Circuit Bootstrapping: Faster and Smaller

Ruida Wang^{1,2}, Yundi Wen^{3}, Zhihao Li^{1,2}, Xianhui Lu$^{1(\boxtimes)}$, Benqiang Wei^{1,2}, Kun Liu^{1,2}, and Kunpeng Wang^{1}

^{1} Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
^{2} School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
^{3} Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China
{wangruida, luxianhui}@iie.ac.cn

Abstract. We present a novel circuit bootstrapping algorithm that outperforms the state-of-the-art TFHE method with 9.9× speedup and 15.6× key size reduction. These improvements can be attributed to two technical contributions. Firstly, we redesigned the circuit bootstrapping workflow to operate exclusively under the ring ciphertext type, which eliminates the need for conversion between LWE and RLWE ciphertexts. Secondly, we improve the LMKC+ blind rotation algorithm by reducing the number of automorphisms, then propose the first automorphism type multi-value functional bootstrapping. These automorphism-based techniques lead to further key size optimization, and are of independent interest besides circuit bootstrapping. Based on our new circuit bootstrapping we can evaluate AES-128 in 26.2s (single thread), achieving 10.3× speedup compared with the state-of-the-art TFHE-based approach.

Keywords: Circuit Bootstrapping · FHE · TFHE · FHEW.

1 Introduction

Fully homomorphic encryption (FHE) is a powerful tool that enables arbitrary functions to be run directly on encrypted data, yielding the same encrypted results as if the functions were applied to plaintext. Currently, most in-use FHE schemes [3,4,16,8,11] are based on the (ring) learning with errors problem (LWE/RLWE) [33], and introduce noise during encryption. As the homomorphic evaluation goes on, the noises accumulate and may lead to decryption failures. To address this issue, the core procedure of FHE is the bootstrapping technique presented by Gentry [18]. Bootstrapping refreshes the noise of ciphertext, making it possible to perform further homomorphic operations.

Third-generation FHE schemes, including GSW [20], FHEW [2,16] and TFHE [9,10,11], have achieved high bootstrapping efficiency. Among them, the gate

^{0} There are some updates and latest results in https://eprint.iacr.org/2024/1318 bootstrapping algorithm of TFHE, which has advanced into functional bootstrapping (FBS) [5], has the lowest bootstrapping latency [11]. It involves the evaluation of a single-bit logic gate almost for free while refreshing noise. The computation method based on this technique is called the fully homomorphic evaluation mode. However, this mode is less suitable for extensive circuits due to the necessity of bootstrapping after each gate. TFHE also provides a leveled homomorphic evaluation (LHE) mode, enabling the computation of large-scale logic circuits before bootstrapping, such as the weighted finite automata (WFA), bit sequence representation (BSR) and look-up table (LUT). This mode consists of circuit operations, an LWE-to-LWE pre key switching, and a core procedure known as circuit bootstrapping.

1.1 Leveled Homomorphic Evaluation Mode

In the LHE mode, the input and output of the circuit evaluations are two different types: the input is a low-noise RGSW ciphertext, while the output is a high-noise LWE. Therefore, TFHE circuit bootstrapping consists of two steps with different purposes: the first step is functional bootstrapping, which aims to refresh the noise of the ciphertext and perform essential pre-computations. The second step is ciphertext conversion, which uses private key switching (PrivateKS). to convert the LWE ciphertext into its corresponding RGSW form, enabling the next circuit evaluation.

To ensure security and efficiency, the LHE mode spans three levels with different parameters. At Level 0, the ciphertext has a small dimension $\underline{n}$, a small modulus $\underline{q}$, and a large noise. It is the input for the functional bootstrapping, so that a small dimension can improve the efficiency. Level 1 uses medium dimension N and modulus q to balance computational efficiency and the supported circuit depth. Level 2 has the lowest noise and the highest modulus $\bar{Q}$. This is a choice to provide enough depth for the whole LHE evaluation. However, to ensure security, Level 2 necessitates a large dimension $\bar{N}$. The leveled homomorphic evaluation mode and TFHE circuit bootstrapping are described in Fig. 1.

The LHE mode of the TFHE scheme presents an attractive feature in its ability to support large-scale circuit evaluations. However, its practical application has been limited by the excessive delay and large key size associated with circuit bootstrapping. Specifically, the latency of circuit bootstrapping is nearly ten times that of functional bootstrapping [21], which presents an efficiency bottleneck. The following research uses multi-value FBS (MV-FBS) [5] to replace the ℓ times of FBS, where ℓ is the gadget length of the RGSW ciphertext, at the cost of some additional noise. The state-of-the-art algorithm uses another MV-FBS method so-called multi-output programmable bootstrapping (PBSmanyLUT [12]) instead, without noise increases. However, the running time of these improvements is still seven times more than that of FBS. In addition, the key size of the circuit bootstrapping can reach hundreds of megabytes. In particular, homomorphic libraries such as TFHEpp [29] and MOSFHET [21] implement a pre-computed variant of circuit bootstrapping with a key size exceeding 2.6 GB. It puts a significant strain on the bandwidth and storage space

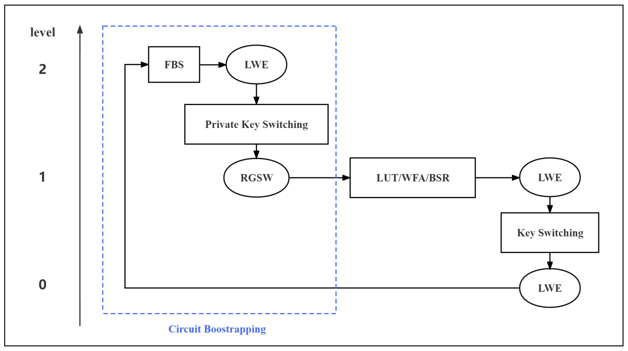


Fig. 1: The LHE mode of TFHE, with circuit bootstrapping highlighted in blue dashed box. Ellipses hold the types of ciphertexts. Rectangles hold algorithms.

in real-world applications, and makes the hardware acceleration through the use of ASIC and FPGA nearly unfeasible. The motivation of this paper is to design a faster and smaller circuit bootstrapping algorithm, as a stepping stone towards the practical FHE applications.

1.2 Our Results

We propose a novel circuit bootstrapping algorithm by making two contributions. Firstly, we reshape the workflow of circuit bootstrapping, resulting in significant improvements in the key size and computation efficiency. Secondly, we optimize the functional bootstrapping step using an automorphism-based variant, which leads to a further optimization of memory efficiency. Finally, we apply our newly proposed algorithm in transciphering, greatly reducing the latency of the homomorphic evaluation of AES circuit.

Novel Workflow. Our redesigned circuit bootstrapping algorithm and its corresponding LHE mode is shown in Fig. 2. The first step is to apply MV-FBS without the sample extraction[4]. The output is an RLWE ciphertext, which eliminates the conversion from LWE to ring ciphertexts in step 2. Our second step involves homomorphic trace evaluation (HomTrace) and scheme switching, which can achieve ciphertext conversion with low computational cost and key size. It should be noted that the step 2 algorithms are unable to switch ciphertext dimensions, so our circuit bootstrapping only spans two levels, with parameters corresponding to level 0 and level 2 of TFHE circuit bootstrapping. The circuit evaluations in LHE mode are also performed on level 2.[5]

[4] Sample extraction is to extract LWE sample from RLWE, which is detailed in Sec. 2.4

[5] This results in a slight increase in circuit computation latency as we discussed in Sup. A, but allows for greater circuit depth and the batch processing of more ciphertexts at once (see TFHE horizontal/ vertical/mixed packing technology [11]).

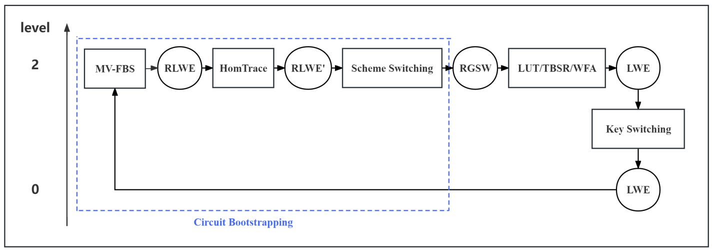


Fig. 2: Our redesigned circuit bootstrapping and the LHE mode, with circuit bootstrapping highlighted in blue dashed box. Ellipses hold the types of ciphertexts. Rectangles hold algorithms.

Under this novel workflow, we propose two variants of circuit bootstrapping algorithms: the controlled-MUX (CMUX) variant performs the MV-FBS step using the CMUX gate; and the automorphism (AUTO) variant performs the MV-FBS step using homomorphic automorphisms.

The Performance of the CMUX Variant. We implement our proposed method based on the OpenFHE [1] library, and demonstrate that our approach has the best performance among all known circuit bootstrapping algorithms:

(1) We implement the unit test of the TFHE circuit bootstrapping method [12] in the same experimental environment, and compare the performance:
   - **Key size**: The key size of our novel workflow in the functional bootstrapping step and the ciphertext conversion step are 19% and 0.2% of the original scheme, respectively. In total, our circuit bootstrapping key size is **6.4%** (15.6×) of the TFHE method.
   - **Runtime**: The computational performance of our novel workflow speedup by factors of 3× and 236× in the functional bootstrapping step and the ciphertext conversion step, respectively. Overall, our approach boosts efficiency by up to **9.9**×.

(2) There is a pre-computed variant of the circuit bootstrapping algorithm implemented in the TFHEpp [29] library and MOSFHET [21] that trades storage for efficiency. This variant pre-stores all possible results of the modulo multiplications in $\mathsf{PrivateKS}$, and then replaces them with modulo addition. Compared with this method, our solution still has a speedup of **3.5**×, and the key size is only **2.2**% (45.5×) that of the pre-computed variant.

The Automorphism-Based Variant. The recently proposed LMKC+ bootstrapping [24] is based on the homomorphic automorphism. This method can reduce the ciphertext dimension and decrease the key size, offering a memory improvement over the GINX method [17]. We propose two optimizations based on LMKC+ and integrate them into our circuit bootstrapping workflow:

(1) We present the first automorphism-based multi-value functional bootstrapping algorithm, which may have independent interests. This method can be

integrated into the MV-FBS step of our proposed circuit bootstrapping workflow. As a result, we can construct an automorphism-based (AUTO) variant. This variant achieves an additional **18.6%** reduction in key size relative to the previously discussed CMUX variant, at the cost of a 10.5% increase in run time. Given its characteristics, this variant is suitable for scenarios where storage is prioritized, such as in mobile devices and hardware acceleration applications.

(2) We give an improved automorphism-based functional bootstrapping algorithm using a sparse isomorphism. Compared to LMKC+, under the parameters of circuit bootstrapping, our method can reduce the number of automorphisms by 10.4%-26.4% (depending on the tolerable decryption failure probability), while also slightly reducing the key size.

Application. To demonstrate the efficacy of theoretical results, we look into the application of our proposed circuit bootstrapping algorithm in *transciphering*. The mission of transciphering is to reduce the size of the transfer ciphertext, making FHE more manageable for devices with limited bandwidth, memory, and computing power. The core idea is to use symmetric encryption for data transmission, and pre-compute into FHE ciphertext by the server before evaluation.

The pre-computation process is to homomorphic evaluate the decryption circuit of the symmetric scheme, which is the efficiency bottleneck of transciphering. Regarding the homomorphic evaluation of the advanced encryption standard (AES), Gentry et al. first used the BGV scheme, with a latency of 18 minutes[6] [19]. The state-of-the-art is recently proposed by Trama et al. using the TFHE scheme [36]. They utilized functional bootstrapping under FHE mode, managing to bring down the algorithm latency to 270 seconds (1 thread). We implement the homomorphic evaluation of AES circuits through our proposed circuit bootstrapping algorithm under LHE mode, with a latency of only **26.2s**. Our result is **10.3**$\times$ more efficient than the functional bootstrapping implementation.

1.3 Technical Overview

Novel Workflow. After the optimization from the PBSmanyLUT algorithm proposed by Chillotti et al. [12], the ciphertext conversion step has become the most time-consuming part of the circuit bootstrapping algorithm with a huge evaluation key size. There are two main reasons: firstly, in the LWE-to-RLWE private key switching algorithm, each evaluation key includes the gadget RLWE ciphertext for every component of the LWE key **sk**. This results in a total of $2(\bar{N}+1)\ell_{\mathsf{ks}}$ RLWE ciphertexts, where $\bar{N}$ represents the dimension of the level 2 LWE ciphertext, and ℓ_{ks} denotes the length of gadget decomposition during PrivateKS. Simultaneously, since the polynomial multiplication must be performed for each key component, the number of multiplications reaches $O(\bar{N}^2)$. Secondly, the noise from PrivateKS will be accumulated into the RGSW ciphertext output by function bootstrapping. Thus it is necessary to select a higher

[6] They also propose a 4-minute variant without bootstrapping. However, it can not be used for transciphering since the ciphertext is too noisy to do further evaluation

gadget decomposition length ℓ_{ks} to control the noise. This further reduces the efficiency of the ciphertext conversion step and increases the key size.

Our approach is to rebuild the circuit bootstrapping algorithm purely on the ring structure. This eliminates the need for switching between LWE and RLWE, as seen in the TFHE method. In this way, the evaluation key becomes the gadget RLWE ciphertext for the RLWE key $\mathbf{sk}$, i.e. for one polynomial instead of every component of a vector. As a result, each switching key contains $O(1)$ RLWE ciphertexts, greatly reducing the key size and computation complexity.

However, removing the LWE ciphertext type from the circuit bootstrapping workflow presents a technical challenge. Specifically, following modulus switching and blind rotation, the bootstrapping algorithm generates an RLWE ciphertext. The desired result is located in the constant term of the polynomial encrypted by this RLWE, while all other powers of X are redundant and may interfere with the computations. The original functional bootstrapping algorithm extracts the required term into an LWE ciphertext type, which is undesirable for us. To address this issue, we employ the technique of the homomorphic trace evaluation (HomTrace) [6], featuring $\mathsf{Trace}(X^i) = 0$, to eliminate the X power term.

The output from HomTrace is a gadget RLWE ciphertext $\mathrm{RLWE}'(\mathbf{m})$. To efficiently reconstruct the RGSW cyphertext, it is necessary to generate $\mathrm{RLWE}'(\mathbf{sk}\cdot\mathbf{m})$ without private key switching. This can be achieved using ℓ external products, where ℓ represents the gadget length. However, our attention was drawn to the recently proposed scheme switching algorithm [14], which accomplishes the same transformation at nearly half the cost of the naive approach. We employ this algorithm to complete our ciphertext conversion step.

In our CMUX circuit bootstrapping variant under the novel workflow, the cost of the ciphertext conversion step, which consists of HomTrace evaluations and scheme switching, accounts for only 3.0% run time of the circuit bootstrapping, and occupies 2.0% of the key size (compared to 70.8% and 66.9% in [12]).

The Automorphism-Based Variant. We enhance the LMKC+ method and propose the first automorphism-based multi-value functional bootstrapping to improve our circuit bootstrapping. The main observation is that the number of automorphisms that need to be computed in the LMKC+ blind rotation is inversely correlated with the a_i's sparsity in $\mathbb{Z}_{2N}$, where a_i is the i-th component of the input LWE ciphertext. Thus a natural approach is to round a_i with greater sparsity. However, there are several technical subtleties for this idea to work:

The naive sparse rounding disrupts the isomorphism $\mathbb{Z}_{2N}^* \cong \mathbb{Z}_{N/2} \otimes \mathbb{Z}_2$ as presented in [24], which is used to reduce key size. In response, we propose a novel sparse isomorphism. Specifically, the numbers of the form $k \cdot 2^{\vartheta} + 1$ within $\mathbb{Z}_{2N}$ is isomorphic to $\mathbb{Z}_{2N/2^{\vartheta}}$. Based on this isomorphism, we improve the automorphism-based functional bootstrapping algorithm by reducing the number of automorphisms. Another challenge is to extend this method to MV-FBS. There are two approaches to achieving MV-FBS, the first multiplies different polynomials after blind rotation to calculate multiple functions [5]. Nonetheless, the noise of the output ciphertext is affected by the polynomial norms, rendering the circuit bootstrapping sensitive to noise amplitude. The second rounds a_i to multiples of 2^ϑ [12], allowing for 2^ϑ simultaneous function evaluations. However, applying this method to our automorphism-based FBS algorithm would destroy the sparse isomorphism. To solve this problem, we devise a new blind rotation algorithm for a_i in the form of $k \cdot 2^\vartheta$. Then we obtain an automorphism-based MV-FBS, which can be used to optimize our circuit bootstrapping algorithm.

1.4 Paper Organization

This paper is organized as follows: Sec.2 reviews the notations and the FHEW-like cryptosystem. Sec.3 describes our circuit bootstrapping workflow, and compares it with prior works. Sec.4 presents an improved automorphism-based FBS and extends it into MV-FBS. Sec.5 provides the analysis of the noise, key size, and computational complexity of proposed algorithms. Sec.6 gives the implementation, and lists the parameters and experimental results. Sec.7 applies our circuit bootstrapping in the transciphering, providing performance and comparative analysis of homomorphic evaluation of AES circuits. Sec.8 concludes the paper.

2 Preliminary

2.1 Notations

Let $\mathbb{A}$ be a set. Define $\mathbb{A}^n$ as the set of vectors with n elements in $\mathbb{A}$, $\mathbb{A}_q$ as the set $\mathbb{A}$ module q, where the elements' scope is $[-q/2, q/2) \cap \mathbb{A}$. Use $\mathbb{Z}$ to denote the set of integers, $\mathbb{R}$ to denote the set of real numbers, and $\mathbb{B} = \mathbb{Z}_2$ represents the set of binary numbers. Denote $\mathcal{R}$ as the set of integer coefficient polynomials modulo $X^N + 1$, where N is a power of 2. Then $\mathcal{R}$ is the $2N$-th cyclotomic ring.

Use regular letters to represent (modular) integers like $a \in \mathbb{Z}_q$, while bold letters to represent polynomials $\mathbf{a} \in \mathcal{R}$ or vectors $\mathbf{a} \in \mathbb{Z}^n$. The notation a_i refers to the i-th coefficient/term of $\mathbf{a}$. The floor, ceiling, and rounding functions are written as $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$ $\lfloor \cdot \rceil$, respectively.

2.2 Gadget Decomposition

Given a gadget vector $\mathbf{v} = (v_0, v_1, ..., v_{\ell-1})$, the gadget decomposition of a ring element $\mathbf{t} \in R_q$ is to find small elements $(\mathbf{t}_0, ..., \mathbf{t}_{\ell-1})$ such that $\sum_i v_i \mathbf{t}_i = (\text{or } \approx)\mathbf{t}$. Gadget decomposition is the key to keeping errors under control in FHE. There are two types of gadget decomposition differing in the selection of gadget vectors.

Canonical Gadget Decomposition. The gadget vector of the canonical gadget decomposition consists of the power of B, where $\mathbf{v} = (1, B, B^2, ..., B^{\ell-1})$. We say $\ell = \lceil \log_B q \rceil$ is the gadget length, and B is the gadget base. With the gadget vectors, each ring polynomial $\mathbf{t}$ can be decomposed into a group of polynomials $(\mathbf{t}_0, ..., \mathbf{t}_{\ell-1})$ with coefficients less than B, such that $\sum_i v_i \mathbf{t}_i = \mathbf{t}$.

Approximate Gadget Decomposition. Approximate gadget decomposition is a generalization of canonical gadget decomposition. When $B^\ell < q$, the decomposition of a ring element is inexact, then we define $\varepsilon_{\mathsf{gadget}}(\mathbf{t}) = \sum_i v_i \mathbf{t}_i - \mathbf{t}$ is the decomposition error, and ϵ is its infinite norm, where $\epsilon = \| \sum_i v_i \mathbf{t}_i - \mathbf{t} \|_\infty$. In general, the approximate gadget vectors is $\mathbf{v} = (\lceil \frac{q}{B^\ell} \rceil, \lceil \frac{q}{B^\ell} \rceil B, ..., \lceil \frac{q}{B^\ell} \rceil B^{\ell-1})$, where $B^\ell < q$. Then each ring polynomial $\mathbf{t}$ can be decomposed into a set of polynomials $(\mathbf{t}_0, ..., \mathbf{t}_{\ell-1})$ with coefficients less than B, satisfying $\epsilon \leq \frac{1}{2} \lceil \frac{q}{B^\ell} \rceil$.

2.3 FHEW-like Cryptosystem

Following the definition from Micciancio and Polyakov [30], we collectively refer to FHEW [2,16] and TFHE [9,10,11] as FHEW-like cryptosystem. The security of the FHEW-like cryptosystem is based on the LWE/RLWE problems [33,28]. We summarize the kinds of (ring) LWE ciphertexts as follows:

- **LWE:** Given two positive integers n and q, the LWE encryption of the message $m \in \mathbb{Z}$ is defined by $\mathrm{LWE}_{\mathbf{sk},q}(m) = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, where $b = -\mathbf{a} \cdot \mathbf{sk} + m + e$. The vector $\mathbf{a}$ is uniformly sampled from $\mathbb{Z}_q^n$, the key $\mathbf{sk}$ is sampled from a key distribution χ, the error e is sampled from an error distribution χ'.

- **RLWE:** RLWE is a ring version of LWE on $\mathcal{R}_q$. The RLWE encryption of the message $\mathbf{m} \in \mathcal{R}_q$ is defined by $\mathrm{RLWE}_{\mathbf{sk},q}(\mathbf{m}) = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_q^2$, where $\mathbf{b} = -\mathbf{a} \cdot \mathbf{sk} + \mathbf{m} + \mathbf{e}$. The vector $\mathbf{a}$ is uniformly sampled from $\mathcal{R}_q$, the key $\mathbf{sk}$ is sampled from a key distribution χ, and each coefficient of the error e_i is sampled from an error distribution χ'.

- **RLWE′:** Given a gadget vector $\mathbf{v} = (v_0, v_1, ..., v_{\ell-1})$, the gadget RLWE written as RLWE′, is defined by:

$$\mathrm{RLWE'}_{\mathbf{sk},q}(\mathbf{m}) = (\mathrm{RLWE}_{\mathbf{sk},q}(v_0 \cdot \mathbf{m}), \cdots, \mathrm{RLWE}_{\mathbf{sk},q}(v_{\ell-1} \cdot \mathbf{m})).$$

- **RGSW:** The RGSW encryption of the message $\mathbf{m} \in \mathcal{R}_q$ is defined by: $\mathrm{RGSW}_{\mathbf{sk},q}(\mathbf{m}) = (\mathrm{RLWE'}_{\mathbf{sk},q}(\mathbf{sk} \cdot \mathbf{m}), \mathrm{RLWE'}_{\mathbf{sk},q}(\mathbf{m}))$.

Remark 1. In FHEW-like cryptosystem, the gadget RLWE appears as an auxiliary input in algorithms such as the gadget product and the key switching. To simplify the presentation and facilitate understanding, we provide the definition and notation of gadget RLWE following Micciancio and Polyakov[30].

2.3.1 Useful Algorithms

Gadget Product: The gadget product $\odot$ is defined by:

$$\begin{aligned}\mathbf{t} \odot \mathrm{RLWE}'_{\mathbf{sk},q}(\mathbf{m}) &:= \sum_{i=0}^{\ell-1} \mathbf{t}_i \cdot \mathrm{RLWE}_{\mathbf{sk},q}\left(v_i \cdot \mathbf{m}\right) \\ &= \mathrm{RLWE}_{\mathbf{sk},q}\left(\sum_{i=0}^{\ell-1} v_i \cdot \mathbf{t}_i \cdot \mathbf{m}\right) \\ &= \mathrm{RLWE}_{\mathbf{sk},q}(\mathbf{t} \cdot \mathbf{m}),\end{aligned}$$

where $(\mathbf{t}_0, ..., \mathbf{t}_{\ell-1})$ is the gadget decomposition of $\mathbf{t}$ with respect to the gadget vector $\mathbf{v} = (v_0, v_1, ..., v_{\ell-1})$.

Lemma 1 (Kim et al.[23]). *Let B and ℓ denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the gadget product is bounded by*

$$\sigma^2_{\odot,\mathsf{input}} \leq \frac{1}{12}\ell B^2 \sigma^2_{\mathsf{input}} + \frac{1}{3}\mathsf{Var}(\mathbf{m})\epsilon^2$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input LWE$'$ *ciphertext, and* $\mathsf{Var}(\mathbf{m})$ *is the variance of the message distribution.*

Lemma.1 is derived from [23] proposition.1 with the fact $\epsilon \leq \frac{1}{2}\lceil \frac{q}{B^\ell} \rceil$. When using canonical gadget decomposition, the error term $\frac{1}{3}\mathsf{Var}(\mathbf{m})\epsilon^2$ does not exist.

External Product: The external product $\boxdot$ is defined by:

$$\begin{aligned}\mathrm{RLWE}_{\mathbf{sk},q}\left(\mathbf{m}_1\right) \boxdot \mathrm{RGSW}_{\mathbf{sk},q}\left(\mathbf{m}_2\right) &:= \mathbf{a} \odot \mathrm{RLWE}'_{\mathbf{sk},q}\left(\mathbf{sk} \cdot \mathbf{m}_2\right) + \mathbf{b} \odot \mathrm{RLWE}'_{\mathbf{sk},q}\left(\mathbf{m}_2\right) \\ &= \mathrm{RLWE}_{\mathbf{sk},q}\left(\mathbf{a} \cdot \mathbf{sk} \cdot \mathbf{m}_2 + \mathbf{b} \cdot \mathbf{m}_2\right) \\ &= \mathrm{RLWE}_{\mathbf{sk},q}\left(\mathbf{m}_1 \cdot \mathbf{m}_2 + \mathbf{e}_1 \cdot \mathbf{m}_2\right),\end{aligned}$$

where $\mathrm{RLWE}_{\mathbf{sk},q}\left(\mathbf{m}_1\right) = (\mathbf{a}, \mathbf{b})$. The error term $\mathbf{e}_1 \cdot \mathbf{m}_2$ will be sufficiently small if $\mathbf{m}_2$ has a small norm, e.g., binary key distribution. Then the external product outputs the RLWE encryption of the product of $\mathbf{m}_1$ and $\mathbf{m}_2$ with the error variance bounded by $\sigma^2_{\boxdot} = 2\sigma^2_{\odot,\mathrm{RGSW}} + \sigma^2_{\mathrm{RLWE}}$, where $\sigma^2_{\odot,\mathrm{RGSW}}$ denotes the error variance of the gadget product result with the input RGSW ciphertext, and σ^2_{RLWE} is the error variance of the input RLWE ciphertext.

LWE-to-RLWE Private Key Switching: Let $f : \mathbb{Z} \to \mathcal{R}$ be a private Lipschitz morphism, given a private key switching key $\mathsf{PrivateKSK}$ contains $\mathrm{RLWE}'_{\mathbf{sk}',q'}(f(s_i))_{i\in[1,n]}$ and $\mathrm{RLWE}'_{\mathbf{sk}',q'}(f(1))$, where s_i is the i-th term of the LWE secret key, then the LWE-to-RLWE private key switching algorithm $\mathsf{PrivateKS}^f_{\mathbf{sk}\to\mathbf{sk}'} : \mathrm{LWE}_{\mathbf{sk},q}(m) \to \mathrm{RLWE}_{\mathbf{sk}',q'}(f(m))$, is defined by:

$$\begin{aligned}\mathsf{PrivateKS}^f_{\mathbf{sk}\to\mathbf{sk}'}(\mathrm{LWE}_{\mathbf{sk},q}(m)) &:= \sum_{i=1}^{n} a_i \odot \mathrm{RLWE}'_{\mathbf{sk}',q'}(f(s_i)) + b \odot \mathrm{RLWE}'_{\mathbf{sk}',q'}(f(1)) \\ &= \mathrm{RLWE}_{\mathbf{sk}',q'}\left(\sum_{i=1}^{n} f(a_i \cdot s_i) + f(b)\right) \\ &= \mathrm{RLWE}_{\mathbf{sk}',q'}\left(f(m)\right),\end{aligned}$$

where $f(m)$ can be considered a monomial in $\mathcal{R}$.

Homomorphic Automorphism: For an automorphism $\psi_t : \mathcal{R} \to \mathcal{R}$ given by $\mathbf{a}(X) \to \mathbf{a}(X^t)$, and an automorphism key $\mathsf{ATK}_t = \mathrm{RLWE'}_{\mathbf{sk},q}(\mathbf{sk}(X^t))$, the homomorphic automorphism $\mathsf{HomAuto}_t : \mathrm{RLWE}_{\mathbf{sk},q}(\mathbf{m}) \to \mathrm{RLWE}_{\mathbf{sk},q}(\mathbf{m}(X^t))$ is defined by:

$$\begin{aligned}\mathsf{HomAuto}_t(\mathrm{RLWE}_{\mathbf{sk},q}(\mathbf{m})) &= \mathbf{a}(X^t) \odot \mathrm{RLWE'}_{\mathbf{sk},q}(\mathbf{sk}(X^t)) + (0, \mathbf{b}(X^t)) \\ &= \mathrm{RLWE}_{\mathbf{sk},q}\left(\mathbf{a}(X^t) \cdot \mathbf{sk}(X^t) + \mathbf{b}(X^t)\right) \\ &= \mathrm{RLWE}_{\mathbf{sk},q}\left(\mathbf{m}(X^t)\right).\end{aligned}$$

Scheme Switching: Given a scheme switching key $\mathsf{SSK} = \mathrm{RLWE}'_{\mathbf{sk},q}(\mathbf{sk}^2)$, the scheme switching algorithm $\mathsf{SS}_{\mathrm{RLWE}' \to \mathrm{RGSW}} : \mathrm{RLWE}'_{\mathbf{sk},q}(\mathbf{m}) \to \mathrm{RGSW}_{\mathbf{sk},q}(\mathbf{m})$ is defined by:
for every $\mathrm{RLWE}_{\mathbf{sk},q}(v_i \cdot \mathbf{m}) = (\mathbf{a}_i, \mathbf{b}_i)$ with error $\mathbf{e}_i$, $i \in [0, \ell - 1]$, compute

$$\begin{aligned}\mathbf{a}_i \odot \mathrm{RLWE'}_{\mathbf{sk},q}\left(\mathbf{sk}^2\right) + (\mathbf{b}_i, 0) &= \mathrm{RLWE}_{\mathbf{sk},q}\left(\mathbf{a}_i \cdot \mathbf{sk}^2 + \mathbf{b}_i \cdot \mathbf{sk}\right) \\ &= \mathrm{RLWE}_{\mathbf{sk},q}\left((\mathbf{a}_i \cdot \mathbf{sk} + \mathbf{b}_i) \cdot \mathbf{sk}\right) \\ &= \mathrm{RLWE}_{\mathbf{sk},q}\left(v_i \cdot \mathbf{sk} \cdot \mathbf{m} + \mathbf{e}_i \cdot \mathbf{sk}\right),\end{aligned}$$

then obtain $\mathrm{RLWE}'_{\mathbf{sk},q}(\mathbf{sk} \cdot \mathbf{m})$ with an additional error $\mathbf{e}_i \cdot \mathbf{sk}$. This term can remain small by choosing the secret key with a small norm (e.g., binary). Then,

$$\mathsf{SS}_{\mathrm{RLWE}' \to \mathrm{RGSW}}(\mathrm{RLWE}'_{\mathbf{sk},q}(\mathbf{m})) := (\mathrm{RLWE}'_{\mathbf{sk},q}(\mathbf{sk} \cdot \mathbf{m}), \mathrm{RLWE}'_{\mathbf{sk},q}(\mathbf{m})).$$

2.4 Functional Bootstrapping

Bootstrapping is the core algorithm in FHE. Functional bootstrapping can evaluate a 1-in/1-out LUT function encoded in a test polynomial testP, while refreshing ciphertext noise. We mainly focus on the GINX bootstrapping [17] and LMKC+ [24] bootstrapping. The bootstrapping algorithm can be divided into three steps:

Modulus Switching. This algorithm receives an LWE ciphertext $(\mathbf{a}, b) \in \mathrm{LWE}_{\mathbf{sk},q}(m)$, and outputs an LWE ciphertext $\mathrm{LWE}_{\mathbf{sk},2N}(m)$ with modulus $2N$, where N is the parameter to denote the RLWE dimension.

- **GINX Modulus Switching:**
  $\mathsf{MS}(\mathrm{LWE}_{\mathbf{sk},q}(m)) := (\lfloor \frac{2N}{q} \cdot \mathbf{a} \rceil, \lfloor \frac{2N}{q} \cdot b \rceil)$.

- **LMKC+ Round-to-Odd Modulus Switching:**
  $\mathsf{MS}_{\mathsf{odd}}(\mathrm{LWE}_{\mathbf{sk},q}(m)) := (\lfloor \frac{2N}{q} \cdot \mathbf{a} \rceil_{\mathsf{odd}}, \lfloor \frac{2N}{q} \cdot b \rceil_{\mathsf{odd}})$, where $\lfloor \cdot \rceil_{\mathsf{odd}}$ outputs the nearest odd integer for the given input.

Blind Rotation. This algorithm receives the LWE output by the modulus switching step and a target function f, it outputs $\mathrm{RLWE}_{\mathbf{sk}}(\mathsf{testP} \cdot X^{b+\sum_i a_i s_i})$, where testP is so-called test polynomial. The coefficients of testP are encoded as

$$\underbrace{(\underbrace{f(0),\dots,f(0)}_{N/2p \text{ elements}},\underbrace{f(1),\dots,f(1)}_{N/p \text{ elements}},\dots,\underbrace{f(p-1),\dots,f(p-1)}_{N/p \text{ elements}},\underbrace{f(0),\dots,f(0)}_{N/2p \text{ elements}})}_{p+1 \text{ blocks and } N \text{ coefficients}}$$

– **GINX CMUX-Based Blind Rotation.**
Given an initial accumulator $\mathsf{acc}_0 = (0, X^{-b} \cdot \mathsf{testP})$ and a bootstrapping key $\mathsf{BSK} = \mathrm{RGSW}_{\mathbf{sk},Q}(s_i)_{i\in[0,n-1]}$, the blind rotation step updates the accumulator by calculating the CMUX gate:

$$\mathsf{acc}_i = \mathsf{acc}_{i-1} \boxdot ((X^{a_i} - 1) \cdot \mathsf{BSK}_i) + \mathsf{acc}_{i-1} \in \mathcal{R}_Q^2.$$

It should be performed n times to get the final result $\mathrm{RLWE}_{\mathbf{sk}}(\mathsf{testP} \cdot X^{b+\sum_i a_i s_i})$.

– **LMKC+ Automorphism-Based Blind Rotation.**
Under an observation that $\mathbb{Z}_{2N}^* \cong \mathbb{Z}_{N/2} \otimes \mathbb{Z}_2$, each odd a_i can be expressed using the generators $\{g, -1\}$, resulting in the equation:

$$\sum_i a_i s_i = \left(\sum_{j\in I_0^+} s_j + \cdots + g\left(\sum_{j\in I_{N/2-1}^+} s_j - g\left(\sum_{j\in I_0^-} s_j + \cdots + g\left(\sum_{j\in I_{N/2-1}^-} s_j\right)\right)\right)\right) \pmod{2N},$$

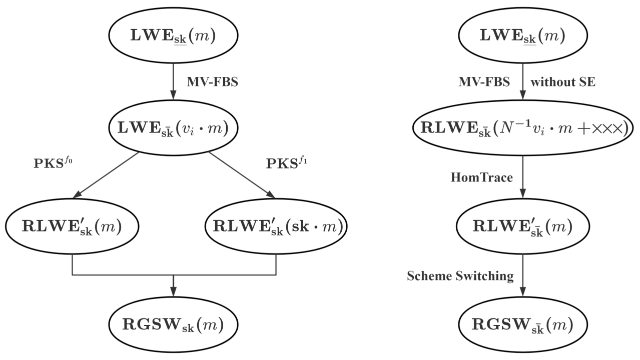
where $I_\ell^+ = \{i : a_i = g^\ell\}$, $I_\ell^- = \{i : a_i = -g^\ell\}$, for $\ell \in [0, N/2-1]$. Then giving an initial $\mathsf{acc} = (0, X^{-gb} \cdot \mathsf{testP}(X^{-g}))$, a bootstrapping key $\mathsf{BSK} = \mathrm{RGSW}_{\mathbf{sk},Q}(X^{s_i})_{i\in[0,n-1]}$, and 2 automorphism keys $\mathsf{ATK}_g, \mathsf{ATK}_{-g}$, the blind rotation algorithm first multiply (external product) the accumulator by BSK_j for all $j \in I_{N/2-1}^-$, then apply $\mathsf{HomAuto}_g$ and repeated for each I (except when after multiplying I_0^-, apply the $\mathsf{HomAuto}_{-g}$). The final results is $\mathrm{RLWE}_{\mathbf{sk}}(\mathsf{testP} \cdot X^{b+\sum_i a_i s_i})$. The detailed LMKC+ blind rotation algorithm is demonstrated in Sup. B.

Sample Extraction. The algorithm $\mathsf{SampleExtract}_i$ receives an RLWE ciphertext $\mathbf{c} = (\mathbf{a}, \mathbf{b}) \in \mathrm{RLWE}_{\mathbf{sk},q}(\mathbf{m})$ and a given position i, it returns an $\mathrm{LWE}_{\mathbf{sk},q}(m_i)$, where m_i is the i-th coefficient of $\mathbf{m}$. The functional bootstrapping algorithm uses $\mathsf{SampleExtract}_0$ to extract $\mathrm{LWE}_{\mathbf{sk},q}(f(m))$ from the RLWE output of the blind rotation step.

2.5 TFHE Circuit Bootstrapping.

Circuit bootstrapping is the core operation to connect TFHE leveled evaluation, which can convert high-noise LWE ciphertext to low-noise RGSW ciphertext. The TFHE circuit bootstrapping consists of the following two steps:

(1) **Functional Bootstrapping Step.** This step call ℓ times functional bootstrapping or 1 $\mathsf{PBSmanyLUT}$ [12] to refresh noise and do the following conversion: $\mathrm{LWE}_{\underline{\mathbf{sk}}}(m) \rightarrow \mathrm{LWE}_{\bar{\mathbf{sk}}}(v_i \cdot m), i \in [0, \ell-1]$, where $\mathbf{v} = (v_0, v_1, ..., v_{\ell-1})$ is the gadget vector.



$\mathbf{LWE}_{\mathrm{sk}}(m)$
MV-FBS
$\mathbf{LWE}_{\bar{\mathrm{sk}}}(v_i \cdot m)$
$\mathbf{PKS}^{f_0}$
$\mathbf{PKS}^{f_1}$
$\mathbf{RLWE}'_{\mathrm{sk}}(m)$
$\mathbf{RLWE}'_{\mathrm{sk}}(\mathbf{sk} \cdot m)$
$\mathbf{RGSW}_{\mathrm{sk}}(m)$
$\mathbf{LWE}_{\mathrm{sk}}(m)$
MV-FBS
without SE
$\mathbf{RLWE}_{\bar{\mathrm{sk}}}(N^{-1}v_i \cdot m + \times\times\times)$
HomTrace
$\mathbf{RLWE}'_{\bar{\mathrm{sk}}}(m)$
Scheme Switching
$\mathbf{RGSW}_{\bar{\mathrm{sk}}}(m)$

Fig. 3: The original and our redesigned circuit bootstrapping workflow.

(2) **Ciphertext Conversion Step.** This step call 2ℓ times of $\mathsf{PrivateKS}$: $\mathsf{PrivateKS}^{f_0}_{\bar{\mathbf{sk}} \to \mathbf{sk}} : \mathrm{LWE}_{\bar{\mathbf{sk}}}(v_i \cdot m) \to \mathrm{RLWE}_{\mathbf{sk}}(v_i \cdot \mathbf{m})$, $\mathsf{PrivateKS}^{f_1}_{\bar{\mathbf{sk}} \to \mathbf{sk}} : \mathrm{LWE}_{\bar{\mathbf{sk}}}(v_i \cdot m) \to \mathrm{RLWE}_{\mathbf{sk}}(v_i \cdot \mathbf{sk} \cdot \mathbf{m}), i \in [0, \ell - 1]$, where f_0 is the identity function, and $f_1(x) = \mathbf{sk} \cdot x$. Then $\mathrm{RGSW}_{\mathbf{sk}}(m) = (\mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{sk} \cdot \mathbf{m}), \mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{m}))$.

3 Novel Work Flow of Circuit Bootstrapping

In the original circuit bootstrapping proposed by Chillotti et al. [10], the first step is achieved by ℓ times of bootstrapping, where ℓ is the length of the gadget decomposition when evaluating the circuit. The second step requires 2ℓ times of private key switching, as shown in Sec.2.5. We introduce a novel circuit bootstrapping workflow that makes it faster and smaller. The original and our redesigned algorithms are shown in Fig.3.

Multi-Value Bootstrapping. The application of a multi-value bootstrapping algorithm to circuit bootstrapping is not new. In the state-of-the-art TFHE circuit bootstrapping method, $\mathsf{PBSmanyLUT}$ is employed to reduce the number of function bootstrappings from ℓ to 1 [12]. This method has already been implemented in the TFHEpp library by Matsuoka et al. [29].

Specifically, the $\mathsf{PBSmanyLUT}$ algorithm uses a specialized modulus switching technique to switch each component a_i of the ciphertext to a multiple of 2^ϑ:

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

As a result, the least ϑ significant bits of the noise are removed, permitting its representation as $e' = e'' \cdot 2^\vartheta$. Consequently, for each increment of 1 in e', during the blind rotation, the test polynomial is rotated by 2^ϑ positions instead of just one. This allows the encoding of 2^ϑ distinct function values at these positions:

$$\underbrace{(\ldots, \underbrace{f_1(m), \ldots, f_{2^\vartheta}(m), \ldots, f_1(m), \ldots, f_{2^\vartheta}(m)}_{N/p \text{ elements}}, \underbrace{f_1(m+1), \ldots, f_{2^\vartheta}(m+1), \ldots, f_1(m+1), \ldots, f_{2^\vartheta}(m+1), \ldots}_{N/p \text{ elements}})}_{p \text{ blocks}}$$

Then by setting the test polynomial to be $\mathsf{testP} = \frac{1}{2}\sum_{i=0}^{\frac{N}{2^{\vartheta}}-1}\sum_{j=0}^{2^{\vartheta}-1} v_i X^{2^{\vartheta}\cdot i+j}$, the ℓ independent FBS in circuit bootstrapping step 1 can be replaced with:

$$\begin{cases} \{\bar{\mathbf{c}}_i\}_{i\in[0,\bar{\ell}-1]} \leftarrow \mathsf{PBSmanyLUT}\left(\mathrm{LWE}_{\underline{\mathbf{sk}},\underline{q}}(m), \mathsf{BSK}, \mathsf{testP}\cdot X^{N/2^{\vartheta+1}}\right) \\ \forall i \in [0,\bar{\ell}-1], \bar{\mathbf{c}}_i + \left(\mathbf{0}, \frac{1}{2}v_i\right) \end{cases}$$

3.1 Step 1: Multi-Value Functional Bootstrapping without Sample Extraction

We adopt the concept of using MV-FBS to reduce the number of bootstrappings, incorporating two improvements. This step receives a high-noise LWE ciphertext, aims to refresh the noise, and transforms it into level 2 *redundant* RLWE ciphertexts, where y_i coefficients are useless:

$$\mathrm{LWE}_{\underline{\mathbf{sk}},\underline{q}}(m) \rightarrow \mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}(\bar{N}^{-1}v_i\cdot m + y_1X + ... + y_{\bar{N}-1}X^{\bar{N}-1}), i\in[0,\ell-1],$$

The first improvement involves the removal of the $\mathsf{SampleExtract}$ step in the MV-FBS, thereby avoiding the expensive LWE-to-RLWE private key switching. Specifically, we do not extract all $\mathrm{LWE}_{\bar{\mathbf{sk}},\bar{Q}}(v_i\cdot m)$ from the accumulator acc. Instead, when encoding test polynomials, we multiply each coefficient by $\bar{N}^{-1}$, to ensure the correctness of $\mathsf{HomTrace}$ in step 2. Then, after blind rotation, we add polynomials $(0, \frac{1}{2}v_0 + \frac{1}{2}v_1X + ... + \frac{1}{2}v_{\ell-1}X^{\ell-1})$ to acc, and use multiplication by X^{-j} to replace sample extraction. As a result, the term $\bar{N}^{-1}v_i\cdot m$ is rotated to the constant term of the plaintext polynomial. This step outputs l redundant polynomialsRLWE$_{\bar{\mathbf{sk}},\bar{Q}}(\bar{N}^{-1}v_i\cdot m + y_1X + ... + y_{\bar{N}-1}X^{\bar{N}-1}), i\in[0,\ell-1]$. Further details can be found in lines 7-10 in Alg.6.

Secondly, we introduce an automorphism-based circuit bootstrapping algorithm, denoted by AUTO Variant. It can reduce key size while preserving the functionality. We state this algorithm in Sec. 4, due to its complexity.

3.2 Step 2: Ciphertext Conversion

HomTrace. The homomorphic trace algorithm receives ℓ redundant RLWE ciphertexts, and aims to transform them into a valid level 2 RLWE$'$:

$$\mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}\left(\bar{N}^{-1}v_i\cdot m + y_1X + \ldots + y_{\bar{N}-1}X^{\bar{N}-1}\right) \rightarrow \mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}\left(v_i\cdot\mathbf{m}\right), i\in[0,\ell-1],$$
$$\mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\mathbf{m}) = (\mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}(v_0\cdot\mathbf{m}), \mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}(v_1\cdot\mathbf{m}), ..., \mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}(v_{\ell-1}\cdot\mathbf{m})).$$

If omit $\bar{N}^{-1}$, the received redundant RLWE ciphertexts have valid values to reconstruct RGSW in the constant term. Then we perform homomorphic trace evaluation (HomTrace) to remove the X power terms. Specifically, for an algebraic extension E/F, the function Trace denoted as $\mathrm{Trace}(\cdot) = \sum_{\sigma\in\mathrm{Gal}(E/F)}\sigma(\cdot)$ has the unique property: $\mathrm{Trace}(1) = N$ and $\mathrm{Trace}\left(X^i\right) = 0$ for all $0 \neq i \in [N]$. Thus,

Algorithm 1 Homomorphic Evaluation of the Trace Function (HomTrace)

```
Input: the set of RLWE ciphertexts $\{\bar{\mathbf{c}}_j\}_{j\in[0,\ell-1]}$ output by alg.6,
Input: automorphism keys $\mathsf{ATK}_u = \mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\bar{\mathbf{sk}}(X^u))_{u\in\{2,2^2,\ldots,2^{\log N}\}}$,
Output: $\bar{\mathbf{c}}_j = \mathrm{RLWE}_{\bar{\mathbf{sk}}}(v_j \cdot \mathbf{m})$, $j \in [0,\ell-1]$.
1: for $j = 0$ to $\ell - 1$ do:
2:     for $k = 1$ to $\log \bar{N}$ do:
3:         $\bar{\mathbf{c}}_j \leftarrow \bar{\mathbf{c}}_j + \mathsf{HomAuto}_{2^{\log N-k+1}+1}(\bar{\mathbf{c}}_j)$;
4:     end for
5:     return $\bar{\mathbf{c}}_j$
6: end for
```

$$\begin{aligned}
&\mathrm{HomTrace}(\mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}\left(\bar{N}^{-1}v_i\cdot m + y_1X + \ldots + y_{\bar{N}-1}X^{\bar{N}-1}\right))\\
=&\mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}\left(\mathrm{Trace}(\bar{N}^{-1}v_i\cdot m + y_1X + \ldots + y_{\bar{N}-1}X^{\bar{N}-1})\right)\\
=&\mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}\left(v_i\cdot m\right), i\in[0,\ell-1].
\end{aligned}$$

$\bar{N}^{-1}$ is the inverse of $\bar{N}$ in $\mathbb{Z}_{\bar{Q}}$, so that $\bar{N}$ and $\bar{Q}$ should be coprime numbers. In our circuit bootstrapping implementation, we choose $\bar{N}$ to be a power of 2 and $\bar{Q}$ to be an NTT prime.

An intuitive approach to evaluating the trace function is to calculate each Galois automorphism and sum all the results. However, this approach has a high computational complexity. Instead, we employ a more efficient calculation method used by Chen et al. [6] and Liu et al. [25,26].

Specifically, for $1 \le i \le \bar{N}-1$, if we apply the automorphism $\psi_{\bar{N}+1}(X^i) = X^{i(\bar{N}+1)} = (-1)^i X^i$, we simply preserve or flip the sign according to the parity of i. Furthermore, for $2^k \| i$ (i.e., $2^k | i$ while $2^{k+1} \nmid i$), we have

$$\begin{cases}\psi_{\bar{N}/2^k+1}(X^i) = -X^i,\\ \psi_{\bar{N}/2^j+1}(X^i) = X^i, \quad 0\le j<k.\end{cases}$$

Define an automorphism $\phi_t = \psi_1 + \psi_{\bar{N}/2^t+1}$. It is clear that for $2^k \| i$,

$$\prod_{t=j}^{0}\phi_t(X^i) = \phi_j\circ\cdots\circ\phi_0(X^i) = \begin{cases}0, & k\le j\le\log\bar{N},\\ 2^{j+1}, & 0\le j<k.\end{cases}$$

Since $0 < i < \bar{N}$, we have $\prod_{t=\log\bar{N}-1}^{0}\phi_t(X^i) = 0$. Therefore,

$$\prod_{t=\log\bar{N}-1}^{0}\phi_t(\bar{N}^{-1}v_i\cdot m + y_1X + \ldots + y_{\bar{N}-1}X^{\bar{N}-1}) = v_i\cdot m.$$

This technique reduces the number of $\mathsf{HomAuto}$s needed from $\bar{N}$ to $\log\bar{N}$.

Scheme Switching. The scheme switching algorithm receives a level 2 gadget RLWE ciphertext $\mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\mathbf{m})$, and aims to generate $\mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\bar{\mathbf{sk}}\cdot m)$ to reconstruct the level 2 $\mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(\mathbf{m}) = \left(\mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\bar{\mathbf{sk}}\cdot\mathbf{m}), \mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\mathbf{m})\right)$.

We call the scheme switching algorithm [14] in our ciphertext conversion step and propose Alg. 2 as the final component of our circuit bootstrapping algorithm.

Algorithm 2 Scheme Switching (SS)

```
Input: the set of RLWE ciphertexts $\{\bar{\mathbf{c}}_j\}_{j\in[0,\ell-1]}$ output by alg.1,
Input: a scheme switching key SSK $= \mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}\left(\mathbf{sk}^2\right)$
Output: $\bar{\mathbf{c}} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}\left(\mathbf{m}\right)$.
1: $\mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\mathbf{m}) = (\bar{\mathbf{c}}_0, \bar{\mathbf{c}}_1, ..., \bar{\mathbf{c}}_{\ell-1})$
2: $\bar{\mathbf{c}} \leftarrow \mathsf{SS}_{\mathrm{RLWE}'\rightarrow\mathrm{RGSW}}(\mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\mathbf{m}))$;
3: return $\bar{\mathbf{c}}$
```

3.3 Analysis

The cost of our novel circuit bootstrapping and the original method in Step 1 are similar. Specifically, the MV-FBS requires $O(\ell \underline{n} \bar{N} \log \bar{Q})$ bit size storage and $O(\ell \underline{n} \bar{N} \log \bar{N})$ multiplication complexity. We then focus on comparing in Step 2.

Key Size. The ciphertext conversion step of the TFHE circuit bootstrapping needs two level 1 private key switching keys, resulting in the inclusion of $2(\bar{N}+1)\ell_{\mathsf{ks}}$ level 1 RLWE ciphertexts. In our novel workflow, the ciphertext conversion step needs a level 2 HomTrace key, and a level 2 scheme switching key. It contains $(\log \bar{N}\ell_{\mathsf{trace}} + \ell_{\mathsf{ss}})$ level 2 RLWE ciphertexts. The number of RLWE ciphertexts can be easily translated into bit size. For example, an RLWE ciphertext with dimension N and modulus Q roughly requires $2N \log Q$ bits of space.

Complexity. In the TFHE circuit bootstrapping, the private key switching algorithm involves $4\ell_{\mathsf{ks}}\ell N(\bar{N}+1)$ modular multiplications. In our novel workflow, the HomTrace and scheme switching utilize $\ell \log \bar{N}$ and ℓ level 2 gadget product, respectively. Notice that the gadget product can be converted to NTT/FFT, as each $\odot$ operation requires $(\ell+1)$ NTT/FFTs.

Comparison. We then present the performance metrics of prior studies and conduct a cost analysis of our workflow, as described in Tab. 1. The new circuit bootstrapping workflow significantly reduces the computational and storage overhead of the ciphertext conversion step.

Under the specific parameters given in Sec. 6[7], the key size of the ciphertext conversion step in TFHE circuit bootstrapping is 320.16 MB, accounting for 66.9% of the overall key size. In contrast, our improved ciphertext conversion reduces the key size to 0.6 MB, accounting for only 2.0%. In terms of computational efficiency, the ciphertext conversion step in TFHE circuit bootstrapping

[7] The parameters chosen for the TFHE circuit bootstrapping are listed in Tab. 6. Our new framework utilizes the parameter set CMUX_1 recommended in Tab. 5.

	Complexity	Storage
Step.1	$O(\ell \underline{n} \bar{N} \log \bar{N})$	$O(\ell \underline{n} \bar{N} \log \bar{Q})$
Step.2 TFHE	$O(\ell^2 N \bar{N})$	$O(\ell N \bar{N} \log Q)$
Step.2 Ours	$O(\ell^2 \bar{N} \log^2 \bar{N})$	$O(\ell \bar{N} \log \bar{N} \log \bar{Q})$

Table 1: Computational complexity in the circuit bootstrapping.

accounts for 70.7% of the total runtime, while the runtime of our optimized Step 2 only accounts for 3.0%. For detailed data, please consult Tab. 8 in Sec. 6.3.

4 Automorphism-Based Bootstrapping and MV-FBS

In this section, we incorporate the latest automorphism-based bootstrapping method [24], realizing the AUTO variant of our circuit bootstrapping algorithm.

Approximate Gadget Decomposition. The approximate gadget decomposition (Sec. 2.3.1) is widely applied in TFHE bootstrapping. It can reduce the gadget length ℓ of the decomposition while maintaining a noise level similar to that of the canonical decomposition, thereby enhancing operational efficiency and reducing key size. However, LMKC+ bootstrapping method [24] does not consider this technique in their paper. To give a fair comparison and achieve similar performance between the AUTO variant and the CMUX variant of our circuit bootstrapping, we first introduce the approximate gadget decomposition into the automorphism-based bootstrapping algorithm.

By using the approximate gadget decomposition in LMKC+ bootstrapping, we successfully reduce the gadget length used by the external product and the automorphism from 3 to 2. Our testing in the OpenFHE library yielded significant benefits. Specifically, it led to a 33.4% reduction in the key size and a 25.6% decrease in execution time. Detailed results can be found in C.1.

However, we found that the recently released 1.1.0 version of OpenFHE already includes support for LMKC+ bootstrapping based on approximate gadget decomposition. Even though this development is concurrent with ours, we decided to move this part of the contribution to the supporting materials C.1, not as the main contribution of this paper. We wish to indicate that the remaining improvements are built upon the approximate gadget decomposition technique.

Furthermore, we improve the LMKC+ automorphism-based blind rotation algorithm through a sparse isomorphism, reducing the number of automorphisms, and thereby decreasing the runtime and noise growth. Building on this, we extend the automorphism-based FBS to an MV-FBS without sample extraction that can be employed in our circuit bootstrapping method.

4.1 Improved Automorphism-Based Blind Rotation using Sparse Isomorphism

We observed that the number of automorphisms in the LMKC+ blind rotation algorithm is inversely related to the sparsity of a_i in $\mathbb{Z}_{2N}^*$, $i \in [0, n-1]$. Without optimization, this number is roughly equal to the count of different values of a_i[8], and is always bounded by $\mathsf{min}\{n, N\}$. Our strategy to enhance the scheme involves controlling the sparsity of a_i through the modulus switching step. By using a sparse rounding to enforce a_i to take on the form of $\{kA\}_{k\in[0,\lceil 2N/A\rceil-1]}$, where $A > 2$ controls the sparsity, we can reduce the number of automorphisms from $\mathsf{min}\{n, N\}$ to $\mathsf{min}\{n, \lceil 2N/A\rceil\}$ in one blind rotation.

A drawback of the naive sparse approach is its requirement for more automorphism keys compared with LMKC+. Specifically, LMKC+ introduces a isomorphism $\mathbb{Z}_{2N}^* \cong \mathbb{Z}_{N/2} \otimes \mathbb{Z}_2$ with generators $\{g, -1\}$, reducing the number of keys from N to 2, by mapping every odd a_i to $\pm g^k$. We further improve this technique using a sparse isomorphism: for $N \geq 8$ and $\vartheta \geq 2$, the numbers in the form of $k \cdot 2^{\vartheta} + 1$ in $\mathbb{Z}_{2N}$ form a multiplicative cyclic group with the generator $\{g\}$,

[8] LMKC+ has proposed an optimization to reduce the number of automorphisms by using additional storage. More details can be found in Sec. 4.2.

Algorithm 3 Memory Efficient Blind Rotation for $a_i = k \cdot 2^{\vartheta} + 1(\mathsf{BR}_1)$

```
Input: an LWE encryption $\underline{\mathbf{c}} = (\underline{\mathbf{a}}, \underline{b}) \in \mathrm{LWE}_{\underline{\mathbf{sk}},2N}(m)$,
Input: an initial $\mathsf{acc} = (0, X^{gb} \cdot \mathsf{testP}(X^g))$,
Input: a bootstrapping key $\mathsf{BSK} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(X^{\underline{s}_i})_{i\in[0,n-1]}$,
Input: a automorphism key $\mathsf{ATK}_g = \mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\bar{\mathbf{sk}}(X^g))$.
Output: $\bar{\mathbf{c}} \in \mathrm{RLWE}_{\bar{\mathbf{sk}}}(\mathsf{testP} \cdot X^{\underline{b}+\sum_{i=0}^{n-1}\underline{a}_i\underline{s}_i})$.
1: for ($\ell = \frac{2N}{2^{\vartheta}} - 1$; $\ell > 0$; $\ell = \ell - 1$) do:
2:     for $j \in I_\ell$ do:
3:         $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{BSK}_j$;
4:     end for
5:     $\mathsf{acc} = \mathsf{HomAuto}_g(\mathsf{acc}, \mathsf{ATK})$;
6: end for
7: for $j \in I_0$ do:
8:     $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{BSK}_j$;
9: end for
10: return $\bar{\mathbf{c}} = \mathsf{acc}$
```

which is isomorphic to $\mathbb{Z}_{2N/2^{\vartheta}}$. For example, when $N = 16$ and $\vartheta = 2$, the set $\{1, 5, 9, 13, 17, 21, 25, 29\}$ is a multiplicative group generated by $g = 5 \pmod{32}$.

Thus, every a_i can be rewritten as g^k, where $k \in \mathbb{Z}_{2N/2^{\vartheta}}$. Let $I_0 = \{i : a_i = g^0\}, ..., I_{2N/2^{\vartheta}-1} = \{i : a_i = g^{2N/2^{\vartheta}-1}\}$, we have

$$\sum_i a_i s_i = \left(\sum_{j\in I_0} s_j + g\left(\sum_{j\in I_1} s_j + \cdots + g\left(\sum_{j\in I_{2N/2^{\vartheta}-1}} s_j\right)\right)\right) \pmod{2N}$$

Then we propose a blind rotation algorithm for a_i in the form of $k \cdot 2^{\vartheta} + 1$ that uses a single automorphism key ATK_g as in Alg. 3. The correctness of the algorithm is guaranteed by the homomorphic automorphism, and the correctness proof can be found in Sup. C.2.

4.2 The Number of Automorphisms

Lee et al. [24] introduced a technique to reduce the number of automorphisms. This technique can cap the total number of automorphisms at $t(w-1)/w + N/w$, where t represents the number of non-empty I_ℓ, and w signifies the window size denoting the pre-stored number of automorphism keys. A comprehensive summary of this technique is provided in Sup. C.3. Our sparse isomorphism can reduce the total number of I_ℓ from $2N$ to $N' = 2N/2^{\vartheta}$, thereby reducing the corresponding upper bound of the number of automorphisms to $t(w-1)/w + N'/w$. We then present a computationally efficient blind rotation algorithm for a_i in the form of $k \cdot 2^{\vartheta} + 1$, see Alg. 4.

Determination of Window Size. However, it is difficult to precisely formulate the expected number of the automorphisms n_{auto} from the formula, since

Algorithm 4 Computation Efficient Blind Rotation for $a_i = k \cdot 2^{\vartheta} + 1(\mathsf{BR}_2)$

```
Input: an LWE encryption $\underline{\mathbf{c}} = (\underline{\mathbf{a}}, \underline{b}) \in \mathrm{LWE}_{\underline{\mathbf{sk}},2N}(m)$,
Input: an initial $\mathsf{acc} = (0, X^{gb} \cdot \mathsf{testP}(X^g))$,
Input: a bootstrapping key $\mathsf{BSK} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(X^{\underline{s}_i})_{i\in[0,n-1]}$,
Input: a set of automorphism keys $\mathsf{ATK}_{g^u} = \mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\bar{\mathbf{sk}}(X^{g^u}))_{u\in[1,w]}$.
Output: $\bar{\mathbf{c}} \in \mathrm{RLWE}_{\bar{\mathbf{sk}}}(\mathsf{testP} \cdot X^{\underline{b}+\sum_{i=0}^{n-1} \underline{a}_i \underline{s}_i})$.
1:  $v \leftarrow 0$
2:  for ($\ell = \frac{2N}{2^{\vartheta}} - 1$; $\ell > 0$; $\ell = \ell - 1$) do:
3:      for $j \in I_\ell$ do:
4:          $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{BSK}_j$;
5:      end for
6:      $v \leftarrow v + 1$
7:      if ($I_{\ell-1} \neq \emptyset$ or $v = w$ or $\ell = 1$) then
8:          $\mathsf{acc} = \mathsf{HomAuto}_{g^v}(\mathsf{acc}, \mathsf{ATK})$;
9:          $v \leftarrow 0$
10:     end if
11: end for
12: for $j \in I_0$ do:
13:     $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{BSK}_j$;
14: end for
15: return $\bar{\mathbf{c}} = \mathsf{acc}$
```

the distribution of t is complicated. To obtain a perspective on the average performance, we propose to use the Monte Carlo simulation [22] to estimate the number of automorphisms. We simulate n_{auto} under varying selections of w and the details of our simulation can be found in Sup. C.4. For different N', the chosen w and minimized n_{auto} are listed in Tab. 2.

N'	512	1024	2048
w	8	15	30
n_{auto}	302	368	410

Table 2: Window sizes and the corresponding n_{auto}.

In Sec. 6, considering security, decryption failure probability and computational efficiency, we choose $N = 2048$, $\vartheta = 2$. The window size is reduced from 30 to 15, and n_{auto} is reduced from 410 to 368. As analyzed in LMKC+, our reduction of window size results in smaller key sizes, and the reduction of n_{auto} results in smaller noise increments. The details are shown in C.5.

4.3 Sparse Rounding and Bootstrapping

We then propose the FBS algorithm, wherein BR_2 serves as its core component, see Alg. 5. During the modulus switching (line 1), we use sparse rounding

Algorithm 5 Functional Bootstrapping Algorithm

```
Input: an LWE encryption $\underline{\mathbf{c}} = (\underline{\mathbf{a}}, \underline{b}) \in \mathrm{LWE}_{\underline{\mathbf{sk}},\underline{q}}(m)$,
Input: a LUT represented by $L = [f(0), f(1), \ldots, f(p-1)]$ encoding $f$,
Input: a bootstrapping key $\mathsf{BSK} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(X^{\underline{s}_i})_{i\in[0,n-1]}$,
Input: a set of automorphism keys $\mathsf{ATK}_{g^u} = \mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\bar{\mathbf{sk}}(X^{g^u}))_{u\in[1,w]}$.
Output: $\bar{c} \in \mathrm{LWE}_{\bar{\mathbf{sk}},\bar{Q}}(f(m))$.
1: $b = \lfloor \frac{2N}{\underline{q}} \underline{b} \rceil_{k\cdot 2^\vartheta+1}$ and $a_i = \left\lfloor \frac{2N}{\underline{q}} \underline{a}_i \right\rceil_{k\cdot 2^\vartheta+1} \in \mathbb{Z}_{2N}$ for each $i \in [0, n-1]$
2: $\mathsf{testP} = X^{\frac{N}{2p}} \sum_{j=0}^{p-1} X^{j\frac{N}{p}} \sum_{k=0}^{\frac{N}{p}-1} f(j) X^k$
3: $\mathsf{acc} = (0, X^{gb} \cdot \mathsf{testP}(X^g))$
4: $\mathsf{acc} = \mathsf{BR}_2(\mathbf{c} = (\mathbf{a}, b), \mathsf{acc}, \mathsf{BSK}, \mathsf{ATK})$
5: $\bar{c} = \mathsf{SampleExtract}(\mathsf{acc})$,
6: return $\bar{c}$
```

$\lfloor x \rceil_{k\cdot 2^\vartheta+1}$ instead of $\lfloor x \rceil$, where $\lfloor x \rceil_{k\cdot 2^\vartheta+1}$ outputs the nearest integer in $\mathbb{Z}_{2N}$ of the form $k \cdot 2^\vartheta + 1$ for the input x. This step guarantees that every component a_i in the ciphertext received by the BR_2 is in the form of $k \cdot 2^\vartheta + 1$, allowing the blind rotation to work properly. However, the rounding error is larger than the original modulus switching, which we analyze in Sec. 5.

4.4 Automorphism-Based Multi-Value Functional Bootstrapping

In this section, we propose the first automorphism-based MV-FBS. This algorithm exploits the sparsity of the ciphertext components a_i to compute 2^ϑ look-up tables at a cost of only one BR_2.

As we mentioned in Sec. 3, the method of [12] employs a specialized modulus switching technique to switch each component a_i of the ciphertext to a multiple of 2^ϑ. However, this approach poses a challenge as it disrupts our sparse isomorphism, which necessitates the preservation of the form $k \cdot 2^\vartheta + 1$. To address this issue, we use the following trick:

$$X^{b+\sum_i a_i s_i} = X^{b+\sum_i (a_i+1)s_i - \sum_i s_i}.$$

Specifically, we introduce an auxiliary key $\mathsf{AUX} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(X^{-\sum_{i=0}^{n-1} \underline{s}_i})$. Subsequently, we perform a multiplication of this auxiliary key with acc. Then, we put the updated acc and $a'_i = a_i + 1$ meeting the form $k \cdot 2^\vartheta + 1$ into the BR_2 blind rotation algorithm. Finally, we present our automorphism-based MV-FBS without sample extraction in Alg. 6. Additionally, we provide a standard version of the automorphism-based MV-FBS which may be of independent interest. For further details, please refer to Alg. 8 in the Sup. C.6.

5 Analysis

We conduct a comprehensive analysis encompassing error growth, key size, and computational complexity of our circuit bootstrapping workflow, considering

Algorithm 6 Auto-Based MV-FBS Algorithm without Sample Extraction

```
Input: an LWE encryption $\underline{\mathbf{c}} = (\underline{\mathbf{a}}, \underline{b}) \in \mathrm{LWE}_{\underline{\mathbf{sk}},\underline{q}}(m)$,
Input: a bootstrapping key $\mathsf{BSK} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(X^{\underline{s}_i})_{i\in[0,n-1]}$,
Input: a set of automorphism keys $\mathsf{ATK}_{g^u} = \mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\bar{\mathbf{sk}}(X^{g^u}))_{u\in[1,w]}$,
Input: an auxiliary key $\mathsf{AUX} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(X^{-\sum_{i=0}^{n-1}\underline{s}_i})$.
Output: $\bar{\mathbf{c}}_j \in \mathrm{RLWE}_{\bar{\mathbf{sk}}}(\mathsf{testP}\cdot X^{b+\sum_{i=0}^{n-1}a_is_i-j})_{j\in[0,\ell-1]}$.
1: $b = \left[\left\lfloor \frac{\underline{b}\cdot 2N\cdot 2^{-\vartheta}}{\underline{q}} \right\rceil \cdot 2^{\vartheta}\right]_{2N}$ and $a_i = \left[\left\lfloor \frac{\underline{a}_i\cdot 2N\cdot 2^{-\vartheta}}{\underline{q}} \right\rceil \cdot 2^{\vartheta}\right]_{2N}$
2: $\mathsf{testP} = \frac{1}{2}\sum_{i=0}^{\frac{N}{2^{\vartheta}}-1}\sum_{j=0}^{2^{\vartheta}-1}\bar{N}^{-1}v_iX^{2^{\vartheta}\cdot i+j}$
3: $\mathsf{acc} = (0, X^{gb}\cdot\mathsf{testP}(X^g))$
4: $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{AUX}$
5: $[a_i = \underline{a}_i + 1]_{2N}$
6: $\mathsf{acc} = \mathsf{BR}_2(\mathbf{c} = (\mathbf{a}, b), \mathsf{acc}, \mathsf{BSK}, \mathsf{ATK})$
7: $\mathsf{acc} = \mathsf{acc} + (0, \frac{1}{2}v_0 + \frac{1}{2}v_1X + ... + \frac{1}{2}v_{\ell-1}X^{\ell-1})$
8: for $j = 0$ to $\ell - 1$ do:
9:     $\bar{\mathbf{c}}_j = X^{-j}\cdot\mathsf{acc}$,
10:     return $\bar{\mathbf{c}}_j$
11: end for
```

both the CMUX variant and the AUTO variant. The parameters $\ell, \ell_{\mathsf{ep}}, \ell_{\mathsf{auto}}, \ell_{\mathsf{trace}}, \ell_{\mathsf{ss}}$ and $B, B_{\mathsf{ep}}, B_{\mathsf{auto}}, B_{\mathsf{trace}}, B_{\mathsf{ss}}$ denote the gadget decomposition length and base in the circuit evaluation, external product, automorphism, HomTrace, and scheme switching, respectively. Correspondingly, $\varepsilon, \varepsilon_{\mathsf{ep}}, \varepsilon_{\mathsf{auto}}, \varepsilon_{\mathsf{trace}}, \varepsilon_{\mathsf{ss}}$ denote the decomposition error. The underline symbol $\underline{n}$ and $\underline{q}$ denote the dimension and modulus of the level 0 ciphertext, respectively, while $\bar{N}$ and $\bar{Q}$ represent those of level 2.

5.1 Error Analysis

The error growth of our circuit bootstrapping algorithm can be calculated through a step-by-step analysis. In this section, we present the results, with detailed proofs available in Sup. D. We use σ^2_{in} to denote the maximum ciphertext noise variance before the circuit bootstrapping process, and use $\sigma^2_{\mathsf{ms}}, \sigma^2_{\mathsf{br}}, \sigma^2_{\mathsf{trace}}, \sigma^2_{\mathsf{ss}}$ to signify the error variance after each step of our circuit bootstrapping algorithm:

– Modulus Switching (Sparse Rounding):

$$\sigma^2_{\mathsf{ms}} = \begin{cases} \frac{4\bar{N}^2\sigma^2_{\mathsf{in}}}{\underline{q}^2} + (\frac{\underline{n}}{2}+1)\cdot(\frac{2^{2\vartheta}}{12} - \frac{\bar{N}^2}{3\underline{q}^2}) + \frac{\underline{n}\bar{N}^2}{4\underline{q}^2}, \text{CMUX variant;} \\ \frac{4\bar{N}^2\sigma^2_{\mathsf{in}}}{\underline{q}^2} + (\underline{n}\sigma^2+1)\cdot(\frac{2^{2\vartheta}}{12} + \frac{2\bar{N}^2}{3\underline{q}^2}) - \frac{\bar{N}^2}{\underline{q}^2}, \text{AUTO variant;} \end{cases}$$

– Blind Rotation:

$$\sigma^2_{\mathsf{br}} = \begin{cases} 2\underline{n}\times\left(\frac{1}{6}\bar{N}\ell_{\mathsf{ep}}B^2_{\mathsf{ep}}\times\sigma^2 + \frac{1}{3}(\bar{N}+1)\varepsilon^2_{\mathsf{ep}}\right), \text{CMUX variant;} \\ \underline{n}\times\left(\frac{1}{6}\bar{N}\ell_{\mathsf{ep}}B^2_{\mathsf{ep}}\times\sigma^2 + \frac{1}{3}(\bar{N}+1)\varepsilon^2_{\mathsf{ep}}\right) \\ +n_{\mathsf{auto}}\times\left(\frac{1}{12}\bar{N}\ell_{\mathsf{auto}}B^2_{\mathsf{auto}}\times\sigma^2 + \frac{1}{3}\times\frac{\bar{N}}{2}\varepsilon^2_{\mathsf{auto}}\right), \text{AUTO variant;} \end{cases}$$

– HomTrace:

$$\sigma^2_{\mathsf{trace}} = \sigma^2_{\mathsf{br}} + \frac{1}{3}\left(\bar{N}^2 - 1\right) \cdot \left(\frac{1}{12}\bar{N}\ell_{\mathsf{trace}} B^2_{\mathsf{trace}} \times \sigma^2 + \frac{1}{3} \times \frac{\bar{N}}{2}\varepsilon^2_{\mathsf{trace}}\right)$$

– Scheme Switching:

$$\sigma^2_{\mathsf{ss}} = \sigma^2_{\mathsf{trace}} + \left(\frac{1}{12}\bar{N}\ell_{\mathsf{ss}} B^2_{\mathsf{ss}} \times \sigma^2 + \frac{1}{3} \times \frac{\bar{N}^2}{4}\varepsilon^2_{\mathsf{ss}}\right)$$

5.1.1 Max Depth.

σ^2_{ss} depicts the noise variance of the RGSW ciphertext output by the circuit bootstrapping algorithm, which subsequently serves as input for the next circuit operations, such as LUT. We use *Noise Add* to define the additional noise in each layer of CMUX gates during circuit evaluations, and *MAX Depth* is used to define the maximum circuit depth it can support before the next circuit bootstrapping. Then we have the following:

– RGSW Output $= \sigma^2_{\mathsf{ss}}$

– Noise Add $= \frac{1}{12}\bar{N}\ell B^2 \times \sigma^2_{\mathsf{ss}} + \frac{1}{6}(\bar{N}+1)\varepsilon^2$

– MAX Depth $\approx \frac{\sigma^2_{\mathsf{in}} - \sigma^2_{\mathsf{ms}_2}}{q^2} / \frac{\text{Noise Add}}{Q^2}$

The ciphertext output from circuit evaluation needs to pre-switch from level 2 to level 0 before serving as the input for the next circuit bootstrapping. Therefore, when calculating the maximum depth, we must also account for the error introduced during the switching process. There is a modulus switching in the final step (ms_2, which is independent with sparse rounding), and nearly all errors produced in previous steps can be eliminated during it. The error generated by modulus switching itself can be calculated using the following formula:

$$\sigma^2_{\mathsf{ms}_2} = \frac{\|\mathbf{s}\|^2 + 1}{12},$$

where $\sigma^2_{\mathsf{ms}_2}$ denotes the error variance, $\mathbf{s}$ is the level 0 secret key. We have $\|\mathbf{s}\| \leq \sqrt{\underline{n}/2}$ for binary secret key, and $\|\mathbf{s}\| = \sqrt{\underline{n}\sigma^2}$ for Gaussian secret key [24].

5.1.2 Failure Probability.

The ciphertext with the maximum noise throughout the LHE mode evaluation is after the sparse rounding step and prior to entering the next blind rotation. At this point, the error variance is σ^2_{ms}, the ciphertext modulus is $2\bar{N}$, plaintext space is 2, thus the failure probability can be calculated by $1 - \mathsf{erf}\left(\frac{2\bar{N}}{4\sqrt{2}\sigma_{\mathsf{ms}}}\right)$, where erf represents the Gaussian error function.

5.2 Key Size

The circuit bootstrapping key can be divided into a multi-value functional bootstrapping key, a HomTrace key, and a scheme switching key. It can be converted into the number of RLWE ciphertexts as follows:

– Step 1. Multi-Value Functional Bootstrapping:

$$\begin{cases} 2\underline{n}\ell_{\mathsf{ep}}, \text{CMUX variant;} \\ \underline{n}\ell_{\mathsf{ep}} + (\underline{n} + w + 2)\ell_{\mathsf{auto}}, \text{AUTO variant;} \end{cases}$$

– Step 2. HomTrace and Scheme Switching: $\log \bar{N}\ell_{\mathsf{trace}} + \ell_{\mathsf{ss}}$

The total number of RLWE ciphertexts can be easily translated into bit size. For example, a level 2 RLWE ciphertext with dimension $\bar{N}$ and modulus $\bar{Q}$ roughly requires $2\bar{N}\log\bar{Q}$ bits of space.

5.3 Computational Complexity.

We measure the computational complexity of our circuit bootstrapping by the number of NTTs. Notice that the gadget product can be converted to NTT as each $\odot$ needs $(\ell + 1)$ NTTs, and each external product $\boxdot$ requires 2 gadget product $\odot$. HomTrace requires $\log\bar{N}$ times $\odot$ and Scheme Switching can be performed with 1 $\odot$. The number of NTTs can be computed as follows:

– Step 1. Multi-Value Functional Bootstrapping:

$$\begin{cases} 2\underline{n}(\ell_{\mathsf{ep}} + 1), \text{CMUX variant;} \\ 2\underline{n}(\ell_{\mathsf{ep}} + 1) + n_{\mathsf{auto}}(\ell_{\mathsf{auto}} + 1), \text{AUTO variant;} \end{cases}$$

– Step 2. HomTrace and Scheme Switching: $\log\bar{N}(\ell_{\mathsf{trace}} + 1) + \ell_{\mathsf{ss}} + 1$

6 Parameter Selection and Implementation

6.1 Parameters for Security.

The security of the FHEW-like cryptosystem is based on the (ring) learning with errors problem [34,28]. The security level is determined by the key distribution, ciphertext dimension, ciphertext modulus, and initial noise. We list these parameters both for our CMUX variant circuit bootstrapping algorithm and AUTO variant in Tab. 3, such that the ciphertexts in each level achieve over 128-bit security. The underline symbol $\underline{n}$ and $\underline{q}$ denote the dimension and modulus of the level 0 ciphertext, respectively, while $\bar{N}$ and $\bar{Q}$ represent those of level 2. The security parameter λ is tested by Lattice estimator[9].

Method	λ	Lv0 key	Lv2 key	$\underline{n}$	$\underline{q}$	$\bar{N}$	$\bar{Q}$	σ
CMUX variant	128	Binary	Binary	571	2^{10}	2^{11}	$\approx 2^{54}$	3.2
AUTO variant	128	$\sigma = 3.2$	Binary	458	2^{10}	2^{11}	$\approx 2^{54}$	3.2

Table 3: Security and parameters of our circuit bootstrapping.

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

6.2 Parameters for Noise Management.

FHE algorithms manage the noise growth through gadget decomposition, and by controlling the decomposition length and base. In the FHE evaluation mode, the noise level determines the failure probability of the functional bootstrapping algorithm. Things are different in the LHE evaluation mode. As we pre-define σ_{in}^2 to denote the maximum ciphertext noise variance prior to circuit bootstrapping, the decryption failure probability Prob is determined by σ_{in} and the sparse parameter ϑ. We present the Prob of our circuit bootstrapping algorithm under different choices of parameters in Tab. 4. Finally, we choose $\sigma_{\mathsf{in}}^2 = 2^{10}, \vartheta = 2$ to achieve a lower decryption failure probability, although a larger ϑ can help to reduce the number of automorphisms n_{auto}.

Terms	$\sigma_{\mathsf{in}}^2 = 2^{10}, \vartheta = 2$	$\sigma_{\mathsf{in}}^2 = 2^{6}, \vartheta = 3$
Prob	$2^{-49.5}$	$2^{-31.2}$
n_{auto}	368	302

Table 4: The decryption failure probability and the number of automorphisms in the average case under different choices of parameter sets. The failure probability is chosen as the higher one among the CMUX variant and AUTO variant.

Noise control plays a crucial role in determining the supported depth of circuit evaluations. We then discuss the specifics of gadget decomposition length and base at each step. Depending on the supported circuit depths, we present several recommended parameter sets as listed in Tab. 5.

Sets	ℓ_{ep}	B_{ep}	ℓ_{auto}	B_{auto}	ℓ_{trace}	B_{trace}	ℓ_{ss}	B_{ss}	ℓ	B	Max Depth	Key Size	# of NTTs
AUTO_1	1	2^{26}	1	2^{26}	2	2^{17}	1	2^{28}	2	2^{6}	7	25.0 MB	2510
CMUX_1	1	2^{26}	–	–	2	2^{17}	1	2^{28}	2	2^{5}	8	30.7 MB	2354
AUTO_2	2	2^{17}	2	2^{17}	2	2^{17}	1	2^{28}	2	2^{6}	73	49.4 MB	3730
CMUX_2	2	2^{17}	–	–	2	2^{17}	1	2^{28}	2	2^{6}	115	60.8 MB	3496
AUTO_3	2	2^{17}	2	2^{17}	2	2^{17}	2	2^{19}	2	2^{7}	149	49.7 MB	3732
CMUX_3	2	2^{17}	–	–	2	2^{17}	2	2^{19}	2	2^{7}	236	60.9 MB	3498
AUTO_4	2	2^{17}	2	2^{17}	3	2^{13}	2	2^{19}	2	2^{8}	$7,900+$	49.7 MB	3754
CMUX_4	2	2^{17}	–	–	3	2^{13}	2	2^{19}	2	2^{8}	$10,900+$	61.1 MB	3520
AUTO_5	2	2^{17}	2	2^{17}	4	2^{11}	2	2^{19}	2	2^{8}	$16,800+$	50.0 MB	3776
CMUX_5	2	2^{17}	–	–	4	2^{11}	2	2^{19}	2	2^{8}	$20,200+$	61.4 MB	3542

Table 5: The recommended parameter sets for noise management. For each parameter set, we have listed the corresponding max supported circuit depth, circuit bootstrapping key size, and the number of needed NTT/FFTs.

6.3 Implementation Results and Comparison

The TFHE circuit bootstrapping uses three levels of parameters, which we have set to be consistent with the TFHEpp library [29], as detailed in Tab. 6. It is important to note that these definitions, following Chillotti [9,10,11], employ different notions compared with ours. Specifically, TFHE uses the real torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ and $\mathbb{T}_N[X] = \mathcal{R}/\mathbb{Z}$ to describe the message and ciphertext spaces, and implements $\mathbb{T}$ by $\mathbb{Z}_q$ with $q = 2^{32}$ or $q = 2^{64}$.

Level	λ	Key	n	Torus	σ	ℓ_{ep}	B_{ep}	ℓ_{ks}	B_{ks}	ℓ	B
2	153	binary	$\bar{N} = 2048$	Int64	2^{-44}	4	2^9	–	–	–	–
1	129	binary	$N = 1024$	Int32	2^{-25}	–	–	10	2^3	3	2^6
0	128	binary	$\underline{n} = 635$	Int32	2^{-15}	–	–	–	–	–	–

Table 6: Security and parameters of the TFHE circuit bootstrapping [29].

It is important to highlight that in the private key switching step of the TFHE circuit bootstrapping, there exists a pre-computed method that trades storage for computational efficiency [16]. Specifically, given the fixed gadget decomposition base B_{ks}, the number of possible results of $a_{i,j} \cdot \mathrm{RLWE}_{\mathsf{sk}'}(v_j \cdot sk_i)$ in the $\mathsf{PrivateKS}$ computation is limited, where $a_{i,j}$ represents the gadget decomposition of a_i. Consequently, all possible outcomes can be pre-stored as a new switching key. This method allows modular multiplication to be substituted with modular addition, leading to enhanced efficiency and noise management within the private key switching algorithm. However, it also results in a significant increase in key size.

In the following analysis, we give the performances of both the original TFHE circuit bootstrapping method and the pre-computed variant $\mathrm{TFHE}_{\mathsf{precom}}$ as two different control groups. To begin with, we calculate the maximum supported circuit depth for these two control groups, as presented in Tab. 7.

Method	σ^2_{in}	RGSW Output	Noise Add (per CMUX)	Max Depth
TFHE	2^{-10}	$2^{-33.3}$	$2^{-13.3}$	9
$\mathrm{TFHE}_{\mathsf{precom}}$	2^{-10}	$2^{-38.2}$	$2^{-18.2}$	294

Table 7: Noises and the maximal CMUX depth.

Then, we select the recommended parameter set $\mathsf{CMUX/AUTO}_1$ in our method for comparison with the TFHE method, considering their similar maximum supported circuit depths. In comparison with the $\mathrm{TFHE}_{\mathsf{precom}}$ method, we opt for $\mathsf{CMUX/AUTO}_5$. Although $\mathsf{CMUX/AUTO}_4$ would suffice for this comparison, our observations have revealed that the computational costs of sets 4 and 5 are similar. However, the supported circuit depth of set 5 significantly exceeds that of set 4, leading us to choose $\mathsf{CMUX}/\mathsf{AUTO}_5$ for our implementation.

6.3.1 Implementation Results. We implement our circuit bootstrapping method in the OpenFHE library [1]. To enable a direct performance comparison between TFHE circuit bootstrapping and ours, we also implement the unit test for TFHE circuit bootstrapping in OpenFHE, replacing all FFT operations with NTTs. This eliminates the running time difference caused by library optimization and the efficiency difference between FFT and NTT. The evaluation environment is a PC with 11th Gen Intel(R) Core(TM) i5-11500 @ 2.70GHz and 32 GB of RAM, running Ubuntu 22.04.2 LTS. Detailed runtime results are presented in Tab. 8.

Methods	Sets	Key Size (in MB)			Run Time (in ms)		
		Step.1	Step.2	Total	Step.1	Step.2	Total
TFHE	TFHEpp	158.75	320.16	478.91	256.4	620.6	877
Ours	AUTO_1	24.41	0.60	**25 (5.2%)**	96.96	2.64	**99.6(8.8×)**
	CMUX_1	30.11	0.60	**30.71 (6.4%)**	85.94	2.63	**88.57(9.9×)**
$\text{TFHE}_{\mathsf{precom}}$	TFHEpp	158.75	2561.28	2720.03	256.4	227.68	484.08
Ours	AUTO_5	48.83	1.21	**50.04 (1.8%)**	152.31	4.59	**156.9(3.1×)**
	CMUX_5	60.22	1.21	**61.43 (2.2%)**	135.77	4.59	**140.36(3.5×)**

Table 8: Our proposed circuit bootstrapping performance compared with the state-of-the-art method.

Remark 2. There are three open-sourced TFHE circuit bootstrapping implementations using FFT and advanced vector extensions (AVX). TFHElib[10] achieves a runtime of 137 ms for 110 bit security using AVX-2, while TFHEpp [29] reduces the run time under 128 bit security using $\mathsf{PBSmanyLUT}$. MOSFHET [21] further reduces FFT runtime using AVX-512. These implementations' latency is shorter than that of our experiment above, but it is due to library optimization and the use of AVX. As mentioned in [30], AVX-2 can accelerate FFT operations by up to 8×. We have not found an open-source FHE library that supports AVX-accelerated NTT.

Therefore, we also provide an optimized implementation of our scheme using AVX-512 as part of our work. Following extensive testing and comparison, our experimental results are superior to all existing circuit bootstrapping implementations, as indicated in Tab. 9. In the time comparison of step 1, it is evident

[10] https://github.com/tfhe/experimental-tfhe

Methods	Library	Run Time (in ms)		
		Step.1	Step.2	Total
Ours	Ours	42.98	1.46	**44.4**
$\text{TFHE}_{\mathsf{precom}}$	TFHEpp[11]	25.8	32.3	58.1
	MOSFHET[12]	21.7	181.2	202.9

Table 9: Our AVX-accelerated implementation compared with the state-of-the-art implementations. It is worth mentioning that the step.2 test results of MOSFHET are significantly slower than TFHEpp. This is due to the fact that in this library, the private key switching, which should be executed at level 1, is executed at level 2.

that, due to a lack of sufficient optimization (such as assembly language), the efficiency of our basic operation unit (NTT/FFT) is slower than that of TFHEpp and MOSFHET. However, due to the superiority of our method, our circuit bootstrapping implementation still leads among libraries. Further optimization of our library will be included in our future works.

7 Application

Homomorphic Evaluation of AES. In transciphering scenarios[13], the evaluation of the advanced encryption standard (AES) circuit through FHE has been explored using a variety of schemes. For instance, Cheon et al. and Coron et al. both implemented the AES circuit using the vDGHV[37] scheme[7,13]. Doröz et al. suggested an AES evaluation using LTV[27] scheme[15]. However, these evaluation methods exhibit extremely high latency. Gentry et al. proposed an AES evaluation method using the BGV scheme [19]. Although this method still has a delay of 18 minutes, the amortized time can reach 5.8 seconds, since BGV supports packing messages into plaintext slots for SIMD (Single Instruction Multiple Data) computation.

Actually, the FHEW-like scheme proves to be a suitable choice to decrease the delay of AES homomorphic evaluation. The AES circuit which we detailed in Sup. E.2 mainly consists of the following four operations: AddRoundKey, SubBytes, ShiftRows, and MixColumns. Under such bit-wise encryption schemes, most of them are straightforward to implement: AddRoundKey can be calculated through XOR gates; ShiftRows only requires a reordering of the ciphertext; MixColumns can also be realized through a combination of XOR and shifting[15]. The computationally expensive operation is primarily the SubBytes operation, also widely known as the S-box evaluation.

[11] https://github.com/virtualsecureplatform/TFHEpp
[12] https://github.com/antoniocgj/MOSFHET
[13] We detailed this application scenario in Sup. E.1

Scheme	Evaluation Mode	Hardware	Run Time
BGV	–	i5-3320M, 4GB RAM	1,080s[19] [14]
FHEW-like	FHE	i7-12700H, 64GB RAM	270s[36]
	$\mathsf{LHE}_{\mathsf{our}}$	i5-11500, 32GB RAM	**26.2s**

Table 10: Our AES evaluation time compared with the state-of-the-art works.

AES evaluation under FHE mode using functional bootstrapping: In FHE mode, each 8×8 S-box evaluation requires $16\times$ 255=4,080 gate. The homomorphic evaluation of AES needs 16×10=160 S-boxes, with a total of over 600,000 gate bootstrappings. Trama et al. designed a new evaluation method for AES using functional bootstrapping with a plaintext space of 4 bits[36]. Through further optimization utilizing a tree-based look-up-table approach, the complete AES evaluation requires 4,244 functional bootstrappings. This method currently offers the best latency for homomorphic AES evaluation, achieving a run time of 270 seconds on a single thread (TFHE-lib, AVX-2, Assembler Language, Intel(R) Core(TM) i7-12700H CPU with 64 GB RAM). The detailed evaluation methods are shown in Sup. E.3.

AES evaluation under LHE mode using our novel circuit bootstrapping: In LHE mode, each S-box evaluation requires an 8-level CMUX circuit and 8 circuit bootstrappings. Other operations can be performed almost for free. Therefore, the evaluation cost of AES primarily stems from 8×160=1,280 circuit bootstrappings, which take up 98% of the overall runtime. Given that the S-box evaluation has a maximum circuit depth of 8, we can employ a smaller parameter set CMUX_1 for the circuit bootstrapping algorithm. Under this set of parameters, the run time of circuit bootstrapping is 19.2(step 1)+0.83(step 2)=20 ms, and the overall latency of homomorphic evaluation of AES is 26.2s, see Tab. 10. Our result achieves $10.3\times$ better compared to FHE mode evaluation using functional bootstrapping [36]. This is remarkable because we do not take into account the hardware gap and assembly language, which could further improve our performance. It is also important to note that, the speedup ratio of AES evaluation using our proposed circuit bootstrapping method versus the traditional one should be consistent with our CBS speedup. This is supported by the experiments, which show that 98% of the time overhead of AES evaluation comes from circuit bootstrapping.

[14] Gentry also proposed a homomorphic AES evaluation approach that does not employ the BGV bootstrapping algorithm, with a latency of 240s. As the homomorphic ciphertext derived from this method is noisy and does not support further operations, it is not considered in the transciphering scenario. It is worth mentioning that even compared to this method, we have achieved a $9\times$ performance improvement.

8 Conclusion

In this paper, we propose a faster and smaller circuit bootstrapping work flow. Our approach significantly reduces the latency by approximately 90% compared with the state-of-the-art method, while maintaining the key size within 50 MB. This storage advancement also opens up the possibility for further research into hardware acceleration of circuit bootstrapping. We anticipate our work as a stepping stone towards the practical application of the TFHE LHE mode.

References

1. 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)
2. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34. pp. 297–314. Springer (2014)
3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
5. 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)
6. 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)
7. Cheon, J.H., Coron, J.S., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: Batch fully homomorphic encryption over the integers. In: Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32. pp. 315–335. Springer (2013)
8. 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)
9. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. ADVANCES IN CRYPTOLOGY-ASIACRYPT 2016, PT I **10031**, 3–33 (2016)
10. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. 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. pp. 377–408. Springer (2017)
11. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)

12. 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)
13. Coron, J.S., Lepoint, T., Tibouchi, M.: Scale-invariant fully homomorphic encryption over the integers. In: Public-Key Cryptography–PKC 2014: 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings 17. pp. 311–328. Springer (2014)
14. De Micheli, G., Kim, D., Micciancio, D., Suhl, A.: Faster amortized fhew bootstrapping using ring automorphisms. Cryptology ePrint Archive (2023)
15. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic aes evaluation using the modified ltv scheme. Designs, Codes and Cryptography **80**, 333–358 (2016)
16. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: 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 34. pp. 617–640. Springer (2015)
17. Gama, N., Izabachene, M., Nguyen, P.Q., Xie, X.: Structural lattice reduction: generalized worst-case to average-case reductions and homomorphic cryptosystems. In: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. pp. 528–558. Springer (2016)
18. Gentry, C.: A fully homomorphic encryption scheme. Stanford university (2009)
19. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
20. 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)
21. 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)
22. James, F.: Monte carlo theory and practice. Reports on progress in Physics **43**(9), 1145 (1980)
23. Kim, A., Lee, Y., Deryabin, M., Eom, J., Choi, R.: Lfhe: Fully homomorphic encryption with bootstrapping key size less than a megabyte. Cryptology ePrint Archive (2023)
24. Lee, Y., Micciancio, D., Kim, A., Choi, R., Deryabin, M., Eom, J., Yoo, D.: Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 227–256. Springer (2023)
25. Liu, F.H., Wang, H.: Batch bootstrapping i: a new framework for simd bootstrapping in polynomial modulus. In: Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III. pp. 321–352. Springer (2023)
26. Liu, F.H., Wang, H.: Batch bootstrapping ii: bootstrapping in polynomial modulus only requires o˜(1) fhe multiplications in amortization. In: Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III. pp. 353–384. Springer (2023)

27. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the forty-fourth annual ACM symposium on Theory of computing. pp. 1219–1234 (2012)
28. 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. pp. 1–23. Springer, Heidelberg (2010)
29. Matsuoka, K., Banno, R., Matsumoto, N., Sato, T., Bian, S.: Virtual secure platform: A five-stage pipeline processor over tfhe. In: USENIX Security Symposium. pp. 4007–4024 (2021)
30. 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)
31. Muttaqin, K., Rahmadoni, J.: Analysis and design of file security system aes (advanced encryption standard) cryptography based. Journal of Applied Engineering and Technological Science (JAETS) **1**(2), 113–123 (2020)
32. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop. pp. 113–124 (2011)
33. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
34. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6), 34:1–34:40 (2009)
35. Rijmen, V., Daemen, J.: Advanced encryption standard. Proceedings of federal information processing standards publications, national institute of standards and technology **19**, 22 (2001)
36. Trama, D., Clet, P.E., Boudguiga, A., Sirdey, R.: At last! a homomorphic aes evaluation in less than 30 seconds by means of tfhe. Cryptology ePrint Archive (2023)
37. Van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. 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. 24–43. Springer (2010)

Supplementary Material

A Performing Circuit Evaluation in Level 2

As mentioned in Sec.1.2, our LHE mode evaluate circuits at level 2. This results in an slight increase in circuit computation latency. To determine the exact performance for large circuits, we run circuit evaluation experiments under the TFHEpp parameter set and CMUX_5 parameter set. The results show that our approach has a 1.5× latency per CMUX gate compared with the TFHE solution in OpenFHE library. However, our approach maintains better performance as the depth of circuit operations approaches 4,000, see Fig.4 (In fact, the TFHE method under the TFHEpp parameter set does not support such a large circuit depth). It is important to note that we are comparing our approach to the pre-computed variant of TFHE circuit bootstrapping, which has a key size nearly 46 times larger than ours. Furthermore, while TFHE has reached its limit for the depth of CMUX circuits, our approach can continue to compute CMUX gates to a depth of 20,000+.

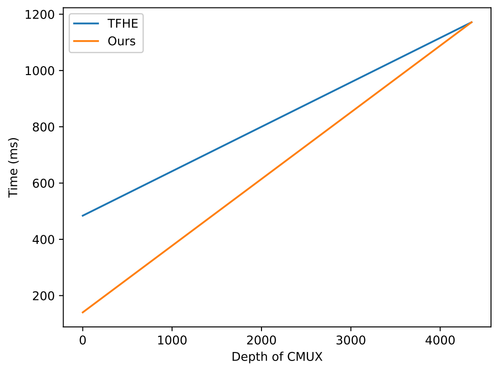


Fig. 4: LHE mode run time

B LMKC+ Basic Blind Rotation Algorithm for Odd a_i

We give a detailed LMKC+ blind rotation algorithm for Odd a_i in Alg. 7.

C Automorphism-Based Bootstrapping and MV-FBS

C.1 Automorphism-Based Gate Bootstrapping with Approximate Gadget Decomposition

We provide a performance comparison of the bootstrapping algorithm based on approximate decomposition and canonical decomposition, see Tab. 11. LMKC+

Algorithm 7 LMKC+ Blind Rotation Algorithm for Odd a_i.

```
Input: an LWE encryption $\underline{\mathbf{c}} = (\underline{\mathbf{a}}, \underline{b}) \in \mathrm{LWE}_{\underline{\mathbf{sk}},2N}(m)$,
Input: an initial $\mathsf{acc} = (0, X^{-gb} \cdot \mathsf{testP}(X^{-g}))$,
Input: a bootstrapping key $\mathsf{BSK} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(X^{\underline{s}_i})_{i \in [0,n-1]}$,
Input: two automorphism keys $\mathsf{ATK}_g$ and $\mathsf{ATK}_{-g}$.
Output: $\bar{\mathbf{c}} \in \mathrm{RLWE}_{\bar{\mathbf{sk}}}(\mathsf{testP} \cdot X^{\underline{b}+\sum_{i=0}^{n-1} \underline{a}_i \underline{s}_i})$.
1: for ($\ell = N/2 - 1$; $\ell > 0$; $\ell = \ell - 1$) do:
2:     for $j \in I_\ell^-$ do:
3:         $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{BSK}_j$;
4:     end for
5:     $\mathsf{acc} = \mathsf{HomAuto}_g(\mathsf{acc}, \mathsf{ATK}_g)$;
6: end for
7: for $j \in I_0^-$ do:
8:     $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{BSK}_j$;
9: end for
10: $\mathsf{acc} = \mathsf{HomAuto}_{-g}(\mathsf{acc}, \mathsf{ATK}_{-g})$;
11: for ($\ell = N/2 - 1$; $\ell > 0$; $\ell = \ell - 1$) do:
12:     for $j \in I_\ell^+$ do:
13:         $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{BSK}_j$;
14:     end for
15:     $\mathsf{acc} = \mathsf{HomAuto}_g(\mathsf{acc}, \mathsf{ATK}_g)$;
16: end for
17: for $j \in I_0^+$ do:
18:     $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{BSK}_j$;
19: end for
20: return $\bar{\mathbf{c}} = \mathsf{acc}$
```

serves as the control group, utilizing canonical decomposition. GINX uses the CMUX gate based on approximate decomposition to calculate blind rotation, while Ours uses the automorphism based on approximate decomposition. ℓ_{ep} and ℓ_{auto} denote the gadget decomposition length in the external product and automorphism, respectively.

Method	λ	key	n	q	N	Q	σ	ℓ_{ep}	ℓ_{auto}	Key Size	Run Time
LMKC+	128	$\sigma = 3.2$	458	2^{10}	2^{10}	$\approx 2^{28}$	3.2	3	3	19.03 MB[15]	99.52ms
GINX	128	Binary	571	2^{10}	2^{10}	$\approx 2^{25}$	3.2	2	–	13.94 MB	65.5 ms
Ours	128	$\sigma = 3.2$	458	2^{10}	2^{10}	$\approx 2^{28}$	3.2	2	2	12.67 MB	73.98ms

Table 11: Security and parameters of our circuit bootstrapping.

By using the approximate gadget decomposition, we reduce the gadget length from 3 to 2. As a result, it can diminish the key size of the algorithm by 33.4% and the execution time by 25.6%.

C.2 Correctness of Memory Efficiency Blind Rotation Algorithm

We give the correctness proof of our memory efficiency blind rotation algorithm (Alg. 3) proposed in section 4.1.

Proof. The initial acc can be considered as a trivial ciphertext $\mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}\,(X^{gb}\cdot \mathsf{testP})$. For $\ell = \frac{2N}{2^{\vartheta}} - 1$, after executing the steps in line 3, acc is multiplied together with $\mathsf{BSK}_{j,j\in I_{2N/2^{\vartheta}-1}}$. Then homomorphically evaluating $\psi_g : X \to X^g$ in line 5, we obtain

$$\mathsf{acc} = \mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}\left((X^{g^2 b}\cdot \mathsf{testP}(X^{g^2}))\cdot X^{g\cdot\sum_{j\in I_{2N/2^{\vartheta}-1}} s_j}\right)$$

Repeating the above process and completing the for loop with respect to l, we get the output

$$\bar{\mathbf{c}} = \mathsf{acc} = \mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}\left((X^{g^{2N/2^{\vartheta}} b}\cdot \mathsf{testP}(X^{g^{2N/2^{\vartheta}}}))\cdot X^{\sum_i a_i s_i}\right)$$

Since $g^{2N/2^{\vartheta}} = 1$, we have

$$\bar{\mathbf{c}} = \mathrm{RLWE}_{\bar{\mathbf{sk}},\bar{Q}}\left(\mathsf{testP}\cdot X^{b+\sum_i a_i s_i}\right).$$

C.3 Computation Optimization to Reduce Automorphisms

Lee et al. [24] introduced a technique to reduce the number of automorphisms. The basic idea is that when some I_ℓ[16] are empty, there is no external product $\boxdot$ performed, and the automorphisms can be composed into a single one. However, to handle all possible situations, the number of automorphism keys needed is $N-1$. Therefore, techniques are needed to represent all possible automorphisms with a limited number w of automorphisms, where w is called the window size.

Suppose there are t non-empty sets I_ℓ, with the corresponding composed automorphism being g^{v_ℓ}, where $v_\ell \geq 1$ is the index accumulated from I_ℓ to the next non-empty set. In [24], v_ℓ was represented by $v_\ell = v'_\ell + v''_\ell \cdot w$, where $0 \leq v'_\ell < w$. By storing $\{\mathsf{ATK}_{g^u}\}_{u\in[1,w]}$,

$$g^{v_\ell} = g^{v'_\ell}\underbrace{g^w \cdots g^w}_{v''_\ell \text{ times}},$$

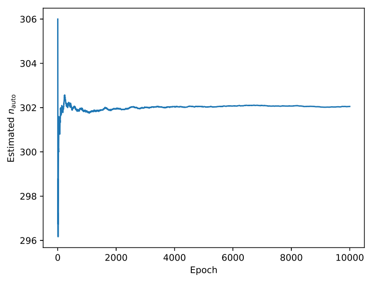
can be represented by $1 + v''_\ell$ known automorphisms. The total number of automorphisms is then bounded by $t(w-1)/w + N/w$, as given in Lee et al. [24].

[16] $I_0 = \{i : a_i = g^0\}, ..., I_{2N/2^{\vartheta}-1} = \{i : a_i = g^{2N/2^{\vartheta}-1}\}$, see section 4.1 for more details

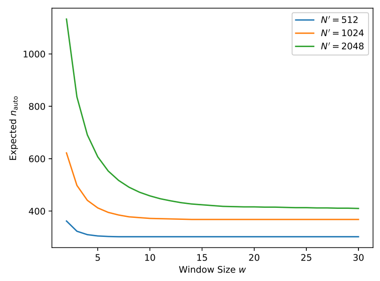
C.4 Monte Carlo Simulation

The factors that affect the number of automorphisms are the indices of all non-empty sets I_ℓ and their corresponding v_ℓ. The simulation can be reduced to a stochastic process where we randomly place n balls into N' bins. Since α_i are uniformly distributed, the probability of a ball going into each container is the same. For each epoch, the number of automorphisms needed corresponding to a certain window size w is directly computable.

The convergence of the Monte Carlo process and the changing curve of the number of automorphisms with respect to w is shown in Fig. 5. Since w only affects a small proportion of the key size, we pick a w large enough to minimize the number of automorphisms.



(a) Convergence of the Process



(b) Expected n_{auto}

Fig. 5: The Monte Carlo Process

C.5 LMKC Comparison

Error Analysis. When analyzing errors in FHEW-like bootstrapping algorithms, one can intuitively compare the values of $\sigma^2_{\mathsf{FBS}}/\sigma^2_{\odot}$. Tab.12 lists the error ratios of our proposed bootstrapping Alg. 5 in the worst case, and compare it with LMKC+ and GINX. It demonstrates that when w is large, our scheme has the lowest error growth, even in the worst case.

Error Ratio	GINX	LMKC+	Ours
$\sigma^2_{\mathsf{FBS}}/\sigma^2_{\odot}$	$4n$	$2n + \frac{w-1}{w}n + \frac{N}{w}$	$2n + \frac{w-1}{w}\mathsf{min}\{n, \frac{2N}{2^\vartheta}\} + \frac{2N}{2^\vartheta w}$

Table 12: Error ratio in the worst case.

Key Size and Computational Complexity. The key size and computational complexity of the LMKC+ bootstrapping method and ours are listed in Tab. 13. Our scheme has a slight advantage in key size due to the elimination of the need to store ATK_{-g} and the ability to choose a smaller w, as demonstrated by the experimental results in Sec. 4.2. In terms of computational complexity, both schemes require the same number of external products, but we use fewer automorphisms. We provide expressions for the number of automorphisms required by the LMKC+ method and our method under both worst-case and average-case.

Methods	# keys (in RLWE$'$)	# $\mathsf{HomAutos}$ (worst case)	# $\mathsf{HomAutos}$ (average case)	# $\boxdot$
LMKC+	$2n+w+1$	$\frac{w-1}{w}n+\frac{N}{w}$	$N-(\frac{w-1}{w}e^{-n/N})N$	n
Ours	$2n+w$	$\frac{w-1}{w}\mathsf{min}\{n, \frac{2N}{2^{\vartheta}}\}+\frac{2N}{2^{\vartheta}w}$	$\frac{2N}{2^{\vartheta}}-(\frac{w-1}{w}e^{-n2^{\vartheta}/2N})\frac{2N}{2^{\vartheta}}$	n

Table 13: Key size and computation complexity.

C.6 Automorphism-Based Muti-Value Functional Bootstrapping

In this section, we provide a standard version of the automorphism-based MV-FBS algorithm in Alg.8. This algorithm may have independent interests.

Algorithm 8 Automorphism-Based MV-FBS Algorithm

```
Input: an LWE encryption $\underline{\mathbf{c}} = (\underline{\mathbf{a}}, \underline{b}) \in \mathrm{LWE}_{\underline{\mathbf{sk}},\underline{q}}(m)$,
Input: $2^{\vartheta}$ LUTs, each represented by $L_j$ encoding $f_j$, for $j \in [0, 2^{\vartheta}-1]$
Input: a bootstrapping key $\mathsf{BSK} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(X^{\underline{s}_i})_{i\in[0,n-1]}$,
Input: a set of automorphism keys $\mathsf{ATK}_{g^u} = \mathrm{RLWE}'_{\bar{\mathbf{sk}},\bar{Q}}(\bar{\mathbf{sk}}(X^{g^u}))_{u\in[1,w]}$,
Input: a auxiliary key $\mathsf{AUX} = \mathrm{RGSW}_{\bar{\mathbf{sk}},\bar{Q}}(X^{-\sum_i \underline{s}_i})_{i\in[0,n-1]}$.
Output: $\bar{\mathbf{c}}_j \in \mathrm{LWE}_{\bar{\mathbf{sk}},\bar{Q}}(f_j(m))_{j\in[0,2^{\vartheta}-1]}$.
1: $b = \left[\left\lfloor \frac{\underline{b}\cdot 2N\cdot 2^{-\vartheta}}{\underline{q}} \right\rceil \cdot 2^{\vartheta}\right]_{2N}$ and $a_i = \left[\left\lfloor \frac{\underline{a}_i\cdot 2N\cdot 2^{-\vartheta}}{\underline{q}} \right\rceil \cdot 2^{\vartheta}\right]_{2N}$
2: $\mathsf{testP} = X^{\frac{N}{2p}} \sum_{j=0}^{p-1} X^{j\frac{N}{p}} \sum_{k=0}^{\frac{N}{p2^{\vartheta}}-1} X^{k\cdot 2^{\vartheta}} \sum_{i=0}^{2^{\vartheta}-1} f_{i+1}(j) X^i$
3: $\mathsf{acc} = (0, X^{gb}\cdot \mathsf{testP}(X^g))$
4: $\mathsf{acc} = \mathsf{acc} \boxdot \mathsf{AUX}$
5: $[a_i = \underline{a}_i + 1]_{2N}$
6: $\mathsf{acc} = \mathsf{BR}_2(\mathbf{c} = (\mathbf{a}, b), \mathsf{acc}, \mathsf{BSK}, \mathsf{ATK})$
7: for $j = 0$ to $2^{\vartheta}-1$ do:
8:     $\bar{\mathbf{c}}_j = \mathsf{SampleExtract}_j(\mathsf{acc})$,
9:     return $\bar{\mathbf{c}}_j$
10: end for
```

D Analysis

We provide a detailed analysis of our AUTO variant circuit bootstrapping algorithm, and the analysis of the CMUX variant is similar.

Sparse Rounding Error.

Theorem 1. *Let n, q denote the dimension and the modulus of input LWE ciphertexts, respectively, $\bar{N}$ denote the ring polynomials dimension of RLWE ciphertexts. The error variance of the modulus switching result caused by the $k \cdot 2^{\vartheta} + 1$ sparse rounding is bounded by*

$$\sigma_{\mathsf{ms}}^2 = \frac{4\bar{N}^2\sigma_{\mathsf{in}}^2}{q^2} + (\underline{n}\sigma^2 + 1) \cdot (\frac{2^{2\vartheta}}{12} + \frac{2\bar{N}^2}{3q^2}) - \frac{\bar{N}^2}{q^2},$$

where σ_{in}^2 is the error variance of the input LWE ciphertext, and σ^2 is the error variance of the key distribution.

Proof. We consider the input ciphertext $\underline{\mathbf{c}} = (\underline{\mathbf{a}}, \underline{b}) \in \mathrm{LWE}_{\underline{\mathbf{sk}},\underline{q}}(m)$ satisfying $\underline{b} = -\sum_{i=0}^{n-1} \underline{a}_i \cdot \underline{s}_i + m + e$, the modulus switching algorithm with sparse rounding outputs a new ciphertext $\mathbf{c} = (\mathbf{a}, b)$, where $b = \lfloor \frac{2\bar{N}}{q}\underline{b} \rceil_{k\cdot 2^{\vartheta}+1}$ and $a_i = \left\lfloor \frac{2\bar{N}}{q}\underline{a}_i \right\rceil_{k\cdot 2^{\vartheta}+1}$.

Let $\left\lfloor \frac{2\bar{N}}{q}\underline{a}_i \right\rceil_{k\cdot 2^{\vartheta}+1} = \frac{2\bar{N}}{q}\underline{a}_i + a_i'$, then we have $a_i' \in \frac{2\bar{N}}{q} \cdot \mathrm{U}\left(\llbracket -2^{\vartheta-1}\frac{q}{2\bar{N}}, 2^{\vartheta-1}\frac{q}{2\bar{N}} \rrbracket\right)$. So $Var(a_i') = (\frac{2^{2\vartheta}}{12} - \frac{\bar{N}^2}{3q^2})$, $E(a_i') = -\frac{\bar{N}}{q}$. Since

$$\begin{aligned} b + \sum_{i=0}^{n-1} a_i \cdot s_i &= \frac{2\bar{N}}{q}\underline{b} + b' + \sum_{i=0}^{n-1}\left(\frac{2\bar{N}}{q}\underline{a}_i + a_i'\right) \cdot s_i \\ &= \frac{2\bar{N}}{q}\left(\underline{b} - \sum_{i=0}^{n-1} \underline{a}_i \cdot s_i\right) + b' + \sum_{i=0}^{n-1} a_i' \cdot s_i \\ &= \frac{2\bar{N}}{q}m + \frac{2\bar{N}}{q}e + b' + \sum_{i=0}^{n-1} a_i' \cdot s_i, \end{aligned}$$

then the error variance is,

$$\begin{aligned} \sigma_{\mathsf{ms}}^2 &= Var(\frac{2\bar{N}}{q}e + b' + \sum_{i=0}^{n-1} a_i' \cdot s_i) \\ &= \frac{4\bar{N}^2\sigma_{\mathsf{in}}^2}{q^2} + Var(b') + n \cdot Var\left(a_i'\right) \cdot \left(Var\left(s_i\right) + E^2\left(s_i\right)\right) + n \cdot E^2\left(a_i'\right) \cdot Var\left(s_i\right) \\ &= \frac{4\bar{N}^2\sigma_{\mathsf{in}}^2}{q^2} + (\underline{n}\sigma^2 + 1) \cdot (\frac{2^{2\vartheta}}{12} + \frac{2\bar{N}^2}{3q^2}) - \frac{\bar{N}^2}{q^2} \end{aligned}$$

Circuit Bootstrapping Error.

Theorem 2. *The parameters $\ell, \ell_{\mathsf{ep}}, \ell_{\mathsf{auto}}, \ell_{\mathsf{trace}}, \ell_{\mathsf{ss}}$ and $B, B_{\mathsf{ep}}, B_{\mathsf{auto}}, B_{\mathsf{trace}}, B_{\mathsf{ss}}$ denote the gadget decomposition length and base in the circuit evaluation, external product, automorphism, HomTrace, and scheme switching, respectively. $\varepsilon, \varepsilon_{\mathsf{ep}}, \varepsilon_{\mathsf{auto}}, \varepsilon_{\mathsf{trace}}, \varepsilon_{\mathsf{ss}}$ denote the corresponding decomposition error. The underline symbol $\underline{n}$ and $\underline{q}$ denote the dimension and modulus of the level 0 ciphertext, respectively, while $\bar{\bar{N}}$ and $\bar{Q}$ represent those of level 2. ϑ denotes the sparse rounding parameter, then the error variance of the result of the circuit bootstrapping algorithm is bounded by*

$$\begin{aligned}\sigma^2_{\mathsf{CBS}} \leq\ & \underline{n} \times \left(\frac{1}{6}\bar{N}\ell_{\mathsf{ep}}B^2_{\mathsf{ep}} \times \sigma^2 + \frac{1}{3}(\bar{N}+1)\varepsilon^2_{\mathsf{ep}}\right) \\ & + n_{\mathsf{auto}} \times \left(\frac{1}{12}\bar{N}\ell_{\mathsf{auto}}B^2_{\mathsf{auto}} \times \sigma^2 + \frac{1}{3} \times \frac{\bar{N}}{2}\varepsilon^2_{\mathsf{auto}}\right) \\ & + \frac{1}{3}\left(\bar{N}^2 - 1\right) \cdot \left(\frac{1}{12}\bar{N}\ell_{\mathsf{trace}}B^2_{\mathsf{trace}} \times \sigma^2 + \frac{1}{3} \times \frac{\bar{N}}{2}\varepsilon^2_{\mathsf{trace}}\right) \\ & + \left(\frac{1}{12}\bar{N}\ell_{\mathsf{ss}}B^2_{\mathsf{ss}} \times \sigma^2 + \frac{1}{3} \times \frac{\bar{N}^2}{4}\varepsilon^2_{\mathsf{ss}}\right)\end{aligned}$$

where σ^2 is the error variance of the keys.

Proof. The modulus switching procedure adds a sparse rounding noise bounded by σ^2_{ms}, but will be refreshed in the blind rotation.

The blind rotation procedure evaluates n_{auto} automorphisms, each requires a gadget product $\odot$, and also evaluates n external products, contains $2n$ $\odot$. From Lemma 1, this step adds gadget product noises bounded by

$$\underline{n}\times\left(\frac{1}{6}\bar{N}\ell_{\mathsf{ep}}B^2_{\mathsf{ep}} \times \sigma^2 + \frac{1}{3}(\bar{N}+1)\varepsilon^2_{\mathsf{ep}}\right) + n_{\mathsf{auto}}\times\left(\frac{1}{12}\bar{N}\ell_{\mathsf{auto}}B^2_{\mathsf{auto}} \times \sigma^2 + \frac{1}{3} \times \frac{\bar{N}}{2}\varepsilon^2_{\mathsf{auto}}\right).$$

The sample extraction does not add any noise.

The HomTrace evaluation has $\log \bar{N}$ iterations, the noise variance of the k-th iteration can be expressed as $\sigma^2_k \leq 4\sigma^2_{k-1} + \sigma^2_{\mathsf{auto}}$, thus the whole step adds noises bounded by

$$\left(1 + 4 + \cdots + 4^{\log \bar{N} - 1}\right)\sigma^2_{\mathsf{auto}} = \frac{1}{3}\left(\bar{N}^2 - 1\right)\cdot\left(\frac{1}{12}\bar{N}\ell_{\mathsf{trace}}B^2_{\mathsf{trace}} \times \sigma^2 + \frac{1}{3} \times \frac{\bar{N}}{2}\varepsilon^2_{\mathsf{trace}}\right).$$

The scheme switching requires one gadget product $\odot$, and adds gadget product noises bounded by

$$\frac{1}{12}\bar{N}\ell_{\mathsf{ss}}B^2_{\mathsf{ss}} \times \sigma^2 + \frac{1}{3} \times \frac{\bar{N}^2}{4}\varepsilon^2_{\mathsf{ss}}.$$

E Application

E.1 Transciphering

In the realm of data security, the concept of *transciphering* is a novel concept that leverages the strengths of symmetric encryption to address the transfer challenges posed by FHE. The crux of the problem lies in ciphertext expansion - the FHE ciphertext is significantly larger than the plaintext. This expansion can be a major hurdle, especially for devices with limited resources.

Transciphering, however, offers a solution. This method was first proposed by Naehrig et al[32]. The core idea is to use symmetric encryption for data transmission, which is then converted into homomorphic ciphertext by the server to do further evaluation. Specifically, the server receives the symmetric ciphertext $\mathcal{E}(m)$ and the homomorphic ciphertext of the symmetric key $Enc(k)$ from the client, and locally homomorphically evaluates the decryption circuit of the symmetric encryption to obtain the corresponding homomorphic ciphertext:

$$\mathsf{Eval}_{\mathcal{E}^{-1}}(Enc(k), \mathcal{E}(m)) = Enc(\mathcal{E}^{-1}(k, \mathcal{E}(m))) = Enc(m)$$

This approach can significantly reduce the transfer size of the ciphertext, making it more manageable for devices with limited bandwidth, memory, and computing power.

One of the most widely used encryption standards is the advanced encryption standard (AES)[35]. Known for its security and efficiency, AES is often the go-to choice for securing sensitive information. However, when it comes to transciphering, AES presents a unique challenge due to its high computational complexity. The issue of reducing the latency of homomorphic evaluation of the AES circuit during server pre-processing is a matter of academic consideration.

E.2 The Work Flow of The AES Encryption

AES-128 is the most widely used variant of AES, which encrypts 16-byte messages with a 16-byte key. The encryption process of AES-128 includes 10 rounds of calculations, each round function mainly consists of the following four operations: AddRoundKey, SubBytes, ShiftRows and MixColumns, see Alg.9. The details of each operation can be found in Fig.6.

Algorithm 9 AES Encryption and Decryption

```
AES Encryption:
AddRoundKey
for round = 1 to 9 do
    SubBytes
    ShiftRows
    MixColumns
    AddRoundKey
end for
SubBytes
ShiftRows
AddRoundKey
```

```
AES Decryption:
InvAddRoundKey
for round = 1 to 9 do
    InvShiftRows
    InvSubBytes
    InvAddRoundKey
    InvMixColumns
end for
InvShiftRows
InvSubBytes
InvAddRoundKey
```

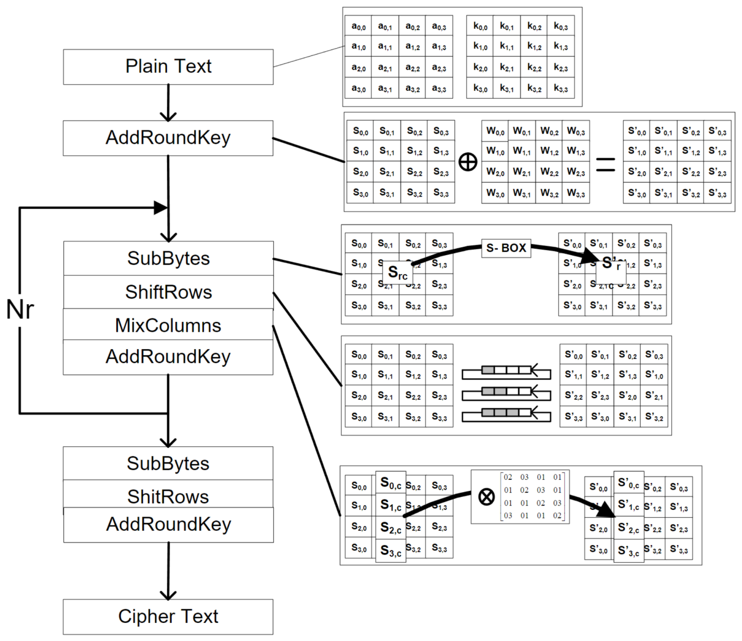


Fig. 6: The AES algorithm encryption process[31].

E.3 The Detailed AES Evaluation Methods

FHE Mode. In FHE mode, the 8-8 Sbox of the SubBytes is implemented by functional bootstrapping. Trama et al. demonstrated that the optimal strategy for AES homomorphic implementation is using 4-bit encryption, where bit-wise operations are unsupported. This necessitates converting not just SubBytes, but also AddRoundKey, ShiftRows, and MixColumns, to 8-8 LUTs (equivalent to 4 functional bootstrappings using tree-based LUT [11]). The entire AES evaluation requires 4,244 times 4-bit functional bootstrappings.

LHE Mode. LHE mode uses bit-wise encryption, simplifying AddRoundKey to mere ciphertext addition, ShiftRows to reordering ciphertexts, and MixColumns to addition and reordering, making these operations almost cost-free. Each 8-8 Sbox evaluation involves external product LUT (low-cost, detailed in [21]) and 8 circuit bootstrappings (corresponding to 8 input ciphertexts), totaling 1280 circuit bootstrappings.