Efficient Full Domain Functional Bootstrapping from Recursive LUT Decomposition

Intak Hwang, Shinwon Lee, Seonhong Min, and Yongsoo Song

Seoul National University
{intak.hwang, chowon0708, minsh, y.song}@snu.ac.kr

Abstract. Fully Homomorphic Encryption over the Torus (TFHE) enables efficient evaluation of arbitrary lookup tables (LUT) over encrypted data, allowing complex functions to be computed without decryption. However, in TFHE, only lookup tables with a negacyclic structure can be homomorphically evaluated, which limits the range of functions that can be supported. To overcome this limitation and enable the evaluation of arbitrary functions, the notion of full-domain functional bootstrapping (FDFB) was introduced. However, existing FDFB methods require at least two consecutive bootstrapping operations to evaluate a single function, resulting in significant latency and overhead.
In this work, we present a novel FDFB scheme that supports arbitrary lookup tables by decomposing them into multiple small negacyclic LUTs and one compact full-domain LUT. This structure allows most computations to be handled by fast negacyclic bootstrapping, significantly reducing the computational cost. To address the need for maintaining distinct evaluation keys for each LUT length, we apply Extended Bootstrapping (PKC 2021), which enables all operations to run within a fixed ring dimension. Combined with Extended Bootstrapping, our method nearly halves the bootstrapping cost compared to prior FDFB approaches while maintaining a constant key size, negligible parameter overhead, and strong scalability.
Finally, we implement our algorithm using the TFHE-go library and evaluate its performance across various settings. Our method achieves up to a 3.41× speedup over previous FDFB schemes without increasing key size, and retains up to a 1.91× advantage even when Extended Bootstrapping is applied to both.

Keywords: Homomorphic Encryption, Bootstrapping

1 Introduction

Homomorphic encryption(HE) is a cryptosystem that enables computation directly on encrypted data without requiring decryption. Since its first realization by Gentry [12], it has emerged as a powerful tool for enabling secure computation in various applications. In the past decade, there has been significant progress in performance for homomorphic encryptions. The current best-performing homomorphic encryption schemes are BGV [5], BFV [4,11], CKKS [7], TFHE [8], and

FHEW [10], all of which derive their security from the hardness of the Learning With Errors (LWE)[25] and its variant over the ring of cyclotomic integers (RLWE)[23,26].

While most homomorphic encryption schemes primarily focus on supporting addition and multiplication over encrypted data, TFHE [8] and FHEW [10] distinguish themselves by enabling the direct evaluation of arbitrary Boolean gates with relatively low latency. This gate-by-gate computation model provides a practical foundation for applications that require fine-grained control over encrypted bits. Building on this capability, a series of follow-up works have further expanded the TFHE framework in meaningful directions. A particularly notable advancement is programmable bootstrapping [6,9,14], which builds upon prior support for multi-bit ciphertexts in TFHE by enabling the direct evaluation of complex lookup tables (LUT) during bootstrapping. This allows for expressive computation without introducing additional cost. Consequently, TFHE can efficiently evaluate functions that are difficult to represent using only additions and multiplications.

However, programmable bootstrapping is limited to evaluating LUTs with a negacyclic structure, which restricts the range of computable functions. To be specific, evaluated LUT $f_N : \mathbb{Z}_{2N} \to \mathbb{Z}_q$ should satisfy the negacyclic condition $f_N(i+N) = -f_N(i)$ for $i \in [0, N)$. To circumvent this limitation, a line of research on full-domain functional bootstrapping (FDFB) has emerged [18], aiming to support arbitrary LUT evaluation without such structural constraints. Various instantiations of FDFB have been proposed to improve the efficiency, but all existing approaches fundamentally require at least two bootstrapping operations, which makes the latency of FDFB roughly twice that of single bootstrapping.

1.1 Our contribution

While full-domain functional bootstrapping (FDFB) significantly improves functional flexibility, this comes at the cost of increased computational overhead, as it typically requires multiple rounds of bootstrapping. In this work, we present a new FDFB algorithm that reduces the computational cost by up to half compared to the state-of-the-art FDFB, while incurring negligible parameter overhead and no increase in key size. Moreover, our FDFB algorithm is highly parallelizable, making it well-suited for practical deployment in large-scale secure computation settings.

Our algorithm is based on the key observation that any full-domain lookup table of length $N = 2^m$ can be decomposed into the sum of a negacyclic LUT of length 2^m and a full-domain LUT of length 2^{m-1}. More precisely, for any function f_{2^m} defined over $\mathbb{Z}_{2^m}$, we can write

$$f_{2^m}(i) = \bar{f}_{2^{m-1}}(i) + f_{2^{m-1}}(i),$$

where each component is given by

$$\bar{f}_{2^{m-1}}(i) = \frac{1}{2}\left(f_{2^m}(i) - f_{2^m}(i+2^{m-1})\right), \quad f_{2^{m-1}}(i) = \frac{1}{2}\left(f_{2^m}(i) + f_{2^m}(i+2^{m-1})\right).$$

Here, $\bar{f}_{2^{m-1}}$ corresponds to a function defined over $\mathbb{Z}_{2^m}$ with a negacyclic condition $\bar{f}_{2^{m-1}}(i + 2^{m-1}) = -\bar{f}_{2^{m-1}}(i)$ for $i \in [0, 2^{m-1})$, and $f_{2^{m-1}}$ is a function defined over $\mathbb{Z}_{2^{m-1}}$. By recursively applying the above decomposition μ times, an arbitrary full-domain lookup table can be expressed as the sum of μ negacyclic lookup tables of decreasing sizes and a single full-domain lookup table of length $2^{m-\mu}$, which can be written as $f_{2^m}(i) = f_{2^{m-\mu}}(i) + \sum_{k=1}^{\mu} \bar{f}_{2^{m-k}}(i)$.

This decomposition is particularly useful because FDFB is significantly slower than a single negacyclic functional bootstrapping, generally requiring at least twice the latency. Moreover, its computational cost increases proportionally with the LUT size. By breaking down a large full-domain LUT into several smaller negacyclic LUTs and a single small full-domain LUT, we can perform most of the computation using negacyclic bootstrapping, while only a small portion of the function needs to go through the costly FDFB. As a result, the overall evaluation time is substantially reduced compared to applying FDFB to the full lookup table.

For example, consider a full-domain LUT of length 2^m decomposed into three negacyclic LUTs of sizes 2^{m-1}, 2^{m-2}, and 2^{m-3}, along with a single full-domain LUT of size 2^{m-3}. Traditional FDFB evaluates the entire full-domain LUT using 2 bootstrapping operations over size 2^m. In contrast, our method evaluates the LUT at a total relative cost of $1/2 + 1/4 + 1/8$ with respect to a single negacyclic LUT evaluation, plus an additional $(1/8) \cdot 2$ for evaluating the small full-domain LUT twice, resulting in an overall relative cost of about $9/8$. Overall, this leads to nearly a $2\times$ improvement in bootstrapping efficiency in practice. In addition, our new bootstrapping algorithm is algorithmically parallelizable by evaluating each small lookup tables in parallel. Then, essentially the latency will depend on the evaluation of the largest negacyclic lookup table, which leads to $4\times$ improvements in bootstrapping time when fully parallelized.

While this approach improves computational efficiency, it also requires evaluating LUTs of various lengths, which in turn necessitates maintaining distinct evaluation keys for each LUT size. This is because, to perform bootstrapping, the evaluator must access special public keys (referred to as evaluation keys) that enable operations such as KeySwitch and BlindRotate. These keys are designed to match the polynomial degree of the ciphertext, which is determined by the LUT size, so each LUT length typically requires a separate set of keys tailored to the corresponding ring dimension. To address this issue, we adopt the extended bootstrapping (EBS) [21], which leverages a ring isomorphism to rewrite polynomial operations in $\mathbb{Z}_q[X]/(X^{2^m} + 1)$ as 2^ν independent computations in $\mathbb{Z}_q[X]/(X^{2^{m-\nu}} + 1)$. This allows the evaluator to simulate higher degree bootstrapping using polynomials of lower degree, enabling all computations to remain within a fixed and compact ring dimension. As a result, LUTs of different lengths can be handled within a unified dimension, avoiding the need for multiple bootstrapping key sets.

Finally, we implement our algorithm using the TFHE-go library [15] and evaluate its performance and the required key size for bootstrapping. Our method achieves up to a $3.41\times$ speedup over previous FDFB schemes without any increase in key size, and remains 1.91× faster even when EBS is applied to the previous FDFB. Another key advantage of our algorithm is its high parallelizability. Since the bootstrapping operations for the decomposed lookup tables are independent, they can be executed concurrently. As a result, our approach can achieve even greater performance gains when leveraging parallel execution.

1.2 Related works

Several approaches for full-domain functional bootstrapping (FDFB) have been proposed in the literature:

- WoP-PBS [9] performs FDFB by extracting the most significant bit (MSB) of the encrypted message and using BFV multiplications to select the appropriate lookup table (LUT). This method requires more than two bootstrapping operations and incurs significant noise growth due to the BFV multiplication.
- Kluczniak and Shih [18] present an optimization that replaces BFV multiplications with multiple functional bootstrappings to select the LUT.
- Yang et al. [27] present an FDFB method that uses only two bootstrapping operations, improving efficiency while maintaining compact parameters.
- Kluczniak and Schild [19] and Ma et al. [24] propose several optimized FDFB schemes that improve efficiency over earlier approaches. For example, [24] introduces FDFB-Compress, which compresses the LWE message via a functional bootstrap and applies the target function on the compressed domain, significantly reducing parameter overhead and making it one of the most efficient FDFB schemes to date.
- Bon et al. [3] adopted an odd plaintext modulus to circumvent the negacyclic constraint. However, this approach is inherently restricted to odd plaintext settings.
- Bergerat et al. [1] and Li et al. [22] proposed FDFB schemes that support large-precision ciphertexts, overcoming the limitations of earlier methods whose bootstrapping could not efficiently scale to high-precision inputs.

Nevertheless, all existing approaches fundamentally require at least two bootstrapping operations, which remain a major bottleneck. Multi-value bootstrapping [6] is explored in [19,24] to evaluate multiple LUTs simultaneously in a single bootstrapping operation. This approach aims to reduce the number of bootstrapping steps by encoding multiple output values at once. However, this technique incurs significant noise growth and large parameter overhead, making it impractical for efficient implementations.

As an independent interest, Hwang et al. [16] recently proposed a new TFHE-like homomorphic encryption scheme which allows full-domain functional bootstrapping with only a single bootstrapping with slightly slower latency compared to the conventional TFHE bootstrapping.

2 Preliminaries

2.1 Notation

We denote the polynomial ring $\mathbb{Z}[X]/(X^N+1)$ by R, where N is defined as 2^m for some integer m. The quotient ring R/qR is denoted by R_q. We also denote the set $\{0,1\}$ by $\mathbb{B}$. For a distribution $\mathcal{X}$, we denote sampling a random variable x from $\mathcal{X}$ as $x \leftarrow \mathcal{X}$. A uniform distribution over a set S is denoted by $\mathcal{U}(S)$, and a Gaussian distribution with mean 0 and variance α^2 is denoted by ρ_α. The discrete Gaussian distribution over a lattice Λ with mean 0 and variance α^2, denoted by $\mathcal{D}_{\Lambda,\alpha}$, is defined as

$$\mathcal{D}_{\Lambda,\alpha}(\mathbf{x}) = \frac{\rho_\alpha(\mathbf{x})}{\sum_{\mathbf{v}\in\Lambda}\rho_\alpha(\mathbf{v})}.$$

We denote the infinity norm of polynomial p by $\|p\|_\infty$ which is defined as the maximum size of the coefficients. Throughout the paper, we use bold uppercase letters (e.g., $\mathbf{A}$) to denote matrices and bold lowercase letters (e.g., $\mathbf{b}$) to denote vectors. All vectors are treated as row matrices. For an integer vector $\mathbf{b}$, an integer a, and a positive integer q, we use the notation $[\mathbf{b}]_q$ and $[a]_q$ to denote the component-wise modular reduction of $\mathbf{b}$ and the modular reduction of a modulo q, respectively.

2.2 Learning with errors

We describe the decisional learning with errors (LWE) assumption [25] and its variant over the ring of cyclotomic integers [23,26].

Definition 1 (Decisional LWE [25]). *For positive integers m, n, q, and distributions χ, ψ over $\mathbb{Z}$, the (decisional) LWE problem $\mathsf{LWE}_{m,n,q,\chi,\psi}$ is to distinguish the following distribution from the uniform distribution $\mathcal{U}(\mathbb{Z}_q^{m\times(n+1)})$:*

$$\{(\mathbf{A}\cdot\mathbf{s}+\mathbf{e} \pmod q, \mathbf{A}) \in \mathbb{Z}_q^{m\times(n+1)} \mid \mathbf{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{m\times n}), \mathbf{s} \leftarrow \chi^n, \mathbf{e} \leftarrow \psi^m\}.$$

Definition 2 (Decisional RLWE [23,26]). *For positive integers m, N, q, and distributions χ', ψ' over R, the (decisional) RLWE problem $\mathsf{RLWE}_{m,N,q,\chi',\psi'}$ is to distinguish the following distribution from the uniform distribution $\mathcal{U}(R_q^{m\times 2})$:*

$$\{(\mathbf{a}\cdot t+\mathbf{e} \pmod q, \mathbf{a}) \in R_q^2 \mid \mathbf{a} \leftarrow \mathcal{U}(R_q^m), t \leftarrow \chi', \mathbf{e} \leftarrow \psi'^m\}.$$

Based on these assumptions, LWE, RLWE cryptosystems can be defined. An LWE encryption of some plaintext $m \in \mathbb{Z}_q$ is given by $(b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$, where $b + \langle \mathbf{a}, \mathbf{s}\rangle = m + e \pmod q$ for the secret $\mathbf{s} \in \mathbb{Z}^n$ and some small noise $e \in \mathbb{Z}$. Similarly, an RLWE encryption of a plaintext $\mu \in R_q$ is given by $(b, a) \in R_q^2$, where $b + a\cdot t = \mu + e \pmod q$ for the secret $t \in R$, and some small noise $e \in R$.

2.3 Gadget Toolkit and RGSW

The gadget toolkit is a commonly used noise management technique for homomorphic encryption. For some integer $d > 0$, we call a vector $\mathbf{g} \in \mathbb{Z}^d$ and a function $h : \mathbb{Z}_q \to \mathbb{Z}^d$ a gadget vector and a decomposition function, respectively, if the following conditions hold for some small $\delta, \epsilon > 0$.

1. For any $a \in \mathbb{Z}_q$, $\|h(a)\|_\infty \leq \delta$.
2. For any $a \in \mathbb{Z}_q$, $\|a - \langle h(a), \mathbf{g}\rangle \pmod q\|_\infty \leq \epsilon$.

A typical example of the gadget vector and decomposition function is the digit decomposition. Assuming that the ciphertext modulus q is a multiple of B^d for some integer $B > 1$, the gadget vector $\mathbf{g}$ is given by $\left(\frac{q}{B^d}, \frac{q}{B^{d-1}}, \dots, \frac{q}{B}\right)$ and the decomposition function h outputs each digit of $\lfloor \frac{B^d}{q} \cdot a \rceil$ in a balanced representation. Then, we can set $\delta = \frac{B}{2}$ and $\epsilon = \frac{q}{2B^d}$ for the inequalities above. It is worth noting that the gadget toolkit can be extended to the ring setting by applying the decomposition function on each coefficient of the input polynomial, so that it outputs a vector of ring elements over R. In this paper, we represent the gadget vector and decomposition function with these two parameters, B and d.

Established upon the gadget toolkit, the RGSW cryptosystem [13], which offers a level-free ciphertext-ciphertext multiplication, can be defined. Given a gadget vector $\mathbf{g} \in \mathbb{Z}^d$ and decomposition function $h : R_q \to R^d$, the RGSW encryption algorithm is described as follows.
• $\underline{\mathsf{RGSW.Enc}(t, \mu)}$: Given the RLWE secret $t \in R$ and a message μ, sample $\mathbf{a} \leftarrow \mathcal{U}(R_q^{2d})$ and $\mathbf{e} \leftarrow \mathcal{D}^{2d}_{\mathbb{Z}^N,\beta}$. Compute and output

$$\begin{bmatrix} \mathbf{g} & \mathbf{0} \\ \mathbf{0} & \mathbf{g} \end{bmatrix} \cdot \mu + \left[-\mathbf{a} \cdot t + \mathbf{e} \pmod q \;\middle|\; \mathbf{a} \right] \in R_q^{2d \times 2}.$$

Now, we describe the *external product*, a multiplicative operation between RLWE and RGSW ciphertexts.

Definition 3 (External Product). *Let $c = (b, a) \in R_q^2$ be an RLWE ciphertext and $\mathbf{C} \in R_q^{2d \times 2}$ be an RGSW ciphertext associated with the gadget vector $\mathbf{g}$ and decomposition function h. The external product, denoted by $c \boxdot \mathbf{C}$ is defined as follows:*

$$c \boxdot \mathbf{C} = \begin{bmatrix} h(b) \\ h(a) \end{bmatrix}^\top \cdot \mathbf{C} \pmod q.$$

2.4 The TFHE scheme

• $\underline{\mathsf{Setup}(1^\lambda)}$: Given a security parameter λ, generate the following public parameters: LWE dimension n, RLWE dimension $N = 2^m$, error parameters α, $\beta > 0$, plaintext and ciphertext modulus p, q, the gadget decomposition base and dimension B, d, and the key-switching base and dimension B', d' with corresponding gadget vector $\mathbf{g} \in \mathbb{Z}^d$, $\mathbf{g'} \in \mathbb{Z}^{d'}$ and decomposition function $h : \mathbb{Z}_q \to \mathbb{Z}^d$, $h' : \mathbb{Z}_q \to \mathbb{Z}^{d'}$. Define *scaling factor* Δ as $\frac{q}{p}$.

• $\underline{\mathsf{KeyGen}(1^\lambda)}$:

- Sample $s_i \leftarrow \mathcal{U}(\mathbb{B})$ for $0 \le i < n$. Return the LWE secret key $\mathbf{s} = (s_0, s_1, \ldots, s_{n-1})$.
- Sample $t_i \leftarrow \mathcal{U}(\mathbb{B})$ for $0 \le i < N$. Return the RLWE secret key $\mathbf{t} = t_0 + t_1 X + \ldots + t_{N-1} X^{N-1}$. Write $\mathbf{t} = (t_0, t_1, \ldots, t_{N-1})$.
- Set $\mathsf{BRK}_i \leftarrow \mathsf{RGSW.Enc}(t, s_i)$ for $0 \le i < n$. Return the blind rotation key $\mathsf{BRK} = \{\mathsf{BRK}_i\}_{0 \le i < n}$, which is also referred to as the bootstrapping key.
- Sample $\mathbf{A}_{i,1} \leftarrow \mathcal{U}(\mathbb{Z}_q^{d' \times n})$ and $\mathbf{e}_i \leftarrow \mathcal{D}^{d'}_{\mathbb{Z},\alpha}$ for $0 \le i < N$. Set $\mathsf{KSK}_i \leftarrow (\mathbf{A}_{i,0} | \mathbf{A}_{i,1}) \in \mathbb{Z}_q^{d' \times (n+1)}$ where $\mathbf{A}_{i,0} = -\mathbf{A}_{i,1} \cdot s + \mathbf{g'} \cdot t_i + \mathbf{e}_i \pmod q$. Return the key switching key $\mathsf{KSK} = \{\mathsf{KSK}_i\}_{0 \le i < N}$

• $\underline{\mathsf{Enc}(\mathbf{s}, m)}$: For a given secret key $\mathbf{s}$, and a message $m \in \mathbb{Z}_p$, sample $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1}) \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$ and $e \leftarrow \mathcal{D}_\alpha$. Return $\mathbf{c} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$ where $b = -\langle \mathbf{a}, \mathbf{s} \rangle + \Delta m + e \pmod q$.

• $\underline{\mathsf{Dec}(\mathbf{s}, \mathbf{c})}$: For a given secret key $\mathbf{s}$, and a LWE ciphertext $\mathbf{c} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$, let $\mu = b + \langle \mathbf{a}, \mathbf{s} \rangle \pmod q$. Output $m = \lfloor \frac{1}{\Delta} \mu \rceil$.

• $\underline{\mathsf{ModSwitch}(\mathbf{c}, q')}$: For a given LWE ciphertext $\mathbf{c} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$, and a ciphertext modulus q', return $\bar{\mathbf{c}} = (\lfloor \frac{q'}{q} b \rceil, \lfloor \frac{q'}{q} \mathbf{a} \rceil) \in \mathbb{Z}_{q'}^{n+1}$.

• $\underline{\mathsf{BlindRotate}(\bar{\mathbf{c}}, \bar{f}_N, \mathsf{BRK})}$: For a given LWE ciphertext $\bar{\mathbf{c}} = (\bar{b}, \bar{\mathbf{a}}) \in \mathbb{Z}_{2N}^{n+1}$, a negacyclic lookup table $\bar{f}_N : \mathbb{Z}_{2N} \to \mathbb{Z}_q$, and a blind rotation key BRK, run Algorithm 1. Here, $\bar{f}_N$ must satisfy the negacyclic condition

$$\bar{f}_N(x) = -\bar{f}_N(x + N) \quad \text{for all} \quad x \in [0, N).$$

• $\underline{\mathsf{SampleExtract}(\mathsf{ct})}$: For a given RLWE ciphertext $\mathsf{ct} = (\sum_{i=0}^{N-1} b_i X^i, \sum_{i=0}^{N-1} a_i X^i) \in R_q^2$, return $(b_0, a_0, -a_{N-1}, \ldots, -a_1) \in \mathbb{Z}_q^{N+1}$. Note that $\mathsf{SampleExtract}$ does not introduce any additional noise.

• $\underline{\mathsf{KeySwitch}(\mathbf{c}, \mathsf{KSK})}$: For a given LWE ciphertext $\mathbf{c} = (b, a_0, \ldots, a_{N-1}) \in \mathbb{Z}_q^{N+1}$, return $(b, \mathbf{0}) + \sum_{i=0}^{N-1} h'(a_i)^\top \cdot \mathsf{KSK}_i \pmod q$.

• $\underline{\mathsf{Bootstrap}(\mathbf{c}, \bar{f}_N, \mathsf{BRK}, \mathsf{KSK})}$: For a LWE ciphertext $\mathbf{c}$, a given negacyclic lookup table $\bar{f}_N : \mathbb{Z}_{2N} \to \mathbb{Z}_q$, a blind rotation key BRK, and a key-switching key KSK, run Algorithm 2. $\bar{f}_N$ should satisfy the negacyclic condition.

Algorithm 1: BlindRotate

```
Input: A LWE ciphertext $(\bar{b}, \bar{\mathbf{a}}) \in \mathbb{Z}_{2N}^{n+1}$, a negacyclic lookup table
       $\bar{f}_N : \mathbb{Z}_{2N} \to \mathbb{Z}_q$, and the blind rotation key $\mathsf{BRK}$
Output: A RLWE ciphertext $\in R_q^2$
1 Let $tv = \sum_{i=0}^{N-1} \bar{f}_N(i) \cdot X^i$
2 $\mathsf{ACC} \leftarrow (X^{-\bar{b}} \cdot tv, 0)$
3 for $i \leftarrow 0$ to $n-1$ do
4     $\mathsf{ACC} \leftarrow \mathsf{ACC} + \Big((X^{-\bar{a}_i} - 1) \cdot \mathsf{ACC}\Big) \boxdot \mathsf{BRK_i}$
5 end
6 return $\mathsf{ACC}$
```

Algorithm 2: TFHE Bootstrapping

```
Input: A LWE ciphertext $\mathbf{c} \in \mathbb{Z}_q^{N+1}$, a negacyclic lookup table
       $\bar{f}_N : \mathbb{Z}_{2N} \to \mathbb{Z}_q$, the blind rotation key $\mathsf{BRK}$, and the
       key-switching key $\mathsf{KSK}$
Output: A LWE ciphertext $\mathbf{c}_{out} \in \mathbb{Z}_q^{N+1}$
1 $\mathbf{c}_{ks} \leftarrow \mathsf{KeySwitch}(\mathsf{KSK}, \mathbf{c})$
2 $\bar{\mathbf{c}} \leftarrow \mathsf{ModSwitch}(\mathbf{c}_{ks}, 2N)$
3 $\mathsf{ACC} \leftarrow \mathsf{BlindRotate}(\bar{\mathbf{c}}, \bar{f}_N, \mathsf{BRK})$
4 $\mathbf{c}_{out} \leftarrow \mathsf{SampleExtract}(\mathsf{ACC})$
5 return $\mathbf{c}_{out}$
```

• $\underline{\mathsf{FDBlindRotate}(\bar{\mathbf{c}}, f_N, \mathsf{BRK})}$: Given a ciphertext $\bar{\mathbf{c}} = (\bar{b}, \bar{\mathbf{a}}) \in \mathbb{Z}_N^{n+1}$ a full-domain lookup table $f_N : \mathbb{Z}_N \to \mathbb{Z}_q$, and a blind rotation key BRK, homomorphically evaluate the f_N.

FDBlindRotate generalizes BlindRotate by lifting the restriction to negacyclic lookup tables. Various instantiations of FDBlindRotate have been proposed in the literature [18,24,16]. Since FDBlindRotate generally requires two or more blind rotations, FDBlindRotate tends to be significantly more costly than negacyclic BlindRotate.

3 Decomposition of Lookup Table

In this section, we introduce a method to express a large full-domain lookup table into several smaller negacyclic lookup tables and a small full-domain lookup table. At the core of this method is a decomposition that expresses a lookup table $f_{2^m} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$ as the sum of a full-domain lookup table $f_{2^{m-1}} : \mathbb{Z}_{2^{m-1}} \to \mathbb{Z}_q$ and a negacyclic lookup table $\bar{f}_{2^{m-1}} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$. Here, a negacyclic lookup table $\bar{f}_{2^{m-1}}$ refers to a function satisfying the condition

$$\bar{f}_{2^{m-1}}(x) = -\bar{f}_{2^{m-1}}(x + 2^{m-1}) \quad \text{for all} \quad x \in [0, 2^{m-1}).$$

By recursively applying the decomposition to the smaller full-domain lookup tables for μ times, the original lookup table can be rewritten as the sum of μ negacyclic lookup tables and one small full-domain lookup table. Figure 1 provides an intuitive visualization of the recursive decomposition with decomposition depth $\mu = 3$, illustrating how the full-domain lookup table $f_{2^m} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$ is expressed as the sum of 3 negacyclic components and a compact full-domain base table.

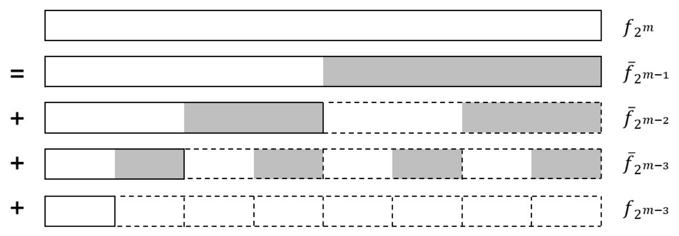

Fig. 1. Recursive decomposition of f_{2^m} with $\mu = 3$.

Following Lemma 1 and Theorem 1 formally state the decomposition structure.

Lemma 1. *Let f_{2^m} be a lookup table $f_{2^m} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$. Then there exists a lookup table $f_{2^{m-1}} : \mathbb{Z}_{2^{m-1}} \to \mathbb{Z}_q$ and a negacyclic lookup table $\bar{f}_{2^{m-1}} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$ such that for all $i \in \mathbb{Z}_{2^m}$, the value $f_{2^m}(i)$ can be reconstructed by the identity*

$$f_{2^m}(i) = \bar{f}_{2^{m-1}}(i) + f_{2^{m-1}}([i]_{2^{m-1}}) + e_i$$

where e_i is a small rounding error satisfying $|e_i| \leq 1$.

Proof. We define $f_{2^{m-1}}$ and $\bar{f}_{2^{m-1}}$ using the values of f_{2^m} as follows.

$$f_{2^{m-1}}(i) := \left\lfloor \frac{1}{2}\big(f_{2^m}(i) + f_{2^m}(i + 2^{m-1})\big) \right\rceil \quad (0 \leq i < 2^{m-1})$$

$$\bar{f}_{2^{m-1}}(i) := \begin{cases} \left\lfloor \frac{1}{2}\big(f_{2^m}(i) - f_{2^m}(i + 2^{m-1})\big) \right\rceil & (0 \leq i < 2^{m-1}) \\ -\bar{f}_{2^{m-1}}(i - 2^{m-1}) & (2^{m-1} \leq i < 2^m) \end{cases}$$

Where $f_{2^m}(i)$ is viewed as a real number, and the arithmetic is performed in $\mathbb{R}$. Then the identity

$$\begin{aligned} &\bar{f}_{2^{m-1}}(i) + f_{2^{m-1}}([i]_{2^{m-1}}) \\ &= \left\lfloor \frac{1}{2}\big(f_{2^m}(i) - f_{2^m}(i + 2^{m-1})\big) \right\rceil + \left\lfloor \frac{1}{2}\big(f_{2^m}(i) + f_{2^m}(i + 2^{m-1})\big) \right\rceil \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2}\big(f_{2^m}(i) - f_{2^m}(i+2^{m-1})\big) + e_i' + \frac{1}{2}\big(f_{2^m}(i) + f_{2^m}(i+2^{m-1})\big) + e_i'' \\
&= f_{2^m}(i) + e_i
\end{aligned}
$$

holds for all $i \in [0, 2^{m-1})$ and

$$
\begin{aligned}
&\bar{f}_{2^{m-1}}(i) + f_{2^{m-1}}([i]_{2^{m-1}}) \\
&= -\left\lfloor \frac{1}{2}\big(f_{2^m}(i-2^{m-1}) - f_{2^m}(i)\big)\right\rceil + \left\lfloor \frac{1}{2}\big(f_{2^m}(i-2^{m-1}) + f_{2^m}(i)\big)\right\rceil \\
&= -\frac{1}{2}\big(f_{2^m}(i-2^{m-1}) - f_{2^m}(i)\big) + e_i' + \frac{1}{2}\big(f_{2^m}(i-2^{m-1}) + f_{2^m}(i)\big) + e_i'' \\
&= f_{2^m}(i) + e_i
\end{aligned}
$$

holds for all $i \in [2^{m-1}, 2^m)$ for some rounding errors e_i', e_i'' with $|e_i'|, |e_i''| \le \frac{1}{2}$, so that their sum $e_i = e_i' + e_i''$ satisfies $|e_i| \le 1$. □

Theorem 1. *Let f_{2^m} be a lookup table $f_{2^m} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$. For a given decomposition depth $1 \le \mu < m$, there exists a sequence of negacyclic lookup tables $\big\{\bar{f}_{2^k} : \mathbb{Z}_{2^{k+1}} \to \mathbb{Z}_q\big\}_{k=m-\mu}^{m-1}$ and a full-domain lookup table $f_{2^{m-\mu}} : \mathbb{Z}_{2^{m-\mu}} \to \mathbb{Z}_q$ such that, for all $i \in \mathbb{Z}_{2^m}$, the value $f_{2^m}(i)$ can be reconstructed by the identity*

$$
f_{2^m}(i) = \sum_{k=m-\mu}^{m-1} \bar{f}_{2^k}([i]_{2^{k+1}}) + f_{2^{m-\mu}}([i]_{2^{m-\mu}}) + e_i.
$$

where e_i is a small rounding error satisfying $|e_i| \le \mu$.

Proof. The proof follows by applying Lemma 1 recursively μ times. □

As $|e_i| \le \mu$ and $q \gg \mu$ in typical settings, the rounding error is negligible. We now present an algorithm that implements the decomposition described in Lemma 1 and Theorem 1.

- DecompLUT(f_{2^m}, μ): For a given full-domain lookup table $f_{2^m} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$ and decomposition depth μ, run Algorithm 3.

This algorithm recursively decomposes f_{2^m} by iteratively applying the decomposition described in Lemma 1. At each level, it computes the sum and difference of the first and second halves of the lookup table. The difference yields a negacyclic lookup table of size 2^k, and the sum becomes a smaller full-domain table of size 2^k used in the next iteration. After μ steps, the algorithm returns μ negacyclic lookup tables and a full-domain lookup table of size $2^{m-\mu}$, which can be used to reconstruct the original table.

4 A Faster Extended Full-Domain Bootstrapping

In this section, we present a new blind rotation algorithm for a faster full-domain functional bootstrapping, based on the recursive lookup table decomposition

```
Algorithm 3: DecompLUT
  Input: A full-domain lookup table $f_{2^m} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$, and a
         decomposition depth $1 \leq \mu < m$
  Output: A sequence of negacyclic lookup table $\{\bar{f}_{2^k}\}_{k=m-\mu}^{m-1}$ where each
          $\bar{f}_{2^k} : \mathbb{Z}_{2^{k+1}} \to \mathbb{Z}_q$ and a full-domain lookup table
          $f_{2^{m-\mu}} : \mathbb{Z}_{2^{m-\mu}} \to \mathbb{Z}_q$
1  $current \leftarrow f_{2^m}$
2  for $k \leftarrow m-1$ down to $m-\mu$ do
3      $n \leftarrow 2^k$
4      for $i \leftarrow 0$ to $n-1$ do
5          $new[i] \leftarrow \left\lfloor \frac{1}{2}\big(current[i] + current[n+i]\big) \right\rceil$
6          $\bar{f}_{2^k}[i] \leftarrow \left\lfloor \frac{1}{2}\big(current[i] - current[n+i]\big) \right\rceil$
7          $\bar{f}_{2^k}[n+i] \leftarrow -\bar{f}_{2^k}[i]$
8      end
9      $current \leftarrow new$
10 end
11 $f_{2^{m-\mu}} \leftarrow current$
12 return $\{\bar{f}_{2^{m-1}}, \dots, \bar{f}_{2^{m-\mu}}\}$, $f_{2^{m-\mu}}$
```

introduced in the previous section. Our method can utilize any existing FDFB method as a baseline without losing its efficiency, and is also algorithmically parallelizable, unlike previous methods.

In prior works, arbitrary lookup tables were evaluated with two or more consecutive negacyclic lookup table evaluations of double the size. Note that a single negacyclic lookup table can be evaluated via TFHE bootstrapping, and the latency of the bootstrapping is proportional to the size of the lookup table. Therefore, when evaluating a size N lookup table, the existing full-domain functional bootstrapping methods require the latency of at least two negacyclic functional bootstrapping of the size $2N$, ending up in roughly $4N$ unit time.

Instead, we propose to decompose the input size N lookup table into several negacyclic lookup tables and a compact lookup table with size $\leq N$, exploiting our novel recursive decomposition method. As a result, we can almost halve the latency of the full-domain functional bootstrapping without any compromise.

4.1 Bootstrapping with Recursive lookup table Decomposition

In Algorithm 4, the full-domain lookup table f_{2^m} is first decomposed using DecompLUT into a sequence of smaller negacyclic lookup tables $\{\bar{f}_{2^k}\}_{k=m-\mu}^{m-1}$ and a base full-domain table $f_{2^{m-\mu}}$. Each negacyclic lookup table $\bar{f}_{2^k} : \mathbb{Z}_{2^{k+1}} \to \mathbb{Z}_q$ satisfies the condition

$$\bar{f}_{2^k}(x) = -\bar{f}_{2^k}(x + 2^k) \quad \text{for all} \quad x \in [0, 2^k).$$

The algorithm then performs μ blind rotation on each $\{\bar{f}_{2^k}\}_{k=m-\mu}^{m-1}$, whose size decreases with k. After each blind rotation, sample extraction and key switching are applied to obtain an LWE ciphertext that is accumulated into the output. After processing all negacyclic components, a full-domain blind rotation is applied to the small-sized base lookup table $f_{2^{m-\mu}}$, along with a final sample extraction and key switching. The result is the sum of all partial ciphertexts, completing the evaluation. We provide the full algorithm in Algorithm 4.

Algorithm 4: Efficient Full Domain Functional Bootstrapping without EBS

```
   Input: A LWE ciphertext $\mathbf{c} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$, a full-domain lookup table
          $f_{2^m} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$, a decomposition depth $1 \le \mu < m$, a set of
          blind rotation key$\{\mathsf{BRK}_{2^k}\}_{k=m-\mu}^{m-1}$, and a set of key-switching
          key $\{\mathsf{KSK}_{2^k}\}_{k=m-\mu}^{m-1}$
   Output: A LWE ciphertext $\mathbf{c}_{out} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$
 1 $\{\bar{f}_{2^k}\}_{k=m-\mu}^{m-1}$ , $f_{2^{m-\mu}} \leftarrow \mathsf{DecompLUT}(f_{2^m}, \mu)$
 2 $(\bar{b}, \bar{\mathbf{a}}) \leftarrow \mathsf{ModSwitch}((b, \mathbf{a}), 2^m)$
 3 Let $\mathbf{c}_{out} = (0, \mathbf{0}) \in \mathbb{Z}_q^{n+1}$
 4 for $k \leftarrow m - \mu$ to $m - 1$ do
 5     $\mathsf{ACC_{res}} \leftarrow \mathsf{BlindRotate}(([\bar{b}]_{2^{k+1}}, [\bar{\mathbf{a}}]_{2^{k+1}}), \bar{f}_{2^k}, \mathsf{BRK}_{2^k})$
 6     $\mathbf{c}' \leftarrow \mathsf{SampleExtract}(\mathsf{ACC_{res}})$
 7     $\mathbf{c}_{out} \leftarrow \mathbf{c}_{out} + \mathsf{KeySwitch}(\mathbf{c}', \mathsf{KSK}_{2^k})$
 8 end
 9 $\mathsf{ACC_{res}} \leftarrow \mathsf{FDBlindRotate}(([\bar{b}]_{2^{m-\mu}}, [\bar{\mathbf{a}}]_{2^{m-\mu}}), f_{2^{m-\mu}}, \mathsf{BRK}_{2^{m-\mu}})$
10 $\mathbf{c}' \leftarrow \mathsf{SampleExtract}(\mathsf{ACC_{res}})$
11 $\mathbf{c}_{out} \leftarrow \mathbf{c}_{out} + \mathsf{KeySwitch}(\mathbf{c}', \mathsf{KSK}_{2^{m-\mu}})$
12 return $\mathbf{c}_{out}$
```

Time complexity. Our decomposition-based algorithm allows us to distribute the computation across smaller lookup tables. Since the cost of blind rotation grows almost linearly with the size of the lookup table, this decomposition leads to substantial savings. Our algorithm returns μ negacyclic lookup tables of size $N, \ldots, N/2^{\mu-1}$ and a regular lookup table of size $N/2^{\mu}$ where μ is the decomposition depth. Then, these lookup tables can be evaluated in a total of $N + \cdots + N/2^{\mu-1} + N/2^{\mu} \cdot 4 = 2N + N/2^{\mu-1}$ unit time without parallelization (N unit time with parallelization). In contrast, as mentioned earlier in this section, existing full-domain bootstrapping methods require a minimum of $4N$ unit time. Therefore, our decomposition-based algorithm provides substantial performance gains over conventional full-domain bootstrapping methods.

Key Size. Let n be the dimension of the input LWE ciphertext. Generally, to evaluate a negacyclic lookup table of size N, we require n RGSW ciphertexts with ring dimension N, and N key-switching keys, which roughly take $n \cdot N$ space complexity, up to a constant determined by the gadget parameters. Therefore, the existing full-domain bootstrapping methods require evaluation keys of the size $2nN$ unit size, as they leverage size $2N$ negacyclic lookup table evaluation. In contrast, our full domain functional bootstrapping algorithm consists of blind rotation over rings with different dimensions, namely $N, N/2, \dots, N/2^{\mu-1}$ for the decomposition depth $\mu > 0$. To facilitate all the blind rotations over these ring degrees, we need evaluation keys of the size $n(N + N/2 + \cdots + N/2^{\mu-1}) = 2nN - nN/2^{\mu-1}$ unit size. To compare with the conventional method, our algorithm shows a slightly smaller key size overhead.

Our algorithm offers significant advantages over conventional full-domain functional bootstrapping, which typically requires two consecutive bootstrapping operations. By leveraging the negacyclic decomposition structure, our approach achieves the same functionality at a substantially reduced computational cost, cutting the overall cost by nearly half. Another important advantage of our method is that each blind rotation operates independently of the others, enabling highly parallelizable and scalable computation across all components. Furthermore, the $\mathsf{FDBlindRotate}$ algorithm for the base full-domain lookup table can be used as a black-box component, allowing any existing FDFB implementation such as [18,24,16] to be directly reused without modification. In addition, since our algorithm treats blind rotation as a black-box subroutine, it is compatible with a wide range of recent optimization techniques such as [2,20], all of which can be orthogonally integrated to further improve performance.

4.2 Optimization via Extended Bootstrapping

In this section, we discuss how to optimize the key size and noise growth for our novel functional bootstrapping method from lookup table decomposition without compromising the computational cost. As specified in the previous section, the evaluation key size of a vanilla TFHE scheme grows linearly to the size of the lookup table evaluated. This makes the evaluation of large lookup tables challenging, as the communication cost and the space complexity are increased. Therefore, we discuss how to make our approach truly scalable and practical by leveraging the *extended bootstrapping* (EBS) technique [21]. As a result, we can evaluate any size lookup table using a single shared blind rotation and key-switching key with a smaller unified ring dimension, by mapping high-dimensional polynomial rings to multiple copies of a lower-dimensional ring.

Extended Bootstrapping (EBS) Let us formally introduce the extended bootstrapping (EBS) proposed in [21]. The main idea of EBS is to rewrite polynomial operations in dimension $N = 2^m$ as 2^ν independent operations in dimension $2^{m-\nu}$ via a module isomorphism

$$\tau : \mathbb{Z}_q[X]/(X^{2^m} + 1) \to \Big(\mathbb{Z}_q[X]/(X^{2^{m-\nu}} + 1)\Big)^{2^\nu},$$

$$\tau\Big(\sum_{i=0}^{2^m} p_i X^i\Big) = \Big(\sum_{i=0}^{2^{m-\nu}} p_{2^\nu i} X^i, \cdots, \sum_{i=0}^{2^{m-\nu}} p_{2^\nu i + 2^\nu - 1} X^i\Big)$$

where ν is referred to as the extend factor. Then,

$$\tau\Big(\mathrm{RLWE}_{t_{\mathsf{ext}}}^{2^m}(pt)\Big) = \Big(\mathrm{RLWE}_t^{2^{m-\nu}}(pt_0), \cdots, \mathrm{RLWE}_t^{2^{m-\nu}}(pt_{2^\nu-1})\Big)$$

holds for $\tau(pt) = (pt_0, \cdots, pt_{2^\nu-1})$. Where the subscript denotes the secret key, the superscript specifies the RLWE ring degree, and the argument represents the encrypted plaintext. The extended key t_{ext} is constructed by copying the entries of t into the indices divisible by 2^ν and setting all other positions to zero.

Furthermore, under the assumption that $z \in \mathbb{Z}$, we can perform RGSW-RLWE external product in parallel by,

$$\begin{aligned}&\mathrm{RGSW}_{t_{\mathsf{ext}}}^{2^m}(z) \boxdot \mathrm{RLWE}_{t_{\mathsf{ext}}}^{2^m}(pt)\\ &\simeq \Big(\mathrm{RGSW}_t^{2^{m-\nu}}(z) \boxdot \mathrm{RLWE}_t^{2^{m-\nu}}(pt_0), \cdots, \mathrm{RGSW}_t^{2^{m-\nu}}(z) \boxdot \mathrm{RLWE}_t^{2^{m-\nu}}(pt_{2^\nu-1})\Big).\end{aligned}$$

Specifically, the decomposition enables each external product to be performed independently across 2^ν components. Because the RGSW ciphertext operates over a reduced dimension $2^{m-\nu}$, the entire procedure avoids additional key growth even when handling lookup tables of varying lengths.

- $\underline{\mathsf{ExtBlindRotate}_\nu(\bar{\mathbf{c}}, \bar{f}_{2^m}, \mathsf{BRK})}$: For a given LWE ciphertext $\bar{\mathbf{c}} = (\bar{b}, \bar{\mathbf{a}}) \in \mathbb{Z}_{2^{m+1}}^{n+1}$, a negacyclic lookup table $\bar{f}_{2^m} : \mathbb{Z}_{2^{m+1}} \to \mathbb{Z}_q$, an extend factor ν, and a blind rotation key BRK, run Algorithm 5.

  Lines 3–8 in Algorithm 5 perform blind rotation in parallel across the 2^ν components of the decomposed accumulator $\overrightarrow{\mathsf{ACC}}$ using the module isomorphism τ.

We now introduce an optimized version of Algorithm 4 that incorporates the extended bootstrapping (EBS) framework. By leveraging EBS, we can ensure that all RGSW operations are carried out over a unified reduced dimension $N' = 2^{m-\mu}$, thereby enabling the reuse of the same BRK across all components. Moreover, each $\mathsf{ExtBlindRotate}$ outputs a single accumulator $\overrightarrow{\mathsf{ACC}}_0$ of RLWE degree N', allowing all results to be key switched together using a single KSK. This unified structure not only improves computational efficiency but also eliminates the key size overhead that arose in the original algorithm when handling lookup tables of varying lengths. Our optimized blind rotation algorithm with EBS is described in Algorithm 6. In the loop spanning Lines 5–8 of Algorithm 6, lookup tables of different lengths are processed using a common BRK with dimension N' by leveraging the EBS framework. To ensure that the reduced RLWE degree $2^{k-\nu}$ remains fixed at N' for each $\mathsf{ExtBlindRotate}$, the extend factor ν is incremented by 1 in each iteration of the loop. Among the variants of $\mathsf{FDBlindRotate}$, we adopt $\mathsf{FDFB\text{-}Compress}$ proposed in [21] as it introduces minimal parameter overhead and requires the fewest number of bootstrapping operations. Note that

Algorithm 5: $\mathsf{ExtBlindRotate}_\nu$

```
Input: A LWE ciphertext $(\bar{b}, \bar{\mathbf{a}}) \in \mathbb{Z}_{2^{m+1}}^{n+1}$, a negacyclic lookup table
       $\bar{f}_{2^m} : \mathbb{Z}_{2^{m+1}} \to \mathbb{Z}_q$, and the blind rotation key $\mathsf{BRK}$
Output: A decomposed RLWE ciphertext $\overrightarrow{\mathsf{ACC}_0}$ with $\tau^{-1}\left(\overrightarrow{\mathsf{ACC}}\right) \in R_q^2$
1  Let $tv = \sum_{i=0}^{2^m-1} \bar{f}(i) \cdot X^i$
2  $\overrightarrow{\mathsf{ACC}} \leftarrow \tau\left((X^{-\bar{b}} \cdot tv, 0)\right)$
3  for $i \leftarrow 0$ to $n-1$ do
4      $\overrightarrow{\mathsf{RotACC}} \leftarrow \tau\left(X^{-\bar{\mathbf{a}}_i} \cdot \tau^{-1}\left(\overrightarrow{\mathsf{ACC}}\right)\right)$
5      for $j \leftarrow 0$ to $2^\nu - 1$ do
6          $\overrightarrow{\mathsf{ACC}_j} \leftarrow \mathsf{BRK}_i \boxdot \left(\overrightarrow{\mathsf{RotACC}_j} - \overrightarrow{\mathsf{ACC}_j}\right) + \overrightarrow{\mathsf{ACC}_j}$
7      end
8  end
9  return $\overrightarrow{\mathsf{ACC}_0}$
```

the input ciphertext for the call to FDBlindRotate at line Line 10 in Algorithm 6 is modulus switched to $\mathbb{Z}_{2^{m+1}}$, as FDFB-Compress requires a modulus twice the size of the full-domain lookup table.

Algorithm 6: Efficient Full Domain Functional Bootstrapping with EBS

```
Input: A LWE ciphertext $\mathbf{c} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$, a full-domain lookup table
       $f_{2^m} : \mathbb{Z}_{2^m} \to \mathbb{Z}_q$, a decomposition depth $1 \le \mu < m$, the blind
       rotation key $\mathsf{BRK}$ and the key-switching key $\mathsf{KSK}$
Output: A LWE ciphertext $\mathbf{c}_{out} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$
1  $\{\bar{f}_{2^k}\}_{k=m-\mu}^{m-1}$ , $f_{2^{m-\mu}} \leftarrow \mathsf{DecompLUT}(f_{2^m}, \mu)$
2  $(\bar{b}, \bar{\mathbf{a}}) \leftarrow \mathsf{ModSwitch}((b, \mathbf{a}), 2^m)$
3  Let $\mathsf{ACC_{res}} = (0, 0) \in R_q^2$
4  $\nu \leftarrow 0$
5  for $k \leftarrow m - \mu$ to $m - 1$ do
6      $\mathsf{ACC_{res}} \leftarrow \mathsf{ACC_{res}} + \mathsf{ExtBlindRotate}_\nu(([\bar{b}]_{2^{k+1}}, [\bar{\mathbf{a}}]_{2^{k+1}}), \bar{f}_{2^k}, \mathsf{BRK})$
7      $\nu \leftarrow \nu + 1$
8  end
9  $(\tilde{b}, \tilde{\mathbf{a}}) \leftarrow \mathsf{ModSwitch}((b, \mathbf{a}), 2^{m+1})$
10 $\mathsf{ACC_{res}} \leftarrow \mathsf{ACC_{res}} + \mathsf{FDBlindRotate}(([\tilde{b}]_{2^{m-\mu+1}}, [\tilde{\mathbf{a}}]_{2^{m-\mu+1}}), f_{2^{m-\mu}}, \mathsf{BRK})$
11 $\mathbf{c}' \leftarrow \mathsf{SampleExtract}(\mathsf{ACC_{res}})$
12 return $\mathbf{c}_{out} = \mathsf{KeySwitch}(\mathbf{c}', \mathsf{KSK})$
```

The exact description of our scheme is as follows. Enc and Dec, remain the same as in the original TFHE scheme.

- $\underline{\mathsf{Setup}(1^\lambda)}$: Given a security parameter λ, generate the following public parameters: LWE dimension n, lookup table length N, decomposition depth μ where RLWE dimension is $N' = 2^{m-\mu}$, error parameters α, $\beta > 0$, plaintext and ciphertext modulus p, q, the gadget decomposition base and dimension B, d, and the key-switching base and dimension B', d'. Define *scaling factor* Δ as $\frac{q}{p}$.

- $\underline{\mathsf{KeyGen}(1^\lambda)}$:

- Sample $s_i \leftarrow \mathcal{U}(\mathbb{B})$ for $0 \le i < n$. Return the LWE secret key $\mathbf{s} = (s_0, s_1, \ldots, s_{n-1})$.
- Sample $t_i \leftarrow \mathcal{U}(\mathbb{B})$ for $0 \le i < N'$. Return the RLWE secret key $t = t_0 + t_1 X + \ldots + t_{N'-1} X^{N'-1}$. Write $\mathbf{t} = (t_0, t_1, \ldots, t_{N'-1})$.
- Set $\mathsf{BRK}_i \leftarrow \mathsf{RGSW.Enc}(t, s_i)$ for $0 \le i < n$. Return the blind rotation key $\mathsf{BRK} = \{\mathsf{BRK}_i\}_{0 \le i < N'}$.
- Set $\mathsf{KSK} \leftarrow \mathsf{SwitchKeyGen}(\mathbf{s}, \mathbf{t})$. Return the key-switching key KSK.

- $\underline{\mathsf{OurFullDomainBootstrap}(\mathbf{c}, f_N, \mu, \mathsf{BRK}, \mathsf{KSK})}$: For a LWE ciphertext $\mathbf{c}$, a given full-domain lookup table $f_N : \mathbb{Z}_N \to \mathbb{Z}_q$, a decomposition depth μ, a blind rotation key BRK, and a key-switching key KSK, run Algorithm 6.

Even with the application of EBS, our method retains its performance advantage over conventional full-domain bootstrapping. Since the use of EBS is orthogonal to our decomposition-based optimization, the total computation time remains nearly halved compared to traditional methods, achieving up to a $2\times$ speedup.

4.3 Error Analysis

In this section, we present a concrete error analysis of our new bootstrapping algorithm. We assume that each LWE and RLWE component is uniformly distributed over $\mathbb{Z}_q$ and R_q, respectively, under the (R)LWE assumption. As mentioned earlier, the gadget decomposition base and dimension are denoted by B and d, respectively, while the key-switching base and dimension are denoted by B' and d'.

Lemma 2 (ModSwitch and KeySwitch). *Let* $(\bar{b}, \bar{\mathbf{a}}) := \mathsf{ModSwitch}((b, \mathbf{a}), N)$, *and* $\mathbf{c}' := \mathsf{KeySwitch}(\mathbf{c}, \mathsf{KSK})$. *Then, the error variances after each operation satisfy:*

$$\mathsf{Var}\left(\mathsf{Err}\left(\frac{q}{N}\bar{b}, \frac{q}{N}\bar{\mathbf{a}}\right)\right) \le \mathsf{Var}(\mathsf{Err}(b, \mathbf{a})) + V_{\mathsf{MS}},$$
$$\mathsf{Var}(\mathsf{Err}(\mathbf{c}')) \le \mathsf{Var}(\mathsf{Err}(\mathbf{c})) + V_{\mathsf{KS}}.$$

where
$$V_{\mathsf{MS}} := \frac{n+1}{12N^2}, \quad V_{\mathsf{KS}} := \frac{1}{12B'^{2d'}} q^2 N^2 + \frac{1}{12}\alpha^2 d' N B'^2.$$

Proof. Follows from [17, Lemma 8,9]. □

Lemma 3 (Extended Bootstrapping). *Let* $\mathsf{ACC} := \mathsf{ExtBlindRotate}_\nu((\bar{b}, \bar{\mathbf{a}}), \bar{f}_N, \mathsf{BRK})$ *with extend factor* ν*. Then, the error variance after* $\mathsf{ExtBlindRotate}$ *satisfy:*

$$\mathsf{Var}(\mathsf{Err}(\mathsf{ACC})) \leq V_{\mathsf{EBR}} := n\Big(\frac{N}{2^\nu} + 1\Big)\frac{q^2}{6B^{2d}} + \frac{1}{3}dn\Big(\frac{N}{2^\nu}\Big)\beta^2 B^2.$$

In other words, error variance of extended blind rotation is determined by the reduced ring degree $N/2^\nu$*, and thus two instances with the same* $N/2^\nu$ *will have the same error variance.*

Proof. From [21, Proposition 2], we get that V_{EBR} corresponds to the error variance of a blind rotation with a reduced RLWE dimension of $N/2^\nu$ where ν is the extend factor. By substituting $N/2^\nu$ into the expression $n(N+1)\frac{q^2}{6B^{2d}} + \frac{1}{3}dnN\beta^2B^2$ from [17, Lemma 10], which gives the error variance for a blind rotation of RLWE dimension N, we obtain the result in lemma. □

Lemma 4 (Our Bootstrapping). *Input error variance of* $\mathsf{ExtBlindRotate}$ *and* $\mathsf{FDBlindRotate}$ *in Algorithm 6 is*

$$V_{\mathsf{MS}} + (\mu+1)V_{\mathsf{EBR}} + \mu^2 + V_{\mathsf{KS}}, \quad \frac{V_{\mathsf{MS}}}{4} + (\mu+1)V_{\mathsf{EBR}} + \mu^2 + V_{\mathsf{KS}}$$

respectively. Here, μ *denotes the decomposition depth, and* ν *used in the definition of* V_{EBR} *is set equal to* μ*.*

Proof. According to Lemma 3 the error variance introduced by $\mathsf{ModSwitch}$ is V_{MS} when the modulus is switched to $N = 2^m$, and $\frac{1}{4}V_{\mathsf{MS}}$ when switched to $2N = 2^{m+1}$.

During Algorithm 6, $\mathsf{ExtBlindRotate}$ and $\mathsf{FDBlindRotate}$ is invoked $(\mu+1)$ times—specifically in lines 5–7 and line 10. As stated in Lemma 3, each invocation of $\mathsf{ExtBlindRotate}$ and $\mathsf{FDBlindRotate}$ introduces an error term with the same variance V_{EBR}, since they all operate on the same reduced ring degree $N/2^\mu = 2^{m-\mu}$ with EBS. These errors accumulate over $\mu+1$ invocations, resulting in a total variance of $(\mu+1)V_{\mathsf{EBR}}$.

In addition, the rounding error introduced by the LUT decomposition in Theorem 1 contributes an error variance of at most μ^2. Since $\mu < \log N$, this contribution is negligible compared to other dominant terms.

Finally, the key switching step contributes an additional error variance of V_{KS}, which completes the total input variance as stated. □

Note that our blind rotation error is $\log(\mu+1)$ bits larger than the original EBS. However, this increase remains moderate in the overall error budget, as V_{MS} and V_{KS} are still the dominant contributors to the total variance.

5 Experimental Results

In this section, we present experimental results for our algorithm and compare them with previous works. The source code is available at `https://github.com/SNUCP/fast-fdfb`. Our implementation is written in Go, based on the TFHE-go library [15].

5.1 Parameter Selection

We provide benchmark results for both the non-EBS and EBS variants (corresponding to Algorithm 4 and Algorithm 6). The parameter sets for the non-EBS and EBS variants are provided in Table 1 and Table 2, respectively. For each plaintext modulus p ranging from 2^5 to 2^8, we specify distinct parameter sets. In all parameter sets, we fix q, α, and β to 2^{64}, $2^{36.00}$, and $2^{12.65}$, respectively. All parameters are chosen to ensure 128-bit security while keeping the bootstrapping failure probability below 2^{-60}. The value of decomposition depth μ is chosen as large as possible to maximize performance, while ensuring that the reduced RLWE dimension $N' = N/2^\mu$ still achieves 128-bit security. Because the non-EBS variant incurs slightly larger errors during blind rotation and key switching, its gadget parameters are set to slightly larger values in certain cases.

p	n	N	μ	B	d	B'	d'
2^5	1160	2^{12}	1	2^{22}	1	2^7	3
2^6		2^{13}	2		2		
2^7		2^{14}	3				
2^8		2^{15}	4				

Table 1. Parameter sets for the non-EBS variant

p	n	N	μ	B	d	B'	d'
2^5	1160	2^{12}	1	2^{22}	1	2^7	3
2^6		2^{13}	2				
2^7		2^{14}	3		2		
2^8		2^{15}	4				

Table 2. Parameter sets for the EBS variant

5.2 Benchmark Results

Now we present the benchmark results of our full domain bootstrapping algorithm. The non-EBS and EBS variants were configured according to the parameter sets given in Table 1 and Table 2, respectively. All benchmarks were conducted on a single core of a desktop equipped with an Intel Core i7-12700F CPU and 32GB of RAM. Each benchmark was repeated 50 times, and the average runtime was reported. For comparison, we additionally implemented the FDFB-Compress scheme presented in [24], as it offers minimal parameter overhead and requires only two bootstrapping operations, which is the fewest among previously known full-domain bootstrapping schemes.

The resulting benchmark times are summarized in Table 3. We also provide the size of the public keys required for bootstrapping in Table 4.

		$p = 2^5$	$p = 2^6$	$p = 2^7$	$p = 2^8$
Non-EBS	FDFB-Compress	79 ms	297 ms	655 ms	1470 ms
	Ours	59 ms	146 ms	300 ms	648 ms
EBS	FDFB-Compress	70 ms	134 ms	393 ms	823 ms
	Ours	57 ms	91 ms	234 ms	431 ms

Table 3. FDFB performance for each plaintext modulus p.

		$p = 2^5$	$p = 2^6$	$p = 2^7$	$p = 2^8$
Non-EBS	FDFB-Compress	254 MB	798 MB	1.56 GB	3.12 GB
	Ours	127 MB	598 MB	1.36 GB	2.98 GB
EBS	FDFB-Compress	127 MB		199 MB	
	Ours				

Table 4. Bootstrapping key size for each plaintext modulus.

As shown in Table 3 and Table 4, applying EBS to our new bootstrapping algorithm yields up to 1.50× speedup compared to the non-EBS version, while significantly reducing the key size. In particular, without EBS, the key size can grow up to 25.1× larger, which is effectively mitigated by our EBS-based approach.

Moreover, our full-domain bootstrapping algorithm achieves substantially better performance than previous approaches without incurring any additional key size growth. For example, when the plaintext modulus is $p = 2^8$, our method is approximately 3.41× faster than FDFB-Compress without EBS, and 1.91× faster even when EBS is applied. Remarkably, our method is only 4.7% slower than a single instance of negacyclic bootstrapping, suggesting that it is nearly the most optimal full-domain bootstrapping algorithm.

Also, our algorithm can achieve even better performance thanks to its parallelizability. Unlike conventional blind rotation, which is inherently difficult to parallelize due to sequential dependencies, our method allows each blind rotation over decomposed LUTs to operate independently, enabling highly parallel and scalable computation across all components.

Acknowledgements

This work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2023-00211649) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. RS-2024-00399491, Development of Privacy-Preserving Multiparty Computation Techniques for Secure Multiparty Data Integration).

References

1. Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J., Tap, S.: Parameter optimization and larger precision for (T)FHE. J. Cryptol. **36**(3), 28 (2023). `https://doi.org/10.1007/S00145-023-09463-5`, `https://doi.org/10.1007/s00145-023-09463-5`
2. Bergerat, L., Chillotti, I., Ligier, D., Orfila, J.B., Roux-Langlois, A., Tap, S.: New secret keys for enhanced performance in (T)FHE. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024: 31st Conference on Computer and Communications Security. pp. 2547–2561. ACM Press, Salt Lake City, UT, USA (October 14–18, 2024). `https://doi.org/10.1145/3658644.3670376`
3. Bon, N., Pointcheval, D., Rivain, M.: Optimized homomorphic evaluation of boolean functions. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2024**(3), 302–341 (2024). `https://doi.org/10.46586/TCHES.V2024.I3.302-341`, `https://doi.org/10.46586/tches.v2024.i3.302-341`
4. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual cryptology conference. pp. 868–886. Springer (2012)
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
6. Carpov, S., Izabachène, M., Mollimard, V.: New techniques for multi-value input homomorphic evaluation and applications. In: Cryptographers' Track at the RSA Conference. pp. 106–126. Springer (2019)
7. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in cryptology–ASIACRYPT 2017: 23rd international conference on the theory and applications of cryptology and information security, Hong kong, China, December 3-7, 2017, proceedings, part i 23. pp. 409–437. Springer (2017)
8. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
9. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. In: International

Conference on the Theory and Application of Cryptology and Information Security. pp. 670–699. Springer (2021)
10. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 617–640. Springer Berlin Heidelberg, Germany, Sofia, Bulgaria (April 26–30, 2015). https://doi.org/10.1007/978-3-662-46800-5_24
11. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
12. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. p. 169–178. STOC '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
13. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I. pp. 75–92. Springer (2013)
14. 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)
15. Hwang, I.: TFHE-go. Online: https://github.com/sp301415/tfhe-go (2023)
16. Hwang, I., Min, S., Song, Y.: Carousel: Fully homomorphic encryption with bootstrapping over automorphism group. Cryptology ePrint Archive, Paper 2024/2032 (2024), https://eprint.iacr.org/2024/2032
17. Hwang, I., Min, S., Song, Y.: Practical circuit privacy/sanitization for TFHE. Cryptology ePrint Archive, Paper 2025/216 (2025), https://eprint.iacr.org/2025/216
18. Kluczniak, K., Schild, L.: FDFB: full domain functional bootstrapping towards practical fully homomorphic encryption. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 501–537 (2023). https://doi.org/10.46586/TCHES.V2023.I1.501-537, https://doi.org/10.46586/tches.v2023.i1.501-537
19. Kluczniak, K., Schild, L.: FDFB^2: Functional bootstrapping via sparse polynomial multiplication. Cryptology ePrint Archive, Paper 2024/1376 (2024), https://eprint.iacr.org/2024/1376
20. Lee, C., Min, S., Seo, J., Song, Y.: Faster TFHE bootstrapping with block binary keys. In: Liu, J.K., Xiang, Y., Nepal, S., Tsudik, G. (eds.) Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2023, Melbourne, VIC, Australia, July 10-14, 2023. pp. 2–13. ACM (2023). https://doi.org/10.1145/3579856.3595804, https://doi.org/10.1145/3579856.3595804
21. Lee, K., Yoon, J.W.: Discretization error reduction for high precision torus fully homomorphic encryption. In: Boldyreva, A., Kolesnikov, V. (eds.) Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13941, pp. 33–62. Springer (2023). https://doi.org/10.1007/978-3-031-31371-4_2, https://doi.org/10.1007/978-3-031-31371-4_2
22. Liu, Z., Micciancio, D., Polyakov, Y.: Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In: Agrawal, S., Lin, D. (eds.) Advances in

Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13792, pp. 130–160. Springer (2022). https://doi.org/10.1007/978-3-031-22966-4_5, https://doi.org/10.1007/978-3-031-22966-4_5

23. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29. pp. 1–23. Springer (2010)
24. Ma, S., Huang, T., Wang, A., Zhou, Q., Wang, X.: Fast and accurate: Efficient full-domain functional bootstrap and digit decomposition for homomorphic computation. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2024**(1), 592–616 (2024). https://doi.org/10.46586/TCHES.V2024.I1.592-616, https://doi.org/10.46586/tches.v2024.i1.592-616
25. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
26. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 617–635. Springer (2009)
27. Yang, Z., Xie, X., Shen, H., Chen, S., Zhou, J.: TOTA: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Paper 2021/1347 (2021), https://eprint.iacr.org/2021/1347