for a power-of-two d. The field trace can be expressed as a composition $\mathrm{Tr}_{E/F} = \mathrm{Tr}_{K_2/F} \circ \cdots \circ \mathrm{Tr}_{K_N/K_{N/2}}$ by using these special automorphisms, which can reduce the number of automorphisms from N to $\log N$ times. The homomorphic trace (HomoTrace) evaluation algorithm $\mathrm{Tr}_{E/F}$ is present in Appendix A.5.

2.4.3 LWE-RLWE Switching

[**SampleExtract**]. The sample extraction operation can extract some LWEs from an RLWE ciphertext. Given the RLWE ciphertext $\mathsf{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}^2_Q$ under the secret key $\boldsymbol{s} \in \mathcal{R}_Q$. We can define sample extraction as

$$\mathsf{SampleExtract}(\mathsf{ct}) = (a_0, -a_{N-1}, -a_{N-2}, ..., -a_1, b_0) \in \mathrm{LWE}^N_{\phi(\boldsymbol{s}),Q}(\phi(m)_0),$$

where the process does not increase the noise and the process can be extended to extract other terms of the RLWE ciphertext.

[**LWE-RLWE Keyswitching**]. The LWE-to-RLWE key switching algorithm can pack some LWE ciphertexts into an RLWE ciphertext. This paper focuses on the base-aware key switching used in TFBS, each mapping to a sequence of consecutive coefficients in the RLWE ciphertext.

The key switching process is detailed in Algorithm 10. It is computationally intensive, requiring $O(Nd_{BA}B)$ gadget products. Therefore, in practice, a read-based version can be employed to help reduce computational costs and noise growth. However, this approach still necessitates $\mathcal{O}(NB_{\mathsf{BA}}d_{\mathsf{BA}}B)$ addition operations for RLWE. A more comprehensive overview of this process can be found in Appendix A.6.

2.5 Building Blocks of FHEW and TFHE Schemes

2.5.1 CMux Gate

CMux gate is the fundamental computational unit in FHEW and TFHE schemes, which is constructed by external product. CMux gate takes two RLWE ciphertexts $\mathsf{ct}_0, \mathsf{ct}_1$ and an RGSW ciphertext CT as input, and compute

$$\begin{aligned}\mathsf{CMux}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{CT}) &= \mathsf{ct}_1 + (\mathsf{ct}_2 - \mathsf{ct}_1) \boxdot \mathsf{CT} \\ &= \begin{cases} \mathsf{ct}_1, & \text{if } m = 0 \text{ ;} \\ \mathsf{ct}_2, & \text{else } m = 1, \end{cases}\end{aligned}$$

where the selection bit for message m is represented by the RGSW ciphertext. The correctness of the CMux gate can be obtained directly from the external product, and the noise variance is twice that of the external product. Input some RGSW ciphertext of $(x_0, ..., x_{\ell-1})$, where $x_i \in [0,1]$, the CMux gates tree can be used to perform the LUT $f(x_0, ..., x_{\ell-1})$ as shown in TFHE scheme [17]. In addition, it introduces vertical and horizontal packing techniques to improve the efficiency of LUT evaluation. We show these details in Appendix **??**.

2.5.2 Blind Rotation

Blind rotation is a fundamental component of bootstrapping in FHEW and TFHE. Specifically, given an LWE ciphertext $(\mathbf{a}, b)$ where the secret is $\mathbf{s} = (s_0, ..., s_{n-1})$, the blind rotation key BRK_i encrypts s_i using RGSW ciphertext. The blind rotation operation then produces a new ciphertext $\mathsf{ct} \in \mathsf{RLWE}(X^{-\varphi(m)})$. In this context, $\varphi(m)$ is defined as $b - \sum_{i=0}^{n-1} a_i s_i$. Alg. 1 shows the CGGI method [15] by using the CMux gate, and the noise growth is derived from the external product. Let σ^2_{BR} is the error variance of BRK, the error variance of blind rotation is $\sigma^2_{\mathsf{BR}} = \varepsilon_{\mathsf{BR1}} \cdot \sigma^2_{\mathsf{BRK}} + \varepsilon_{\mathsf{BR2}}$, where $\varepsilon_{\mathsf{BR1}} = \frac{nNB^2_{\mathsf{BR}}d_{\mathsf{BR}}}{3}$, $\varepsilon_{\mathsf{BR2}} = \frac{nN}{12} \cdot \left(\left\lceil \frac{Q}{B^d_{\mathsf{BR}}} \right\rceil\right)^2$.

2.5.3 Functional Bootstrapping

Functional bootstrapping (FBS) can evaluate a LUT in bootstrapping. Given a function $f : \mathbb{Z}_B \to \mathbb{Z}_B$ and ciphertext $\mathsf{ct} = (\mathbf{a}, b) \in \mathrm{LWE}^n_{\mathbf{s},t/q}(m)$, where $t = 2B$, $q = 2N$, and $m \in \mathbb{Z}_B$, we can embed the LUT into a test polynomial denoted as

$$\mathsf{testP}(X) = \sum_{i=0}^{N-1} \frac{Q}{t} \cdot f\left(\left\lfloor \frac{i \cdot t}{q} \right\rceil\right) \cdot X^i.$$

Algorithm 1 Blind Rotation with CMUX gate (BR)

```
Input:
    The LWE sample $\mathsf{ct} = (\mathbf{a}, b) \in \mathrm{LWE}^n_{\mathbf{s},q}(m)$, where $q = 2N$.
    Thr blind rotation key $\mathsf{BRK} : \{\mathrm{RGSW}_{\boldsymbol{s},Q}(s_i)\}$ for $i \in [0, n-1]$.
Output:
    The ciphertext $\mathsf{acc} \in \mathrm{RLWE}_{\boldsymbol{s},Q}(X^{-\varphi(m)})$.
1: Set $\mathsf{acc} = X^{-b}$,
2: for $i = 0$ to $n - 1$ do
3:         $\mathsf{acc} = \mathsf{CMux}(\mathsf{acc}, X^{a_i} \cdot \mathsf{acc}, \mathsf{BRK}_i)$,
4: end for
5: return acc
```

Then, we initialize accumulator to $\mathsf{acc} = \mathsf{testP}(X) \cdot X^{-(b+\theta)}$ for the line 1 of the Alg. 1, where $\theta = \frac{q}{2t}$.

After performing blind rotation, we obtain the expression $\mathrm{RLWE}(\mathsf{testP}(X) \cdot X^{-(\varphi(m)+\theta)})$, where the constant term corresponds to the message $f(m)$. To obtain the LWE ciphertext of $f(m)$, we first extract the sample, followed by LWE key switching and modulus switching to convert the ciphertext parameters. Additionally, functional bootstrapping can be extended to multi-valued functional bootstrapping (MVFBS), which allows for the simultaneous evaluation of multiple unrelated functions, as demonstrated in schemes such as [18] and [11]. We can also set $t = B$, ensuring that the lookup table (LUT) satisfies the negacyclic property. In this context, we assume that the message of the LWE satisfies $m \in \mathbb{Z}_{t/2}$ during the functional bootstrapping process.

3 LWE-to-RGSW Ciphertext Conversion

Circuit bootstrapping (CBS) was originally proposed by Chillotti et al. [16] in the TFHE scheme. It convert the 1-bit LWE(m) into RGSW(m), serving to LHE mode with the CMux gate. In this section, we propose a ciphertext conversion that can convert multi-bit ciphertext LWE(m) into ciphertext $\mathrm{RGSW}(X^{\mathsf{Encode}(m)})$, which achieves functional bootstrapping via the external product. Note that the RGSW ciphertext can serve as an intermediate form for improving schemes such as tree-based bootstrapping [24]. However, this conversion can not apply the original multi-valued bootstrapping method [18].

First, let us analyze the reasons. Given the gadget vector $\mathbf{g} = (B_0, ..., B_{d-1})$, the multi-valued bootstrapping can pad the gadget vector as a subsequence of the test polynomial as $\mathsf{testP}(X) = \sum_{i=0}^{\frac{N}{2^\vartheta}-1} \sum_{j=0}^{2^\vartheta-1} B_i X^{2^\vartheta \cdot i + j}$, where $\vartheta = \lceil \log d \rceil$. Thus, the blind rotation outputs an RLWE ciphertext as

$$\mathrm{RLWE}(B_0 \cdot m + \cdots + B_{d-1} \cdot mX^{d-1} + *X^d + ... + *X^{N-1}),$$

and then the ciphertext $\mathrm{RLWE}(B_i \cdot m)$ can be extracted by HomoTrace or LWE-to-RLWE key switching. However, in our LWE-to-RGSW ciphertext conversion, the message m and the gadget vector are located in the powers and coefficients of the polynomials, respectively, that is

$$\mathrm{RLWE}(B_0 \cdot X^{\mathsf{Encode}(m)}), \cdots, \mathrm{RLWE}(B_{d-1} \cdot X^{\mathsf{Encode}(m)}),$$

where the method [18] can not generate the ciphertext forms.

3.1 Native Construction

The native method involves invoking functional bootstrapping d times without sample extraction, which generates the RLWE ciphertext of $B_i \cdot X^{\mathsf{Encode}(m)}$. In detail, for the LWE ciphertext $(\mathbf{a}, b) \in \mathrm{LWE}^n_{\mathbf{s},q}(m)$, we initialize the accumulator acc as $B_i X^{-(b+\theta)}$. After performing blind rotation, we obtain the RLWE ciphertext of $B_i \cdot X^{-(\varphi(m)+\theta)}$. The secret key $\boldsymbol{s}$ can then be embedded to create the ciphertext $\mathrm{RLWE}(-B_i \boldsymbol{s} \cdot X^{-(\varphi(m)+\theta)})$ through the RLWE secret key switching process. These $2d$ ciphertext consists of the RGSW ciphertext as

$$(\mathrm{RLWE}'_{\boldsymbol{s},Q}(-\boldsymbol{s} \cdot X^{(\varphi(m)+\theta}), \mathrm{RLWE}'_{\boldsymbol{s},Q}(X^{-(\varphi(m)+\theta)})).$$

Alg. 2 shows the detailed process, and this method needs d blind rotations and d key switching operations in total.

Algorithm 2 Ciphertext Conversion (CC)

```
Input:
    The gadget vector $\mathbf{g} = (B_0, ..., B_{d-1}) \in \mathbb{Z}_Q^d$.
    The LWE sample $\mathsf{ct} = (\mathbf{a}, b) \in \mathrm{LWE}^n_{\mathbf{s},q}(m)$, where $q = 2N$.
    The blind rotation key $\mathsf{BRK} : \{\mathrm{RGSW}_{Q,N}(s_i)\}$, for $i \in [0, n-1]$.
    The key switching $\mathsf{RSK} = \mathrm{RLWE}'_{\boldsymbol{s}}(\boldsymbol{s}^2)$.
Output:
    RGSW ciphertext $\mathsf{CT} \in \mathsf{RGSW}_{\boldsymbol{s},N,Q}(X^{-(\varphi(m)+\theta)})$.
1: for $i = 0$ to $d-1$ do
2:     Set $\mathsf{acc}_i = B_i \cdot X^{-(b+\theta)}$,
3:     for $j = 0$ to $n-1$ do
4:         $\mathsf{acc}_{i,j} \leftarrow \mathsf{CMux}(\mathsf{acc}_{i,j}, X^{a_j} \cdot \mathsf{acc}_{i,j}, \mathsf{BRK}_j)$,
5:     end for
6:     $\mathsf{ct}_{i+d} = \mathsf{acc}_{i,n-1}$,
7:     $\mathsf{ct}_i = \mathsf{RSK.SKSwitch}(\mathsf{acc}_{i,n-1}, \mathsf{RSK})$,
8: end for
9: return $\mathsf{CT} = \{\mathsf{ct}_0, ..., \mathsf{ct}_{2d-1}\}$.
```

3.2 Multi-valued Method for Ciphertext Conversion

Considering that blind rotation is the main computational bottleneck, our core goal is to design a new multi-valued method for ciphertext conversion, reducing the number of blind rotations. The key challenge of this multi-valued method is to efficiently pad the gadget vector into the test polynomial and extract the desired terms.

To address the padding problem, we propose a novel encoding method. Instead of using all the coefficients of the test polynomial, we encode both B_i and m into a single block. For instance, let $\mathbf{g} = (B_0, B_1)$. The initial accumulator, acc, is set to $B_0 X^{-(b+\theta)} + B_1 X^{-(b+\theta)+1}$. Through blind rotation, we can obtain $\mathrm{RLWE}(B_0 X^{-(\varphi(m)+\theta)} + B_1 X^{-(\varphi(m)+\theta)+1})$. However, we still cannot determine the position of B_i within the polynomial because the LWE ciphertext is uniformly distributed. To address this, the additional processing for the LWE ciphertext needs to be implemented to ensure that B_0 and B_1 belong to different classes, allowing us to effectively split and extract them. Our approach is as follows.

Algorithm 3 Ciphertext Conversion with HomTrace (CCH)

```
Input:
    The gadget vector $\mathbf{g} = (B_0, ..., B_{d-1}) \in \mathbb{Z}_Q^d$, and let $2^\vartheta = d$.
    The LWE ciphertext $\mathsf{ct} = (\mathbf{a}, b) \in \mathrm{LWE}^n_{\mathbf{s},q}(m)$.
    The blind rotation key $\mathsf{BRK} : \{\mathrm{RGSW}_{\boldsymbol{s},N}(s_i)\}$, for $i \in [0, n-1]$.
    The automorphism keys $\mathsf{ATK}_u = \mathrm{RLWE}'_{\boldsymbol{s},N}(\boldsymbol{s}(X^u))$, where $u \in \mathbb{Z}^*_{2N}$.
    The key switching $\mathsf{RSK} = \mathrm{RLWE}'_{\boldsymbol{s},N}(\boldsymbol{s}^2)$.
Output:
    RGSW ciphertext $\mathsf{CT} \in \mathrm{RGSW}_{\boldsymbol{s},N}(X^{-(\varphi(m)+\theta)})$.
1: for $i = 0$ to $n-1$ do
2:     $\tilde{\mathsf{ct}}_i = \left[\left\lfloor \frac{\mathsf{ct}_i \cdot 2N \cdot 2^{-\vartheta}}{q} \right\rceil \cdot 2^\vartheta\right]_{2N}$,
3: end for
4: $\mathsf{acc} = B_0 \cdot X^{-(\tilde{b}+\theta)} + B_1 \cdot X^{-(\tilde{b}+\theta)+1} + \cdots + B_{d-1} \cdot X^{-(\tilde{b}+\theta)+d-1}$,
5: for $i = 0$ to $n-1$ do
6:     $\mathsf{acc} \leftarrow \mathsf{CMux}(\mathsf{acc}, X^{\tilde{a}_i} \cdot \mathsf{acc}, \mathsf{BRK}_i)$,
7: end for
8: for $i = 0$ to $d-1$ do
9:     $\mathsf{ct}_{i+d} = \mathsf{acc} \cdot X^{-i}$,
10:    $\mathsf{ct}'_{i+d} = \mathsf{Eval}.\mathrm{Tr}_{K_N/K_{N/d}}(\mathsf{ct}'_{i+d}, \mathsf{ATK})$,
11:    $\mathsf{ct}'_i = \mathsf{RLWE.SKSwitch}(\mathsf{ct}_{i+d}, \mathsf{RSK})$,
12: end for
13: return $\mathsf{CT} = \{\mathsf{ct}'_0, ..., \mathsf{ct}'_{2d-1}\}$.
```

Special Modulus Switching. Given the ciphertext $(\mathbf{a}, b) \in \mathrm{LWE}^n_{\mathbf{s},q}(m)$, we can use the special modulus switching to set the ϑ least significant bits (LSB) to zero of ciphertext as follows

$$a'_i = \left[\left\lfloor \frac{a_i \cdot 2N \cdot 2^{-\vartheta}}{q} \right\rceil \cdot 2^\vartheta\right]_{2N},$$

which result in $\varphi(m) = \Delta \cdot m + 2^\vartheta e \mod 2N$, where $\Delta = 2N/t$ is a multiple of 2^ϑ. Note that the 2^ϑ empty positions can encode the gadget vector. For the case

$d = 2$, and $\vartheta = 1$, we have

$$B_0 X^{-(b+\theta)} + B_1 X^{-(b+\theta)+1} \xrightarrow{\mathsf{BR}} \mathsf{ct} = \mathrm{RLWE}(B_0 X^{-(\varphi(m)+\theta)} + B_1 X^{-(\varphi(m)+\theta+1)}).$$

The term B_0, which is associated with $\varphi(m)$, belongs to one of the even coefficients. Conversely, B_1, which is linked to $\varphi(m) + 1$, is categorized as one of the odd coefficients. Without loss of generality, we may assume that $2^{\vartheta} \geq d$ for a larger length d. Then, the terms $B_0, B_1, \ldots, B_{d-1}$ correspond to the subring and coset of a polynomial with different residue classes.

After that, we can use the automorphism operation to remove and split the terms $B_0, ..., B_{d-1}$. For the simple case, we can evaluate the HomoTrace from the finite field K_N to $K_{N/2}$ as

$$\mathsf{ct} + \mathsf{HomoAuto}_{\psi_{\log N}}(\mathsf{ct})$$

to obtain the ciphertext $\mathrm{RLWE}(B_0 X^{\varphi(m)+\theta})$. With the same way, we can get the ciphertext $\mathrm{RLWE}(B_1 X^{\varphi(m)+\theta})$ as

$$\mathsf{ct} \cdot X^{-1} + \mathsf{HomoAuto}_{\psi_{\log N}}(\mathsf{ct} \cdot X^{-1}).$$

The case can extend to an arbitrary length for the gadget vector by evaluating the HomoTrace from K_N to $K_{N/d}$. The detailed process is shown in Alg. 3, and the correctness and error analyses are shown in Appendix B.

3.3 Analysis

- **Error growth.** Noise growth of ciphertext conversion is generated by modulus switching, blind rotation, HomTrace, and RLWE key switching, the specific noise analyses are detailed in Appendix B.
- **Key size.** The ciphertext conversion key can be divided into blind rotation key BRK, RLWE key switching key RSK, and automorphism key ATK.
  Alg. 2: $(2nd_{\mathsf{BR}} + d_{\mathsf{RK}}) \cdot$ RLWEs
  Alg. 3: $(2nd_{\mathsf{BR}} + d_{\mathsf{RK}} + \log N \cdot d_{\mathsf{AK}}) \cdot$ RLWEs
- **Computational cost.** We analyze the computational cost of ciphertext conversion in Tab. 2 with the blind rotations, key switching, and gadget products.

Method	Blind Rotation	RLWE Key Switching	Gadget Product
Alg. 2	d	d	$d(2n+1)$
Alg. 3	1	$d \cdot (\log d + 1)$	$2n + d \cdot (\log d + 1)$

Table 2. Comparison of computational complexity for ciphertext conversion.

4 Leveled Functional Bootstrapping

Functional bootstrapping (FBS) is a significant technique for the lookup table (LUT) evaluation in the FHEW and TFHE schemes. However, it can only support single-input LUT, and the precision of the LUT is limited for the regular parameters. Guimarães et al. [24] proposed the tree-based FBS technique to support multiple inputs (large precision) LUT evaluation. In this section, we propose a new leveled functional bootstrapping scheme that significantly reduces the computational efficiency and key size of tree-based FBS. Finally, we provide an extensive analysis and comparison.

4.1 Tree-based Functional Bootstrapping

Given l ciphertexts $\mathsf{ct}_k = \mathrm{LWE}^n_{\mathbf{s},q}(m_k)$ for $k \in [0, l-1]$. Tree-based functional bootstrapping leverages multiple functional bootstrapping operations, each associated with ciphertext blocks, to construct a tree structure capable of evaluating muti-input look-up tables. Specifically, for an arbitrary LUT $f : [0, B-1]^l \to [0, B-1]^{l'}$, TFBS algorithm first split it into l' separate functions as

$$\begin{aligned} [0, B-1]^l &\to \quad [0, B-1] \\ (m_0, ..., m_{l-1}) &\to f_t(m_0, ..., m_{l-1})' \end{aligned}$$

and then the function f_t can be built upon a tree-like architecture with l levels. In detail, TFBS encodes the each function f_t into B^{l-1} RLWE ciphertexts, and each ciphertext encrypts a polynomial $\mathsf{testP}_{t,i}$ as

$$\mathsf{testP}_{t,i}(X) = \sum_{j=0}^{B-1} \sum_{k=0}^{\frac{N}{B}-1} f_t(j, m_1, \cdots, m_{l-1}) \cdot X^{j\cdot\frac{N}{B}+k},$$

where $i = m_1 B^{l-2} + \cdots + m_{l-1}$.

The ciphertext $\mathsf{ct}_k = \mathrm{LWE}^n_{\mathbf{s},q}(m_k)$ is utilized to select the corresponding branch, specifically $f_t(\cdots, m_k, \cdots)$ within the tree structure. For each subfunction f_t, the TFBS algorithm first applies blind rotation to ct_0 and $\mathrm{RLWE}(\mathsf{testP}_{t,i})$, where $\mathrm{RLWE}(\mathsf{testP}_{t,i})$ represents the test polynomial for all $i \in [0, B^{l-1}-1]$. After that, one can run the sample extraction on all rotated results, which generates B^{l-1} LWEs encrypting $f_t(m_0, \cdots)$. Then, base-aware key switching (BAKS) can be used to pack the B^{l-1} LWEs into B^{l-2} RLWEs. The evaluation for the function f_t ends until ct_{l-1} is used.

The sub-function f_t evaluation process in TFBS needs $\sum_{i=0}^{l-1} B^i = \frac{B^l-1}{B-1}$ blind rotations, $\sum_{i=0}^{l-2} B^i = \frac{B^{l-1}-1}{B-1}$ base-aware key switching, and one LWE key switching operation, which goes back to the initial parameters. Consequently, for a LUT $f : [0, B-1]^l \to [0, B-1]^{l'}$, the TFBS algorithm necessitates $l' \cdot \frac{B^l-1}{B-1}$ blind rotations and $l' \cdot (\frac{B^{l-1}-1}{B-1} + 1)$ key switching operations. The detailed algorithm is outlined in Algorithm 4. Furthermore, one can accelerate the TFBS scheme by encoding the LUTs in plaintext polynomials rather than RLWE ciphertexts.

Algorithm 4 Tree-based Functional Bootstrapping (TFBS)

```
Input:
    $l$ LWE ciphertexts $\mathsf{ct}_k = \mathrm{LWE}^n_{\mathbf{s},q}(m_k)$, for $k \in [0, l-1]$.
    $l' \cdot B^{l-1}$ polynomials $\mathrm{testP}_{t,i}$ that encode the LUT $f : [0, B-1]^l \to [0, B-1]^{l'}$,
    where $t \in [0, l'-1], i \in [0, B^{l-1}-1]$.
    The blind rotation key $\mathsf{BRK} : \{\mathrm{RGSW}_{\boldsymbol{s},Q}(s_i)\}$, for $i \in [0, n-1]$ .
    The base-aware key switching key $\mathsf{BAK}_{i,b,v} \in \mathrm{RLWE}'_{\boldsymbol{s},Q}(v\phi(\boldsymbol{s})_i \cdot \sum_{q=bN/B}^{(b+1)N/B-1} X^q)$
    for $i \in [0, N-1], v \in [0, B_{\mathsf{BA}}-1]$, and $b \in [0, B-1]$ in Sec. 2.4.3.
    The LWE key switching key $\mathsf{LK}_{i,j,v} \in \mathrm{LWE}^n_{\mathbf{s},Q_{\mathsf{KS}}}\left(v\phi(\boldsymbol{s})_i B^j_{\mathsf{LK}}\right)$ in Sec. 2.4.1.
Output:
    $l'$ LWE ciphertexts corresponding to $f(m_0, ..., m_{l-1})$.
1:  for $t = 0$ to $l'-1$ do
2:      for $k = 0$ to $l-1$ do
3:          for $i = 0$ to $B^{l-k-1}-1$ do
4:              $\mathsf{ct}_{t,k,i} = \mathsf{BlindRotation}(\mathsf{testP}_{t,i}, \mathsf{ct}_k, \mathsf{BRK})$,
5:              $\mathsf{ct}_{t,k,i} = \mathsf{SampleExtract}(\mathsf{ct}_{k,i})$,
6:          end for
7:          for $i = 1$ to $B^{l-k-2}$ do
8:              $\mathsf{testP}_{t,i-1} = \mathsf{BA.KeySwitch}(\mathsf{ct}_{t,k,(i-1)\times B}, ..., \mathsf{ct}_{t,k,i\times B-1}, \mathsf{BAK})$,
9:          end for
10:     end for
11:     $\mathsf{ct}_t = \mathsf{ModSwitch}_{Q,Q_{\mathsf{KS}}}(\mathsf{ct}_{t,l-1})$,
12:     $\bar{\mathsf{ct}}_t = \mathsf{LWE.KeySwitch}(\mathsf{ct}_t, \mathsf{LK})$,
13: end for
14: return $(\bar{\mathsf{ct}}_0, ..., \bar{\mathsf{ct}}_{l'-1})$.
```

This process can use the multi-valued bootstrapping method [11] with only one blind rotation instead of B^{l-1} at the first level of the tree. Then, the number of blind rotation is $l'(1 + \frac{B^{l-1}-1}{B-1})$. Besides computational efficiency, the key size of base-aware key switching is 4.3 GB in scheme [24], which is nearly 250 times larger than that of functional bootstrapping.

4.2 Leveled Functional Bootstrapping

Recall that in function bootstrapping, the LUT is encoded within the test polynomial, which is represented as the RLWE ciphertext Essentially, FBS needs to perform a homomorphic multiplication between the ciphertexts of testP_i and $X^{\mathsf{Encode(m)}}$. Therefore, in level k of the TFBS process, it employs B^{l-k-1} blind rotations to combine them in level k of the tree.

We redesign the coupling of the test polynomial and LWE ciphertext. We employ the proposed ciphertext conversion technique to convert the LWE ciphertext into RGSW ciphertext, then the multiplication evolves into the efficient external product operation. Consequently, within each layer of the tree, the external product can be employed to execute functional bootstrapping. Furthermore, it is important to highlight that the conversion of ciphertext is not contingent upon its encoding within the test polynomial LUT, which means that our methodology is capable of facilitating multi-valued functional bootstrapping between all sub-LUTs, associated with $\mathsf{testP}_{t,i}$, and RGSW ciphertext of $X^{\mathsf{Encode}(\mathsf{m}_k)}$. Therefore, our method only needs l blind rotations for ciphertext conversion and can perform LUT evaluation with LHE mode.

Optimization for RLWEs Packing. It is worth noting that the LFBS also needs some RLWE packing operations to consolidate B RLWEs into a single RLWE corresponding to the encoded form of the test polynomial at the next level of the tree. However, the BAKS algorithm used in the TFBS scheme is notably inefficient. We propose a new RLWEs packing algorithm to address this issue, which improves efficiency while reducing key sizes. Our core idea is to use special automorphisms to perform the packing step. In detail, given B RLWE ciphertexts $\mathrm{RLWE}_{\boldsymbol{s},N,Q}(\boldsymbol{m}_j) = (\boldsymbol{a}_j, \boldsymbol{b}_j)$, we need to pack $m_{0,0}, ..., m_{B-1,0}$ into a new RLWE ciphertext that encrypts $\tilde{\boldsymbol{m}} = \sum_{j=0}^{B-1} m_{j,0} \cdot X^{j\cdot\frac{N}{B}} \cdot \sum_{k=0}^{\frac{N}{B}-1} X^k$, where $m_{j,0}$ is the constant term for $\boldsymbol{m}_j$. A straightforward method can be summarized as follows:

1. Apply the $\mathsf{HomoTrace}$ to each $\mathrm{RLWE}(\boldsymbol{m}_j)$ as shown in Alg. 9.
2. For each $\mathrm{RLWE}(\boldsymbol{m}_j)$, multiply the $X^{j\cdot\frac{N}{B}}$, and sum the results to obtain a new RLWE.
3. Multiply the polynomial $\mathsf{temp}(X) = 1 + X + \cdots + X^{\frac{N}{B}-1}$ with the RLWE.

Since the HomoTrace (Tr_{K_N/K_1}) involves $\log N$ RLWE key switchings, the total computational cost amounts to $B \cdot \log N$ RLWE key switchings. We introduce the PRCA algorithm, which can further reduce the number of key switchings. We observe that when using the automorphism $\psi_1 : X \to X^{2^1+1}$ as $\mathsf{HomoAuto}_{\psi_1}(\mathsf{ct}_0) + \mathsf{ct}_0$, the $N/2$ term of $\boldsymbol{m}_0$ is set to zero, while the constant term is doubled. Subsequently, we can perform the same operation on the ciphertext ct_1, and sum the results as

$$\mathsf{ct}_0 + \mathsf{HomoAuto}_{\psi_1}(\mathsf{ct}_0) + X^{\frac{N}{2}}(\mathsf{ct}_1 + \mathsf{HomoAuto}_{\psi_1}(\mathsf{ct}_1)), \tag{1}$$

where $m_{0,0}$ and $m_{1,0}$ are embedded into constant and $N/2$ terms of the ciphertext, respectively. After that, Eq. 1 can be simplified to

$$(\mathsf{ct}_0 + X^{\frac{N}{2}} \cdot \mathsf{ct}_1) + \mathsf{HomoAuto}_{\psi_1}(\mathsf{ct}_0 - X^{\frac{N}{2}}\mathsf{ct}_1). \tag{2}$$

It is easy to verify that these two formulas are equivalent, thus the number of automorphisms can be reduced to one. More generally, the automorphism $\psi_i : X \to X^{2^i+1}$ can be used to clear all even k in the terms of $\frac{N}{2^i} \cdot k$ by computing $\mathsf{ct}_j + \mathsf{HomoAuto}_{\psi_i}(\mathsf{ct}_j)$. Then, we can utilize the automorphism $\psi_1, ..., \psi_{\log B}$ to integrate the ciphertexts $\mathsf{ct}_0, ..., \mathsf{ct}_{B-1}$ through Eq. 2 with the tree-based structure. This process outputs encryption of $\sum_{j=0}^{B-1} m_{j,0} \cdot X^{j\cdot\frac{N}{B}} \cdot \sum_{k=0}^{\frac{N}{B}-1} *X^k$. Finally, we can evaluate $\mathrm{Tr}_{K_N/K_{N/B}}$ to clear the redundancies and multiply the polynomial $\mathsf{temp}(X)$ to duplicate the target terms. The PRCA process needs

$$\frac{B}{2} + \frac{B}{4} + \cdots + 1 + \log(N/B) = B - 1 + \log(N/B)$$

RLWE key switchings. Alg. 5 presents the detailed PRCA process, the correctness, and noise growth as shown in Appendix A.5.

Algorithm 5 Packing RLWE Ciphertexts via Automorphism (PRCA)

```
Input:
    $B$ RLWE samples $\mathsf{ct}_j \in \mathsf{RLWE}_{\boldsymbol{s},Q}(\boldsymbol{m}_i)$, $j \in [0, B-1]$ .
    The automorphism keys $\mathsf{ATK}_u = \mathsf{RLWE}'_{\boldsymbol{s},Q}(\boldsymbol{s}(X^u))$, where $u \in \mathbb{Z}_{2N}^*$.
    The polynomial $\mathsf{temp}(X) = 1 + X + \cdots + X^{\frac{N}{B}-1}$.
Output:
    An RLWE sample $\tilde{\mathsf{ct}} \in \mathsf{RLWE}_{\boldsymbol{s},Q}(\tilde{\boldsymbol{m}})$).
1: for $j = 0$ to $B-1$ do
2:     $\mathsf{ct}_j = B^{-1} \cdot \mathsf{ct}_j$,
3: for $i = 1$ to $\log B$ do
4:     for $j = 0$ to $B/2^i - 1$ do
5:         $\mathsf{ct}_j = (\mathsf{ct}_{2j} + X^{\frac{N}{2^i}} \cdot \mathsf{ct}_{2j+1}) + \mathsf{HomoAuto}_{\psi_i}(\mathsf{ct}_{2j} - X^{\frac{N}{2^i}} \mathsf{ct}_{2j+1}, \mathsf{ATK})$,
6:     end for
7: end for
8: $\mathsf{ct}' = \mathsf{Eval.}\,\mathrm{Tr}_{K_N/K_{N/B}}(\mathsf{ct}')$
9: $\tilde{\mathsf{ct}} = \mathsf{ct}' \cdot \mathsf{temp}(X)$;
10: return $\tilde{\mathsf{ct}}$.
```

Based on the PRCA algorithm, we can use automorphisms to substitute the sample extraction and base-aware key switching steps in the TFBS and LFBS schemes. We present the complete LFBS scheme in Alg. 6, and the new workflow can be summarized as follows. Input some ciphertexts $\mathsf{ct}_k = \mathrm{LWE}^n_{\mathbf{s},q}(m_k)$, we first apply the ciphertext conversion to obtain $\mathrm{RGSW}(X^{\mathsf{Encode}(\mathsf{m}_k)})$, for $k \in [0, l-1]$. In the first level of each subfunction f_t, the external product can be used to perform multi-valued functional bootstrapping between the ciphertexts $\mathrm{RGSW}(X^{\mathsf{Encode}(\mathsf{m}_0)})$ and all $\mathrm{RLWE}(P_{t,i})$ for $i \in [0, B^{l-1}-1]$, resulting in B^{l-2} RLWE ciphertexts. Then, we use the PRCA algorithm to pack these RLWEs into some new RLWEs, which are used in the next level of the tree. The evaluation for the function f_t ends until $\mathrm{RGSW}(X^{\mathsf{Encode}(\mathsf{m}_{l-1})})$ is used. The correctness can be derived directly from the ciphertext conversion and the PRCA algorithms.

Horizontal Packing. Our leveled functional bootstrapping can support horizontal packing, which can pack the multiple LUTs f_t into a test polynomial using the multi-valued bootstrapping technique. Specifically, in ciphertext conversion, we use the modulus switching on input LWE ciphertext, such that its phase is $\varphi(m) = \Delta m + 2^{\vartheta} e \mod 2N$, and then 2^{ϑ} empty positions can be further utilized to encode multiple LUTs in the test polynomial. In detail, in the TFBS and LFBS schemes, we can first decompose the large LUT into l' sub-LUTs and encode them in B^{l-1} polynomials testP_i, where $l' \leq 2^{\vartheta}$ in our setting. Assuming

Algorithm 6 Leveled Functional Bootstrapping (LFBS)

```
Input:
    $l$ LWE ciphertexts $\mathsf{ct}_k = \mathsf{LWE}^n_{\mathbf{s},q}(m_k)$, for $k \in [0, l-1]$.
    $l' \cdot B^{l-1}$ polynomials $\mathrm{testP}_{t,i}$ that encode the LUT $f : [0, B-1]^l \to [0, B-1]^{l'}$,
    where $t \in [1, l'], i \in [0, B^l - 1]$ .
    The ciphertext conversion key $\mathsf{CCK}$ including $\mathsf{BRK}$, $\mathsf{RSK}$.
    The automorphism keys $\mathsf{ATK}_u = \mathsf{RLWE}'_{\boldsymbol{s},Q}(\boldsymbol{s}(X^u))$, where $u \in \mathbb{Z}^*_{2N}$.
    The LWE key switching key $\mathsf{LK}_{i,j,v} \in \mathsf{LWE}^n_{\mathbf{s},Q_{\mathsf{KS}}}\left(v\phi(\boldsymbol{s})_i B^j_{\mathsf{LK}}\right)$ in Sec. 2.4.1.
Output:
    $l'$ LWE ciphertexts corresponding to $f(m_0, ..., m_{l-1})$.
1:  for $k = 0$ to $l - 1$ do
2:      $\mathsf{CT}_k = \mathsf{CCH}(\mathsf{ct}_k, \mathsf{CCK})$,
3:  end for
4:  for $t = 0$ to $l' - 1$ do
5:      for $k = 0$ to $l - 1$ do
6:          for $i = 0$ to $B^{l-k-1} - 1$ do
7:              $\mathsf{ct}_{t,k,i} = \mathsf{testP}_{t,i} \boxdot \mathsf{CT}_k$,
8:          end for
9:          for $i = 1$ to $B^{l-k-2}$ do
10:             $\mathsf{testP}_{t,i-1} = \mathsf{PRCA}(\mathsf{ct}_{t,k,(i-1)\times B}, ..., \mathsf{ct}_{t,k,i\times B-1}, \mathsf{ATK})$,
11:         end for
12:     end for
13:     $\mathsf{ct}_t = \mathsf{ModSwitch}_{Q,Q_{\mathsf{KS}}}(\mathsf{ct}_{t,l-1})$,
14:     $\bar{\mathsf{ct}}_t = \mathsf{LWE.KeySwitch}(\mathsf{ct}_t, \mathsf{LK})$,
15: end for
16: return $(\bar{\mathsf{ct}}_0, ..., \bar{\mathsf{ct}}_{l'-1})$.
```

$l' = 2^\vartheta$, we can set

$$\mathsf{testP}_i(X) = \sum_{j=0}^{B-1} \sum_{k=0}^{\frac{N/B}{2^\vartheta}-1} \sum_{t=0}^{2^\vartheta-1} f_t(j, m_1, \cdots, m_{l-1}) \cdot X^{j \cdot \frac{N}{B} + 2^\vartheta k + t},$$

where $i = m_1 \cdot B^{l-2} + \cdots + m_{l-1}$, and it enables us to reduce the external products from $\mathcal{O}(l' B^{l-1})$ to $\mathcal{O}(B^{l-1})$. However, this multi-valued encoding approach cannot reduce the number of PRCA algorithm since we need to pack $m_{0,t}, ..., m_{B-1,t}$ of ciphertexts, $t \in [0, l'-1]$, into a RLWE ciphertext that encrypts

$$\tilde{\boldsymbol{m}} = \sum_{j=0}^{B-1} X^{j \cdot \frac{N}{B}} \cdot \sum_{k=0}^{\frac{N/B}{2^\vartheta}-1} X^{2^\vartheta k} \sum_{t=0}^{2^\vartheta-1} m_{j,t} X^t.$$

It can be performed by using l' PRCA and sum operations.

Remark. For handling large precision message $m = m_0 + m_1 B + \cdots m_{l-1} B^{l-1}$, we first can utilize homomorphic digit decomposition technique [30] to obtain the LWE ciphertexts, LWE(m_0), $\cdots$,LWE(m_{l-1}). Subsequently, the aforementioned multi-input functional bootstrapping algorithms, including TFBS (Alg. 4) and LFBS (Alg.6), can be used to perform the large precision LUT.

Algorithms	Blind Rotation	RLWE Key Switching	Gadget Product
TFBS 4	$l'(1+\frac{B^{l-1}-1}{B-1})$	$\mathcal{O}(NBl'B^{l-2})$	$\mathcal{O}(nl'B^{l-1}+NBl'B^{l-2})$
ITFBS 5	$l'(1+\frac{B^{l-1}-1}{B-1})$	$\mathcal{O}(\log Nl'B^{l-2})$	$\mathcal{O}(nl'B^{l}+\log Nl'B^{l-2})$
LFBS 6	l	$\mathcal{O}(\log Nl'B^{l-2})$	$\mathcal{O}(nl+\log Nl'B^{l-2})$

Table 3. Comparison of computational cost for TFBS and LFBS schemes, where the multi-valued functional bootstrapping [18] is used in TFBS. Note that ITFBS is represented as using PRCA to replace the base-aware key switching in the TFBS scheme.

.

4.3 Analysis and Comparison

- **Error growth.** For the TFBS Alg. 4, the noise variances of the blind rotation and key switching are additive. Thus, the noise variance of the tree-based method when applied to l inputs is less than $l \cdot \sigma^2_{\mathsf{BR}} + (l-1)\cdot \sigma^2_{\mathsf{BA}} + \sigma^2_{\mathsf{LK}}$, where the $\sigma^2_{\mathsf{BR}}, \sigma^2_{\mathsf{BA}}, \sigma^2_{\mathsf{LK}}$ is the noise variance of blind rotation, base-aware key switching, LWE key switching, and the details can be found in the corresponding algorithm.
  For LFBS of Alg. 6, the noise growth is a little different, it incorporates new noise contributions from the external product and automorphism operations. The error variance of the proposed algorithm derives from three parts: the ciphertext conversion σ^2_{CC}, the external product $\varepsilon_{\mathsf{EP1}}$ and $\varepsilon_{\mathsf{EP2}}$, and the PRCA step $\varepsilon_{\mathsf{PA}}$ and σ^2_{PA}. From the noise formula calculated from the external product and PRCA steps, the noise variance is less than $l \cdot \varepsilon_{\mathsf{PA}}(\varepsilon_{\mathsf{EP1}} \cdot \sigma^2_{\mathsf{CCH}} + \varepsilon_{\mathsf{EP2}}) + (l-1)\cdot \sigma^2_{\mathsf{PA}} + \sigma^2_{\mathsf{LK}}$.
- **Key size.** The key of the TFBS and LFBS can be divided into blind rotation key BRK, RLWE secret key RSK, base-aware key BAK, automorphism key ATK, and the LWE switching key LK. The total key size can be summarised as
  TFBS Alg. 4: $(2nd_{\mathsf{BR}} + BNd_{\mathsf{BA}}) \cdot \text{RLWEs} + B_{\mathsf{LK}} d_{\mathsf{LK}} N \cdot \text{LWEs}$
  LFBS Alg. 6: $(2nd_{\mathsf{BR}} + d_{\mathsf{RK}} + \log N \cdot d_{\mathsf{AK}}) \cdot \text{RLWEs} + B_{\mathsf{LK}} d_{\mathsf{LK}} N \cdot \text{LWEs}$
- **Computational cost.** Tab. 3 shows the computational cost of three methods for the LUT $f : [0, B-1]^l \to [0, B-1]^{l'}$ evaluation.

5 Parameter and Implementation

This section shows the parameters setting and implementations for the proposed algorithms. Then, we compare similar schemes in terms of computational efficiency and key size.

5.1 Symbols and Parameters

We first recall the notations of all algorithms in as follows.

- λ, Security level;
- t, Plaintext modulus for the LWE sample;
- n, Lattice dimension for the LWE sample;
- q, Ciphertext modulus for the LWE sample;
- l, l', Input and output length of the LUT;
- $N, \bar{N}$, Ring dimension for RLWE/RGSW;
- $Q, \bar{Q}$, Ciphertext modulus for the RLWE/RGSW;
- σ, Standard deviation of Gaussian distribution;
- q_{LK}, Ciphertext modulus used in LWE key switching;
- $B_{\mathsf{LK}}, d_{\mathsf{LK}}$, Gadget base and length for modulus q_{LK} used in the LWE key switching;
- B, d , Gadget base and length for modulus Q used in the external product of TFBS;
- $B_{\mathsf{BR}}, d_{\mathsf{BR}}$, Gadget base and length used in the blind rotation;
- $B_{\mathsf{RK}}, d_{\mathsf{RK}}$, Gadget base and length used in the RLWE secret key switching;
- $B_{\mathsf{BA}}, d_{\mathsf{BA}}$, Gadget base and length used in the base-aware key switching;
- $B_{\mathsf{HT}}, d_{\mathsf{HT}}$, Gadget base and length used in the HomoTrace process;

5.2 Parameters setting for bootstrapping

We show the parameter settings in Tab. 4 and 5 for ciphertext conversion, tree-based functional bootstrapping, and leveled functional bootstrapping, where B and d are the RGSW parameter of the output for ciphertext conversion. To facilitate comparison with the schemes for blind rotation tree, we also present the parameters of blind rotation for scheme [24], i.e., **BR** I and **BR** II. Besides these parameters, we set the $\vartheta = 3$ in the LWE modulus switching.

These parameter sets support two cases for the base $B = 4$ and $B = 16$. The parameters sets **BT_4_1024** and **BT_16_2048** correspond to the TFBS scheme [24], involving the blind rotation, base-aware key switching, and LWE key switching. Our LFBS scheme is associated with the sets **ET_4_2048** and **ET_16_2048**, including ciphertext conversion, HomoTrace, and LWE key switching.

Note that each parameter set has different dimensions and moduli for the ciphertext and secret key. We use the binary secret key in the LWE and RLWE encryptions, setting the initial variances to $\sigma^2 = 3.2$ across all key generation processes. In addition, the RLWE and RGSW ciphertext modulus $\bar{Q}$ and Q are the NTT-friendly numbers with the degree $\bar{N}$ and N, respectively. The key distribution, error variance, and the dimension and modulus of ciphertexts determine the security level of (R)LWE. We use the lattice estimator[6] to evaluate the security, where the security levels of these parameters exceed 128 bits.

[6] https://github.com/malb/lattice-estimator.

Methods	Param. Sets	$\bar{N}$	$\bar{Q}$	N	Q	n	q	B_{BR}	d_{BR}	B_{RK}	d_{RK}	B	d
Scheme [24]	**BR I**	-	-	1024	$\approx 2^{25}$	571	1024	2^4	5	-	-	-	-
	BR II	2048	$\approx 2^{54}$	-	-	648	2048	2^{13}	3	-	-	-	-
Our Alg. 3	**CC I**	2048	$\approx 2^{54}$	-	-	571	2048	2^{17}	2	2^{17}	2	2^6	2
	CC II	2048	$\approx 2^{54}$	-	-	648	2048	2^{10}	4	2^{10}	4	2^9	2

Table 4. Parameters of the ciphertext conversion, and blind rotation.

Methods	Param. Sets	B	Params. for Tab. 4	B_{BA}	d_{BA}	B_{HT}	d_{HT}	q_{LK}	B_{LK}	d_{Lk}
Scheme [24]	**BT_4_1024**	4	**BR I**	2^6	3	-	-	2^{14}	2^7	2
	BT_16_2048	16	**BR II**	2^6	6	-	-	2^{16}	2^8	2
Our Alg. 6	**ET_4_2048**	4	**CBSV I**	-	-	2^{17}	2	2^{14}	2^7	2
	ET_16_2048	16	**CBSV II**	-	-	2^{17}	2	2^{16}	2^8	2

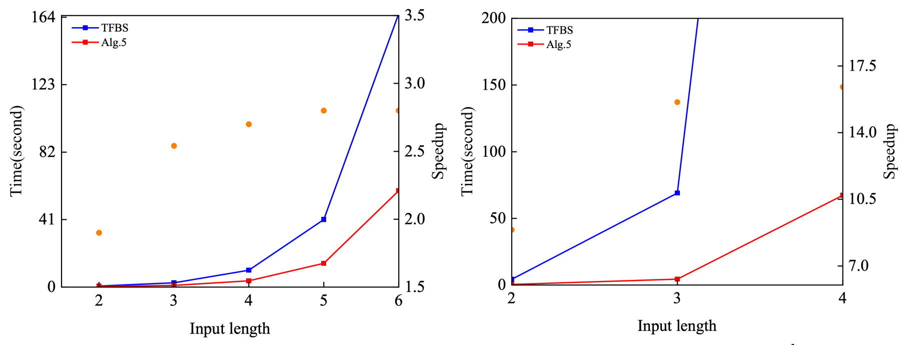
Table 5. Parameters of algorithms for TFBS, LFBS.

Failure Probability. For the LUT $f : B^l \to B^{l'}$ evaluation in the TFBS and LFBS, the output noise is related to depth l as discussed in Sec. 4.3. Based on the noise growth, the probability of decryption failure can be calculated using the formula $1 - \mathsf{erf}\left(\frac{q}{2t\sqrt{2}\sigma_{\mathsf{out}}}\right)$, where erf is the Gaussian function. The plaintext modulus t of LWE ciphertext is set to $2B$ to mitigate the negative effects of functional bootstrapping. We present the decryption failure rates for different lengths in Tab. 6, with all parameters yielding decryption failure rates of less than $2^{-40.8}$. Notably, the parameter set **E_16_2048** can support 32-bit LUT evaluations, making it suitable for many high-precision applications.

Methods	Params. sets	λ	$d = 4$	$d = 6$	$d = 8$
Scheme [24]	**BT_4_1024**	128	$2^{-139.1}$	$2^{-116.1}$	$2^{-86.4}$
	BT_16_2048	128	$2^{-75.7}$	$2^{-75.7}$	$2^{-75.7}$
Our Alg. 6	**ET_4_2048**	128	$2^{-134.5}$	$2^{-58.5}$	$2^{-48.8}$
	ET_16_2048	128	$2^{-78.2}$	$2^{-75.2}$	$2^{-68.9}$

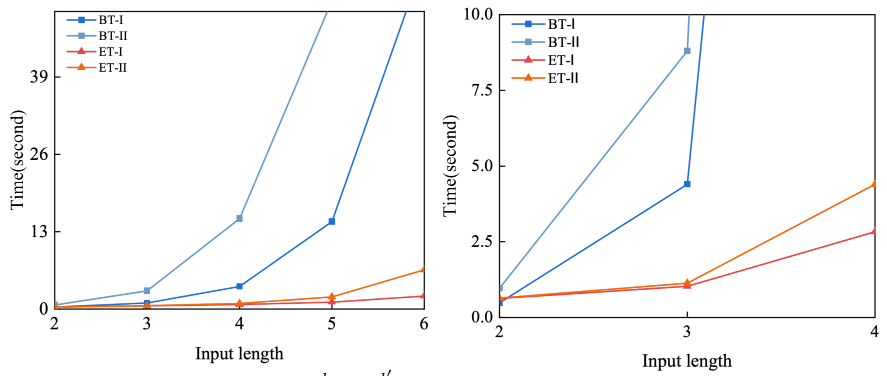
Table 6. Decryption failure rates.

5.3 Experimental Comparison with blind rotating tree



LUT evaluation with $f : 4^l \to 4$ LUT evaluation with $f : 16^l \to 16$

Fig. 2. Performance of TFBS schemes, where the red line indicates the result of replacing BAKS with PRCA, and the vertical axis on the right displays the speedup ratios.



LUT evaluation with $f : 4^l \to 4^{l'}$ LUT evaluation with $f : 16^l \to 16^{l'}$

Fig. 3. Performance of TFBS, LFBS, and schemes[5,36], where BT and ET indicate Blind rotation tree, external product tree. And I and II denote $l' = 1, l' = l$, respectively.

Firstly, we implemented the TFBS, LFBS in OpenFHE library [2]. For a fair comparison with the TFHE-based scheme [24], we use NTT instead of all FFT operations. We stress that TFBS [24] and our approach have the same functionality. Our experiment compute the multivariate function $f(m_0, ..., m_{l-1})$, where $m_i \in [0, B-1]$. All experiments are performed on the Cloud server equipped with an Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50 GHz and 32 GB of RAM, with Gcc-11.3.0 as the compiler.

To analyze the contribution of the two proposed improvements to overall performance, we first replaced the BAKS algorithm used in the TFBS scheme with the PRCA algorithm. Here, we implemented the reading version of BAKS as described in scheme [24]. According to the experimental results presented in Fig. 2, the proposed PRCA method achieves improvements of 1.9 times and 8.9 times over the reading version of the BAKS method for parameters $B = 4$ and $B = 16$, respectively. Furthermore, we demonstrate the run times of TFBS and LFBS by using the PRCA technique in Fig.3. Under these parameter settings, our solutions are more efficient. Specifically, in the case of $B = 16$ with 8-bit to 8-bit LUT evaluation, the running time of our approach is only $\mathbf{0.63}s$, making it $\mathbf{1.5}$ times faster than the TFBS method. For the 12-bit-to-12-bit LUT evaluation, the running time is $\mathbf{1.12}s$, achieving an improvement of $\mathbf{11.7}$ times ($\mathbf{180}\times$ with PRCA algorithm). In the case of 16-bit-to-16-bit LUT evaluation, our method yields improvements of **96** times.

5.4 Comparison for Key Sizes

Methods	Params. sets	BRK	RSK	BAK	HTK	LK	Total
Scheme [24]	**BT_4_1024**	34.8 MB	-	4.68 GB	-	250 MB	4.95 GB
	BT_16_2048	102.5 MB	-	$>$ 10 GB	-	162 MB	$>$ 10.2 GB
Our Alg. 6	**ET_4_2048**	60.2	0.05	-	0.5 MB	250 MB	310.8 MB (**6**%)
	ET_16_2048	136.7	0.1	-	0.5 MB	162 MB	299.3 MB ($<$ **3**%)

Table 7. Key sizes for TFBS and LFBS algorithms, where BRK, RSK BAK, HTK, and LK represent the keys for blind rotation, base-aware, HomoTrace, and LWE key switching, respectively.

We compare the key sizes of the TFBS and LFBS algorithms in Tab. 7. Our proposed Alg. 5 employs RLWE packing with the HomoTrace instead of BAKS, leading to a significant reduction in key sizes. As illustrated in Ta. 7, the key size of our method is only about 300 MB, which is at least 94% smaller than that of the scheme [24].

6 Applications

This section shows two applications for the proposed LFBS algorithm. Firstly, we propose a new large precision switching framework based on the efficient LFBS algorithm. Furthermore, we apply the LFBS algorithm to improve the homomorphic evaluation AES, which mainly focuses on Sbox evaluation.

6.1 Scheme switching with Large Precision

6.1.1 Switching from BFV scheme to LFBS algorithm

Our proposed LFBS algorithm encrypts the digits of an integer into ciphertext blocks, whereas the BFV scheme encrypts the integer vector and performs SIMD mode. Therefore, to convert the BFV scheme to the LFBS scheme, we need to perform sample extraction and homomorphic digit extraction (HDE) to align the message encoding. The existing method uses functional bootstrapping to get each digit of the message as shown in [18,30]. Assume that the plaintext modulus of the BFV is B^l, for a message $m = \sum_{i=0}^{l-1} m_i \cdot B^i$, the method requires lN FBS, resulting in $\mathcal{O}(nNl)$ homomorphic multiplications, where N is the number of encrypted elements in the slot of the BFV scheme.

We propose a batch method that puts the extraction step on the BFV side, reducing the computational cost to $\mathcal{O}(Bl^2)$. Specifically, we utilize the homomorphic digit extraction technique to split messages, which was initially employed in BGV and BFV bootstrapping. Currently, there are two approaches: the first is the lifting polynomial introduced by Halevi and Shoup [26], as detailed in Lemma 1. The second is the digit extraction polynomial developed by Chen and Han [13], as outlined in Lemma 2. Then, we need to analyze the circuit depth and computational complexity of the two methods in our switching process.

Lemma 1 ([26]). *For every prime p and exponent $\ell \geq 1$, there exists a lifting polynomial $F(X) \in \mathbb{Z}[X]$ of degree p such that for all integers $-p/2 < w_0 \leq p/2$ and every $1 \leq i \leq \ell$ it holds that $F(w_0 + p^i w_1) = w_0 \pmod{p^{i+1}}$.*

Using lifting polynomial, we can extract the least significant digit of the input $w \in \mathbb{Z}_{p^\ell}$ through a successive method, where each iteration can remove one additional upper digit. This process requires computing $\ell - 1$ polynomials of degree p, which is equivalent to evaluating a polynomial of degree $p^{\ell-1}$.

Lemma 2 ([13]). *For every prime p and exponent $\ell \geq 1$, there exists a digit extraction polynomial $G_\ell(X) \in \mathbb{Z}[X]$ of degree at most $(\ell-1)(p-1)+1$ such that for all integers $-p/2 < w_0 \leq p/2$, it holds that $G_\ell(w_0 + pw_1) = w_0 \pmod{p^\ell}$.*

By utilizing the digit extraction polynomial $G_\ell(X)$, the least significant digit of the input w can be directly extracted by evaluating a polynomial of degree $(\ell - 1)(p - 1) + 1$.

In tree-based bootstrapping, we can set $B = p^r$, allowing the messages in the BFV scheme to be represented by $\ell = lr$ digits with base p. The goal of switching from the BFV scheme to LFBS is to split the message w into l blocks of length r digits. Typically, it is necessary to extract all ℓ digits and merge them into l blocks. We note that only the lower $(l - 1)r$ digits need to be extracted since the most significant r digits can be obtained by subtracting them from w. Then, we default to the p-adic representation from this point forward. We revisit the lifting polynomial and the digit extraction polynomial in order to obtain a low multiplicative depth of the procedure of the extraction of $(l - 1)r$ digits.

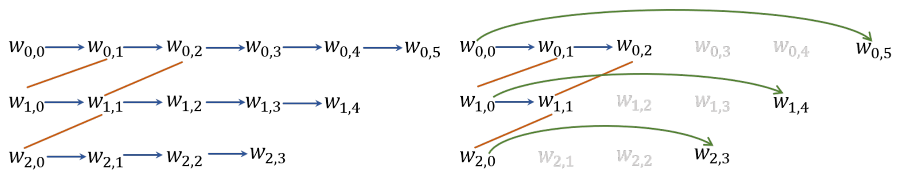
Algorithm 7 Digit Extraction for Halevi/Shoup Method

```
Input:
    Integer $w = \sum_{k=0}^{\ell-1} w_k \cdot p^k$.
Output:
    Integer $w_i$.
1: for $k = 0$ to $\ell - 1$ do
2:     $w_{k,0} = w$
3: end for
4: for $k = 0$ to $i$ do
5:     for $j = 0$ to $\ell - k - 2$ do
6:         $w_{k,j+1} = F(w_{k,j})$
7:         $w_{k+j+1,0} = (w_{k+j+1,0} - w_{k,j+1})/p$
8: return $w_{i,\ell-i-1}$
```

Algorithm 8 Digit Extraction for Chen/Han Method

```
Input:
    Integer $w = \sum_{k=0}^{\ell-1} w_k \cdot p^k$.
Output:
    Integer $w_i$.
1: for $k = 0$ to $i - 1$ do
2:     $w_{k,0} = w$
3: end for
4: for $k = 0$ to $i$ do
5:     $w_{k,\ell-k-1} = G_{\ell-k}(w_{k,0})$
6:     for $j = 0$ to $i - k - 1$ do
7:         $w_{k,j+1} = F(w_{k,j})$
8:         $w_{k+j+1,0} = (w_{k+j+1,0} - w_{k,j+1})/p$
9:     end for
10: end for
11: return $w_{i,\ell-i-1}$
```

In detail, all the digits form a trapezoid in our extraction process as shown in Fig. 4. The first digit of each row is computed by subtracting the digits on the same diagonal and dividing by p. For the last digit of each row, i.e., $w_{i,\ell-1-i}$, it can be computed with either Halevi/Shoup or Chen/Han methods. We can apply the lifting polynomial to this number with the Lemma 1, which requires to sequentially compute $w_{i,0}$, $w_{i,1}$ up to $w_{i,5-i}$. In this method, the degree of polynomial evaluation of the i-th digit is $p^{\ell-1}$ as shown in Alg. 7, which is also the maximum degree for the extraction process. Another method is to directly apply the digit extraction polynomial to the element $w_{i,0}$, which outputs the element $w_{i,5-i}$ with a polynomial of degree $(p-1)(\ell-i-1)+1$ as shown in Lemma 2. Based on the Chen/Han method, the degree of evaluation of the i-th digit is $p^i \cdot ((p-1)(\ell-i-1)+1)$, where p^i is the degree of the lifting polynomial used in the first number in the i row, referring to Alg. 8. Thus, the maximum degree of this procedure is $p^{\ell-r-1} \cdot (r(p-1)+1)$, and we simplify it as $rp^{\ell-r}$.



(a)HDE via Halevi/Shoup method (b)HDE via Chen/Han method

Fig. 4. HDE illustration for $l = 2, r = 3$, and $\ell = 6$. We use $w_{i,j}$ to denote the integer with the base p digit as the i-th digit of w and the next j digits zero, blue arrow to denote lifting polynomial and green arrow to denote digit extraction polynomial.

After that, the second digit in base B is obtained as $w_1 = (((w - w_{0,5})/p - w_{1,4})/p - w_{2,3})/p$. In this step, the division by p can convert the plaintext modulus to B referring to Appendix A.1. Moreover, the first digit in base B is obtained as $w_0 = w_{0,5} + p \cdot w_{1,4} + p^2 \cdot w_{2,3}$, where the ciphertext of digit $w_{i,\ell-1-i}$ can be viewed as an encryption of $p^i \cdot w_{i,\ell-1-i}$ with plaintext modulus p^ℓ. Therefore, the addition is straightforward. To convert the plaintext modulus to B, we need to perform modulus reduction on the ciphertext[7], as shown in Appendix A.1. Finally, this procedure can be easily generalized to any l and r.

	Halevi/Shoup	Chen/Han	FBS[30]
Degree	$p^{\ell-1}$	$rp^{\ell-r}$	-
Depth	$(\ell-1)\log p$	$\log r + (\ell - r)\log p$	-
# homomorphic Mul.	$(\ell-r)\frac{\ell+r}{2} \cdot 2\sqrt{p}$	$(\ell-r)(\sqrt{\frac{\ell+r}{2}} + \frac{\ell-r}{2}) \cdot 2\sqrt{p}$	nlN

Table 8. The degree, multiplicative depth, and the numbers of non-scalar multiplication of HDE with different methods.

We summarize the degree of polynomial, multiplicative depth, and number of homomorphic multiplication in Tab. 8. It has been seen that the Chen/Han method outperforms the Halevi/Shoup method in the three aspects, and the advantage is larger as r increases. Compared to the FBS-based method, the batch method requires fewer homomorphic multiplications. Therefore, we choose the Chen/Han method to perform the HDE step. The detailed process is shown in Appendix C.1.

6.1.2 TFBS to BFV scheme Switching

After the LUT evaluation with the TFBS scheme, we obtain some ciphertext blocks with the based B. Then, to switch from the TFBS scheme to the BFV

[7] In practice, it should be $2B$ due to negative effect.

scheme, we first need to convert the plaintext modulus to B^l. In addition, similar to the existing frameworks PEGASUS [31] and HERMES [4], the switching process involves a ring packing step that packs the LWEs into an RLWE while lifting modulus and dimension of ciphertext in bootstrapping. Referring to the HERMES. For LUT as $f : B^l \to B$, given N LWEs, the whole process works as follows.

- Use the functional bootstrapping for each LWE to convert the plaintext modulus as
$$N \cdot \mathrm{LWE}^n_{B/q} \xrightarrow{\mathsf{FBS}} N \cdot \mathrm{LWE}^n_{B^l/q}$$
by setting the test polynomial $\mathsf{testP}(X) = \sum_{i=0}^{N-1} \frac{Q}{B^l} \cdot (\frac{i \cdot B}{q}) \cdot X^i$, where Q is the modulus in RLWE and RGSW of blind rotation
- Call N/n times LWE-to-RLWE key switching [17] to pack N LWEs into some small-degree RLWEs as
$$N \cdot \mathrm{LWE}^n_{B^l/q} \xrightarrow{\mathsf{LRKS}} N/k \cdot \mathrm{RLWE}_{k,B^l/q}.$$
- Combine the small-degree RLWEs into a large-degree RLWE by ring switching [4], i.e.,
$$N/k \cdot \mathrm{RLWE}_{B^l,q,k} \xrightarrow{\mathsf{RS}} 1 \cdot \mathrm{RLWE}_{B^l,q,N}.$$
- Perform the BFV bootstrapping (BFV-BS) as
$$\mathrm{RLWE}_{B^l,q,N} \xrightarrow{\mathsf{BFV-BS}} \mathrm{RLWE}_{B^l,Q,N}.$$
The BFV bootstrapping involves two operations, coefficient-to-slot transformation, and HED to remove the noise. Meanwhile, the bootstrapping process can lift the ciphertext modulus, and the details can be referred to schemes [13,27].

6.2 Homomorphic Sbox Evaluation with TFBS algorithm

Currently, the homomorphic evaluation of AES based on various fully homomorphic encryption schemes has been proposed, such as BGV-based evaluation [23], CKKS-based evaluation [1], and FHEW/TFHE-based evaluation [35,36]. In particular, the most significant difference between them is the computation strategy of SubBytes, which is the only non-linear function in AES. As far as we know, the FHEW/TFHE-based homomorphic computation achieves the optimal latency and can be divided into two strategies: functional bootstrapping and circuit bootstrapping.

Trama et al. [35] proposed the homomorphic computation of AES using functional bootstrapping mode based on the TFBS algorithm. They divided the 8-to-8-bit lookup table into two 8-to-4-bit lookup tables and used tree-based functional bootstrapping to combine them. This method needs 4 blind rotations and 2 BAKS operations. Our new LFBS method can accelerate Sbox evaluation, which only involves 2 blind rotations and 2 PRCA operations. For the overall AES homomorphism computation, we still need to consider other linear operations including ShiftRows, MixColumns, and AddRoundKey.

7 Conclusion

This paper introduces a new evaluation mode for the FHEW scheme and shows how to build a more efficient leveled functional bootstrapping for large precision LUT evaluation. We first designed a ciphertext conversion, which can convert the LWE ciphertext to RGSW ciphertext with the cyclic group encoding. Based on the ciphertext conversion, we construct a leveled tree-based functional bootstrapping scheme, which uses the efficient external product to replace the blind rotation. Finally, we propose a novel scheme switching framework for this evaluation mode, and it can support large precision polynomial and non-polynomial computation between BFV and the proposed TFBS scheme.

References

1. Aharoni, E., Drucker, N., Ezov, G., Kushnir, E., Shaul, H., Soceanu, O.: E2e near-standard and practical authenticated transciphering. Cryptology ePrint Archive (2023)
2. Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., et al.: Openfhe: Open-source fully homomorphic encryption library. In: proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography. pp. 53–63 (2022)
3. Badawi, A.A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915 (2022), https://eprint.iacr.org/2022/915, https://eprint.iacr.org/2022/915
4. Bae, Y., Cheon, J.H., Kim, J., Park, J.H., Stehlé, D.: Hermes: efficient ring packing using mlwe ciphertexts and application to transciphering. In: Annual International Cryptology Conference. pp. 37–69. Springer (2023)
5. Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Parameter optimization and larger precision for (t) fhe. Journal of Cryptology **36**(3), 28 (2023)
6. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. Journal of Mathematical Cryptology **14**(1), 316–338 (2020)
7. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III 38. pp. 483–512. Springer (2018)
8. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
9. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
10. Brenna, L., Singh, I.S., Johansen, H.D., Johansen, D.: Tfhe-rs: A library for safe and secure remote computing using fully homomorphic encryption and trusted execution environments. Array **13**, 100118 (2022)

11. Carpov, S., Izabachène, M., Mollimard, V.: New techniques for multi-value input homomorphic evaluation and applications. 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. 106–126. Springer (2019)
12. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient homomorphic conversion between (ring) lwe ciphertexts. In: International Conference on Applied Cryptography and Network Security. pp. 460–479. Springer (2021)
13. 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)
14. 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)
15. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22. pp. 3–33. Springer (2016)
16. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 377–408. Springer (2017)
17. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
18. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27. pp. 670–699. Springer (2021)
19. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 617–640. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_24, https://doi.org/10.1007/978-3-662-46800-5_24
20. 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)
21. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
22. 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)
23. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Advances in Cryptology - CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 850–867. Springer (2012), https://doi.org/10.1007/978-3-642-32009-5_49
24. Guimarães, A., Borin, E., Aranha, D.F.: Revisiting the functional bootstrap in tfhe. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 229–253 (2021)

25. Ha, J., Lee, J.: Patching and extending the WWL+ circuit bootstrapping method to FFT domains. Cryptology ePrint Archive, Paper 2024/1318 (2024), https://eprint.iacr.org/2024/1318
26. Halevi, S., Shoup, V.: Algorithms in helib. In: Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34. pp. 554–571. Springer (2014)
27. Halevi, S., Shoup, V.: Bootstrapping for helib. Journal of Cryptology **34**(1), 7 (2021)
28. Lee, D., Min, S., Song, Y.: Functional bootstrapping for packed ciphertexts via homomorphic lut evaluation. Cryptology ePrint Archive (2024)
29. Li, Z., Wei, B., Wang, R., Lu, X., Wang, K.: Full domain functional bootstrapping with least significant bit encoding. In: International Conference on Information Security and Cryptology. pp. 203–223. Springer (2023)
30. Liu, Z., Micciancio, D., Polyakov, Y.: Large-precision homomorphic sign evaluation using fhew/tfhe bootstrapping. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 130–160. Springer (2022)
31. Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 42nd IEEE Symposium on Security and Privacy, SP 2021. pp. 1057–1073. IEEE (2021)
32. 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)
33. Micciancio, D., Polyakov, Y.: Bootstrapping in fhew-like cryptosystems. In: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 17–28 (2021)
34. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
35. Trama, D., Clet, P.E., Boudguiga, A., Sirdey, R.: A homomorphic aes evaluation in less than 30 seconds by means of tfhe. In: Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 79–90 (2023)
36. Wang, R., Wen, Y., Li, Z., Lu, X., Wei, B., Liu, K., Wang, K.: Circuit bootstrapping: faster and smaller. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 342–372. Springer (2024)

A Fundamental Algorithms for LWE and RLWE Ciphertext

A.1 Basic operation in BFV scheme

[**Division by** p]. Division by p is a basic operation in the BFV scheme, which takes an encryption of $pm \bmod p^\ell$ and returns an encryption of $m \bmod p^{\ell-1}$. In the BFV scheme, this operation is free, since if $c_0 + c_1 s = \frac{q}{p^\ell} \cdot pm + e + kq$, then the same ciphertext satisfies $c_0 + c_1 s = \frac{q}{p^{\ell-1}} \cdot m + e + kq$, where q is the ciphertext modulus.

[**Mod** p]. Mod p is another fundamental operation in the BFV scheme. Takes as input an encryption of $m \bmod p^\ell$ with ciphertext modulus $q = p^z$, if the $\ell - 1$ most significant digits of m are zero, it returns an encryption of m with plaintext modulus p and ciphertext modulus $q' = p^{z-\ell+1}$. In the BFV scheme, this operation can be achieved through modulus reduction of the ciphertext, because if $c_0 + c_1 s = \frac{p^z}{p^\ell} \cdot m + e + kp^\ell$, then $[c_0]_{p^{z-\ell+1}} + [c_1]_{p^{z-\ell+1}} s = \frac{p^{z-\ell+1}}{p} \cdot m + e$.

A.2 Correctness of LWE modulus switching

Proof. Let the integers $Q > q > t$, by checking the decryption function, we can get

$$\begin{aligned}
&\left\lfloor \frac{q}{Q} \cdot b \right\rceil - \left\lfloor \frac{q}{Q} \cdot \mathbf{a} \right\rceil \bmod q \\
&= \frac{q}{Q} \cdot b - \left\langle \frac{q}{Q} \cdot \mathbf{a}, \mathbf{s} \right\rangle - \langle \mathbf{r}, \mathbf{s} \rangle + r + kq \\
&= \frac{t}{q} \cdot m + \frac{q}{Q} \cdot e - \langle \mathbf{r}, \mathbf{s} \rangle + r + kq.
\end{aligned}$$

According to the central limit heuristic, the error is close to a Gaussian distribution, and its variance is $\sigma^2_{\mathsf{out}} \le (\frac{q}{Q})^2 \cdot \sigma^2_{\mathsf{in}} + \frac{||\mathbf{s}||_2^2+1}{12}$, where the factor $\frac{1}{12}$ is the standard deviation of a uniform distribution in $[-1/2, 1/2]$. Due to $||\mathbf{s}||_2 < \sqrt{n/2}$ for binary secret key, we have $\sigma^2_{\mathsf{out}} \le (\frac{q}{Q})^2 \cdot \sigma^2_{\mathsf{in}} + \frac{n+2}{24}$.

A.3 Correctness of LWE special modulus switching

Proof. Let $w = 2N \cdot 2^{-\vartheta}$. We note $a''_i = \lfloor \frac{w}{q} a_i \rceil = \frac{w}{q} a_i + \bar{a}_i$, then a''_i belongs to a uniform distribution in $\left[\frac{-w}{2}, \frac{w}{2}\right)$ and $\bar{a}_i \in \frac{w}{q} U[\frac{-q}{2w}, \frac{q}{2w})$. It means that $\mathsf{Var}(a''_i) = \frac{w^2-1}{12}$ and $\mathbb{E}(a''_i) = \frac{-1}{2}$, and $\mathsf{Var}(\bar{a}_i) = \frac{1}{12} - \frac{w^2}{12q^2}$ and $\mathbb{E}(\bar{a}_i) = \frac{-w}{2q}$. The decryption process:

$$\begin{aligned}
&b'' - \langle \mathbf{a}'', \mathbf{s} \rangle \\
&= \frac{w}{q}(b - \langle \mathbf{a}, \mathbf{s} \rangle) + \bar{b} - \langle \bar{\mathbf{a}}, \mathbf{s} \rangle \\
&= \frac{w}{q} m + \frac{w}{q} e + \bar{b} - \langle \bar{\mathbf{a}}, \mathbf{s} \rangle .
\end{aligned}$$

The variance of error is $\sigma^2_{\mathsf{out}} = (\frac{2N}{q})^2 \sigma^2_{\mathsf{in}} + \frac{2^\theta}{12} - \frac{N^2}{3q^2} + \frac{n2^\theta}{24} + \frac{nN^2}{12q^2}$ as shown in scheme [18].

A.4 Correctness of LWE key switching

Proof. Let $\mathsf{LK}_{i,j,v} = (\mathbf{a}'_{i,j,v}, \mathbf{a}'_{i,j,v} \cdot \mathbf{s} + v z_i B_k^j + e_{i,j,v})$ for some $\mathbf{a}'_{i,j,v} \in \mathbb{Z}_q^n$ and $e_{i,j,v} \in \chi_\delta$ with the error variance σ^2_{LK}, the output ciphertext is

$$\begin{aligned}
\mathsf{ct}' &= \mathsf{LWE.KeySwitch}(\mathsf{ct}) \\
&= (\mathbf{0}, b) - \sum_{i,j} \mathsf{LK}_{i,j,a_{i,j}} \bmod Q \\
&= (\mathbf{a}', b') \bmod Q \in \mathrm{LWE}^n_{\mathbf{s},Q}(m),
\end{aligned}$$

It outputs a new LWE ciphertext under the secret key $\mathbf{s}$, where $\mathbf{a}' = -\sum_{i,j} \mathbf{a}'_{i,j,a_{i,j}}$ and $b' = b - \mathbf{a} \cdot \mathbf{z} + \mathbf{a}' \cdot \mathbf{s} - \sum_{i,j} e_{i,j,a_{i,j}}$. Thus, the variance of the noise satisfies $\sigma^2_{\mathsf{out}} \le \sigma^2_{\mathsf{in}} + N d_{\mathsf{LK}} \cdot \sigma^2_{\mathsf{LK}}$.

A.5 HomoTrace for RLWE Ciphertexts

We show the HomoTrace operation for Tr_{K_N/K_1} as shown in Alg. 9. Given the ciphertext $\mathrm{RLWE}(\boldsymbol{m}(X))$, where $\boldsymbol{m}(X) = m_0 + \cdots + m_{N-1}X^{N-1}$, the trace evaluation can obtain new ciphertext $\mathrm{RLWE}(m_0)$. Note that this algorithm can be extended to $\mathsf{Eval.}\,\mathrm{Tr}_{K_N/K_n}$ for all $n|N$, where line 1 of the algorithm also needs to be converted to $\mathsf{ct}' = ((\frac{N}{n})^{-1}\boldsymbol{a}, (\frac{N}{n})^{-1}\boldsymbol{b})$.

Algorithm 9 Evaluation Homomorphic Trace for Tr_{K_N/K_1} ($\mathsf{Eval.}\,\mathrm{Tr}_{K_N/K_1}$)

```
Input:
    The RLWE ciphertext $\mathsf{ct} = (\boldsymbol{a}, \boldsymbol{b}) \in \mathsf{RLWE}_{\boldsymbol{s},N,Q}(\boldsymbol{m}(X))$.
    The automorphism keys $\mathsf{ATK}_{\psi_i} = \mathsf{RLWE}'_{\boldsymbol{s},N,Q}(\boldsymbol{s}(X^{2^i+1}))$, where $i \in [1, \log N]$.
Output:
    RLWE ciphertext $\mathsf{ct}' \in \mathsf{RLWE}_{\boldsymbol{s},N,Q}(m_0))$.
1: $\mathsf{ct}' = (N^{-1}\boldsymbol{a}, N^{-1}\boldsymbol{b})$,
2: for $i = 1$ to $\log N$ do
3:     $\mathsf{ct}' = \mathsf{ct}' + \mathsf{HomoAuto}_{\psi_{\log N - i+1}}(\mathsf{ct}', \mathsf{ATK})$,
4: end for
5: return $\mathsf{ct}'$.
```

Noise growth. The HomTrace evaluation has $\log N$ iterations, the noise variance of the k-th iteration can be expressed as $\sigma_k^2 \le 4\sigma_{k-1}^2 + \sigma_{\mathsf{AT}}^2$, thus the step adds noises bounded by $\sigma_{\mathsf{HT}}^2 \le (1 + \cdots + 4^{\log N-1})\sigma_{\mathsf{AT}}^2 + 4^{\log N} \cdot (N^2)^{-1} \cdot \sigma_{\mathsf{in}}^2 = \frac{N^2-1}{3}\sigma_{\mathsf{AT}}^2 + \sigma_{\mathsf{in}}^2$, where σ_{AT}^2 is the error variance of automorphism. For the detailed analysis, please refer to scheme [12].

Noise growth of Alg. 5. The error variance of Alg. 5 is similar to Alg.9, with the difference being the automorphism for the B ciphertexts. Due to addition and automorphism operations, the error variance of the 3 line of Alg. 5 is bounded by $\sigma_k^2 \le 8\sigma_{k-1}^2 + \sigma_{\mathsf{AT}}^2$, and after $\log B$ iterations, we can get $\sigma_{\text{step }1}^2 \le \frac{B^3-1}{7}\sigma_{\mathsf{AT}}^2 + B\sigma_{\mathsf{in}}^2$. Then, with the evaluation of HomoTrace in step 2, the variance satisfies $\sigma_{\text{step }2}^2 \le \frac{(N-B)^2-1}{3}\sigma_{\mathsf{AT}}^2 + \sigma_{\text{step }1}^2$. Finally, multiplying by the polynomial temp, the message and noise are repeated for N/B positions, respectively, and this step indeed introduces no new noise. Thus, the overall noise is $\sigma_{\text{out}}^2 \le \varepsilon_{\mathsf{PA}}\sigma_{\mathsf{in}}^2 + \sigma_{\mathsf{PA}}^2$, where $\varepsilon_{\mathsf{PA}} = B$, and

$$\sigma_{\mathsf{PA}}^2 = \left(\frac{(N-B)^2-1}{3} + \frac{B^3-1}{7}\right)\sigma_{\mathsf{AT}}^2.$$

A.6 Base-aware key switching

The TFBS needs to pack B LWEs into the serialized blocks in the RLWE ciphertext as shown in Alg. 10. In addition, we introduce a read version similar to LWE key switching, and then the base-aware key switching key is convert to $\mathsf{BAK}_{i,j,b,v} \in \mathrm{RLWE}_{\boldsymbol{s}}(vB_{\mathsf{BA}}^j s_i \cdot \sum_{q=bN/B}^{(b+1)N/B-1} X^q)$ for $v \in [0, B_{\mathsf{BA}} - 1]$, and the key

Algorithm 10 Base-aware Key Switching (BAKS)

```
Input:
    $B$ LWE samples $\mathsf{ct}^{(i)} = (\mathbf{a}^{(i)}, b^{(i)}) \in \mathrm{LWE}^N_{\mathbf{s},Q}(m_i)$ for $i \in [0, B-1]$.
    The base-aware key switching key $\mathsf{BAK}_{i,j,b} \in \mathrm{RLWE}_{\boldsymbol{s}}(B^j_{\mathsf{BA}} s_i \cdot \sum_{q=bN/B}^{(b+1)N/B-1} X^q)$ for
    $i \in [0, N-1], j \in [0, d_{\mathsf{BA}} - 1]$, and $b \in [0, B-1]$.
Output:
    RLWE sample $\mathsf{ct}' = \mathrm{RLWE}^N_{s,Q}(\tilde{\boldsymbol{m}})$.
1: $\tilde{\boldsymbol{b}} = b^{(0)} + \cdots + b^{(0)} X^{\frac{N}{B}-1} + \cdots + b^{(B-1)} X^{(B-1)\frac{N}{B}} + \cdots + b^{(B-1)} X^{N-1}$
2: for $i = 0$ to $N-1$ do
3:     for $b = 0$ to $B-1$ do
4:         $\tilde{a}_{i,b} = a^{(b)}_i$,
5:         approximate decompose $\tilde{a}_{i,b}$ to get $(a_{i,b,0}, .., a_{i,b,l_{\mathsf{BA}}})$,
6:     end for
7: end for
8: return $\mathsf{ct}' = (0, \tilde{\boldsymbol{b}}) - \sum_{i=0}^{N-1} \sum_{j=0}^{l_{\mathsf{BA}}-1} \sum_{b=0}^{B-1} \tilde{a}_{i,j} \cdot \mathsf{BAK}_{i,j,b}$.
```

switching process is

$$\mathsf{ct}' = (0, \tilde{\boldsymbol{b}}) - \sum_{i=0}^{N-1} \sum_{j=0}^{d_{\mathsf{BA}}-1} \sum_{b=0}^{B-1} \mathsf{BAK}_{i,j,b,\tilde{a}_{i,j}},$$

and the error variance is

$$\sigma^2_{\mathsf{out}} = \sigma^2_{\mathsf{in}} + NB \cdot d_{\mathsf{BA}} \cdot \sigma^2_{\mathsf{BA}} + \frac{N}{12} \cdot (\frac{Q}{B^{d_{\mathsf{BA}}}_{\mathsf{BA}}})^2,$$

where σ^2_{in} and σ^2_{BA} are the error variance for input LWE ciphertext and BAK. This method allows for modular multiplication to be replaced with modular addition, improving the efficiency of the key switching process. However, it also results in a significant increase in key size. The sizes of key switching key is $N d_{\mathsf{BA}} B_{\mathsf{BA}} B$ RLWEs, which is B_{BA} times bigger than the computational version.

B Algorithms and Correctness for Ciphertext Conversion

Theorem 1. *Input an* LWE *ciphertext* $\mathsf{ct} = (\mathbf{a}, b) \in \mathrm{LWE}^n_{\mathbf{s},q}(m)$ *with error variance* σ^2_{in}*, Alg. 3 outputs an* RGSW *ciphertext as* $\mathsf{CT} \in \mathrm{RGSW}_{\boldsymbol{s},N,Q}(X^{\varphi(m)+\theta})$*, and its error variance is* σ^2_{CCH}.

Proof. Here, we discuss the correctness and noise growth for the ciphertext conversion of Alg. 3. The analysis can be divided into two steps: special modulus switching and circuit bootstrapping. Firstly, the correctness of the special modulus switching is straightforward, the error variance can be referred to Appendix A.3. Then, the noise from the modulus switching is refreshed by the bootstrapping process.

In addition, lines 5-7 of Alg. 3 outputs the $\mathsf{acc} = \mathrm{RLWE}(B_0 X^{-(\varphi(m)+\theta)} + \cdots + B_{d-1}X^{-(\varphi(m)+\theta)+d-1})$ after n CMux gate with the blind rotation key BRK. Let σ_{BR}^2 is the error variance of BRK, we can get $\sigma_{\mathsf{BR}}^2 = \frac{nNB_{\mathsf{BR}}^2 d_{\mathsf{BR}}}{3}\sigma_{\mathsf{BR}}^2 + \frac{nN}{12}\left\lceil \frac{Q}{B_{\mathsf{BR}}^{d_{\mathsf{BR}}}} \right\rceil^2$. Then, after the HomoTrace evaluation, we can get the ciphertexts $\mathsf{ct}_i = \mathrm{RLWE}_{\boldsymbol{s},Q,N}(B_i X^{\varphi(m)+\theta})$ for $i \in [0, d-1]$. Based on the noise analysis of HomoTrace, the error variance is bound by $\frac{d^2-1}{3}\sigma_{\mathsf{AT}}^2 + \sigma_{\mathsf{BR}}^2$, where σ_{AT}^2 is the error variance of automorphism that equal to variance of RLWE key switching. After that, by performing the RLWE secret key switching with RSK, the new ciphertext is $\mathrm{RLWE}_{\boldsymbol{s},Q,N}(\boldsymbol{s} \cdot B_i X^{\varphi(m)+\theta})$, and the output variance is $\sigma_{\mathsf{CCH}}^2 = \frac{N}{2}(\frac{d^2-1}{3}\sigma_{\mathsf{AT}}^2 + \sigma_{\mathsf{BR}}^2) + \frac{NB_{\mathsf{RK}}^2 d_{\mathsf{RK}}}{12}\sigma_{\mathsf{RK}}^2 + \frac{N}{12}\left\lceil \frac{Q}{B_{\mathsf{RK}}^{d_{\mathsf{RK}}}} \right\rceil^2$. Finally, these RLWE ciphertexts consist of the RGSW ciphertext.

C Switching from BFV scheme to TFBS algorithm

C.1 The workflow of BFV to TFBS scheme switching

We present BFV to the TFBS scheme switching as shown in Fig. 5. Given an RLWE ciphertext, where the i-th slot is $w_i \in \mathbb{Z}_{B^l}$. First, we perform HDE step to obtain l RLWE ciphertexts, where each ciphertext encrypts the corresponding digits of the plaintext vector. Then, the slot-to-coefficient transformation [27] is used to get the coefficient encoding form. Finally, sample extraction is used to obtain $\mathrm{LWE}(w_i^j)$, where i is the index of the slot, and j is the index of the digit. In this way, we complete the BFV to TFBS scheme switching and the message encoding form can support the large precision LUT evaluation.

$$\begin{tikzcd}
\mathrm{RLWE}_{B^d}(w_0, w_1, \cdots, w_{N-1}) \arrow[d, "\text{Digit extraction}"] \\
\mathrm{RLWE}_B(w_0^j, w_1^j, \cdots, w_{N-1}^j), j = 0, \cdots, l-1 \arrow[d, "\text{Slot-to-Coefficient}"] \\
\mathrm{RLWE}_B(w_0^j + w_1^j X + \cdots + w_{N-1}^j)X^{N-1}, j = 0, \cdots, l-1 \arrow[d, "\text{Sample Extraction}"] \\
\mathrm{LWE}_B(w_i^j), i = 0, \cdots, N-1, j = 0, \cdots, l-1
\end{tikzcd}$$

Fig. 5. The workflow of BFV to TFBS scheme switching